CHALMERS | GÖTEBORGS UNIVERSITET

LaTeX-tips

En manual för matematikstudenter (version 2013)

Niklas Andersson och Malin Palö

Institutionen Matematiska vetenskaper vid Göteborgs universitet
och Chalmers tekniska högskola

Förord

Detta häfte är främst tänkt för dig som på egen hand vill bli bättre på att skriva dokument med hjälp av LaTeX inför arbeten i matematik eller andra naturvetenskapliga ämnen på universitetet.

LaTeX är ett väldigt smidigt verktyg när man gör rätt, men det kan lätt kännas hopplöst krångligt när saker inte blir som man tänkt sig. Lägg därtill att det inte finns någon officiell manual, som i vissa andra programmeringsspråk. Däremot finns det gott om manualer skrivna av personer från olika universitet som valt att rikta in sig på olika områden inom LaTeX. För oss, liksom för flera andra, tog det onödigt lång tid innan vi hittade den kunskap som idag känns viktig för oss. Det visade sig dessutom att vissa tips gått i arv mellan årskurser inom matematikprogrammet men aldrig skrivits ner.

Vårt mål med det här dokumentet är att försöka göra en LaTeX-manual som innehåller det vi tycker att man har störst nytta av att veta medan man skriver dokument i LaTeX, men som många inte känner till. Vi har ibland valt att nämna saker som är lätta att hitta på andra ställen, men framför allt försökt att skriva om funktioner som inte alls är lika lätta att söka sig fram till. Vi har dessutom försökt sammanställa det minimum av grunder i LaTeX som vi tror underlättar mycket att känna till för att med lätthet kunna nyttja de många resurser som finns att tillgå. Däremot så har vi medvetet inte tagit med många av de saker som kan tyckas vara grundläggande, men som är lätta att hitta online, utan har i stället försökt att göra en sammanställning av de funktioner som dels underlättar arbetet i sig, men också kraftigt höjer nivån på det färdiga resultatet. Vi har försökt ta med de saker som man sällan hittar i grundläggande manualer för LaTeX men som är lätta att använda om man känner till att de finns.

För de grundläggande saker vi har valt att inte ta upp i det här häftet, så som hur man infogar tabeller, bilder och dylikt, eller hur man skriver specifika matematiska tecken, rekommenderar vi den wiki som finns om LaTeX på `http://en.wikibooks.org/wiki/LaTeX`. Givetvis finns det även många andra liknande manualer på internet.

Det här dokumentet ses bäst i färg på en datorskärm. Vi har valt att göra det så för att dels kunna visa de effekter, och problem, färg i dokument kan bidra till, men också för att saker så som navigering i pdf-filer genererade av LaTeX fungerar väldigt bra i digitalt format. Dessutom kan ni som läsare då kopiera in källkod direkt från det här dokumentet till era filer. Vill man så går det

dock givetvis att skriva ut dokumentet också.

Vi vill avsluta det här förordet med att rikta ett stort tack till Thomas Ericsson för hjälp med korrekturläsning och förslag på förändringar!

Niklas Andersson och Malin Palö
Göteborgs Universitet och Chalmers Tekniska Högskola
Sommaren 2012

Innehåll

Kapitel 1

Grunderna i LaTeX

TeX, vilket är det verktyg som LaTeX bygger vidare på, är ett programmeringsspråk som används till att typsätta dokument på ett enhetligt och snyggt sätt. TeX skapades av Donald Knuth i slutet av 70-talet med syfte att underlätta och höja nivån på digital typsättning av dokument på datorer.

Eftersom att TeX är ett lågnivåspråk kräver det en hel del kunskaper och tid för att kunna användas. LaTeX är ett typsättningsverktyg skapat av Leslie Lamport som utvidgar TeX genom att dels uttöka funktionaliteten men även gör TeX betydligt lättare att använda. Inom ramarna för det vi kallar LaTeX finns också ett stort antal tilläggspaket som ytterligare utökar funktionaliteten. Vissa av dem följer idag med de flesta standardinstallationer av LaTeX, medan andra måste installeras separat.

LaTeX är alltså ett verktyg som kan användas för att typsätta dokument. Det skiljer sig från exempelvis *OpenOffice/Microsoft Word* genom att man i LaTeX skriver sitt dokument i en källkodsliknande text som sedan typsätts av en kompilator istället för att man ändrar direkt i det färdiga dokumentet (som man gör i en så kallad *WYSIWYG-editor*[1]). Detta tillvägagångssätt kan kännas som ett ovant och ologiskt sätt att skapa dokument på, men LaTeX är ett i många aspekter mycket kraftfullare verktyg än ett vanligt ordbehandlingsprogram. Några av anledningarna till detta är att det ger mycket större möjlighet att styra över det färdiga resultatet, samtidigt som det har många smidiga verktyg för att sköta typsättning och liknande av dokumentet, så att det slutgiltiga resultatet blir riktigt bra.

I följande avsnitt kommer vi kortfattat sammanfatta det mest grundläggande man behöver veta för att kunna komma igång med att använda LaTeX.

[1] What You See Is What You Get

1.1 Hello World! - ett första dokument

De dokument man gör skapas genom att man skriver in sin källkod i filer med ändelser `.tex`, vilka sedan används av kompilatorn för att skapa en pdf-fil.

Ett första dokument som resulterar i en pdf-fil innehållandes den klassiska programmeringsfrasen *Hello World* i ett eget dokument ser ut enligt följande:

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

\begin{document}
  Hello World
\end{document}
```

Den första raden specificerar vilken typ av dokument vi vill skapa. I det här fallet har vi angett att vi vill göra ett ensidigt dokument i A4-format där brödtexten har textstorlek 10pt. Inom måsvingarna specificerar vi att det är en artikel som vi skriver, genom att ange att vår *dokumentklass* skall vara dokumentklassen *article*. det finns ett föertal olika dokumentklasser att välja bland, och senare i det här dokumentet kommer vi att berätta mer om dokumentklassen *beamer*, som används för att skapa presentationer (se sid 84). Syftet med dokumentklasser är att styra utseendet på det kompilerade resultatet, och ofta även att lägga ytterligare funktionalitet, eller begränsningar, till LaTeX. I detta dokument kommer vi enbart att beskriva just dokumentklasserna *article* och *beamer*, men även exempelvis dokumentklassen *report* kan vara användbar i studiesammanhang.

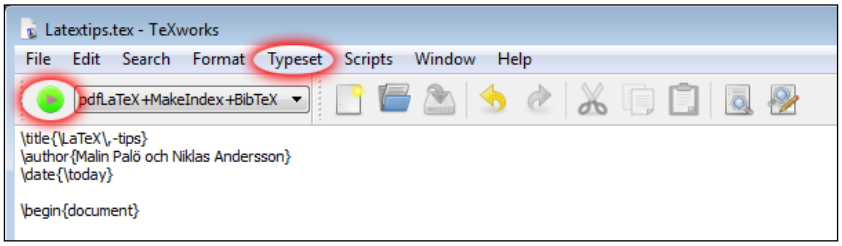
Kommandona `\begin{document}` och `\end{document}` markerar vart innehållet i vårt dokument ligger. I vårt fall innehåller detta enbart meningen *Hello World*. Koden placeras i ett dokument med namnet `HelloWorld.tex`, och denna kompileras sedan för att skapa själva dokumentet.

Området mellan kommandot `\documentclass[...]{...}` och kommandot `\begin{document}` kallas för dokumentets *preambel*. I preambeln lägger man olika inställningar, paketimporter och liknande. Mer information om exakt vad som kan och skall ligga i dokumentets preambel kommer senare i det här dokumentet. För tillfället räcker det att komma ihåg att det är viktigt att `\documentclass[...]{...}` alltid ligger överst i textfilen där vi skriver vår LaTeX-kod.

1.2 Kompilering

1.2.1 Kompilering i Windows

I Windows använder man ofta ett grafiskt användargränssnitt för att kompilera sitt dokument, snarare än att kompilera via en kombination av kommandon i terminalen, som i Linux. Ett exempel på en gratis sådan programvara är MikTEX[2]. MikTEXär en editor tillsammans med ett antal LATEX-kompilatorer, där man kompilerar sin fil genom att trycka på *Typeset* under menyn *Typeset*, eller på motsvarande symbol (se nedan). Om allt har gått bra (om man inte har några syntax-fel, som felstavade kommandon och liknande), så öppnas den resulterande pdf-filen i ett nytt fönster.



Man kan även installera LATEX separat, utan en tillhörande editor, i Windows och sedan använda exakt samma kommandon i kommandofönstret som beskrivs för Linux nedan.

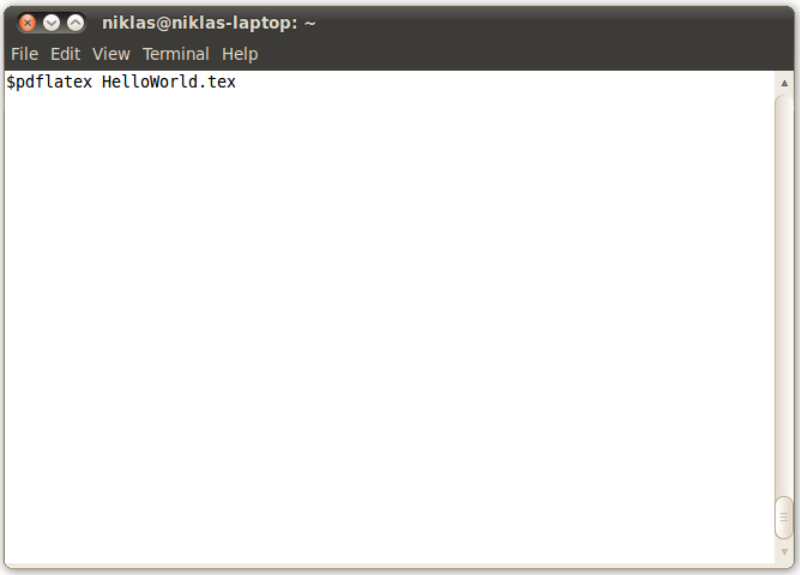
1.2.2 Kompilering i Linux

Kompilering av `tex`-filer i Linux gör enklast genom att man anropar den kompilator man använder från ett terminalfönster. För att kompilera filen `HelloWorld.tex` i Linux, gör följande:

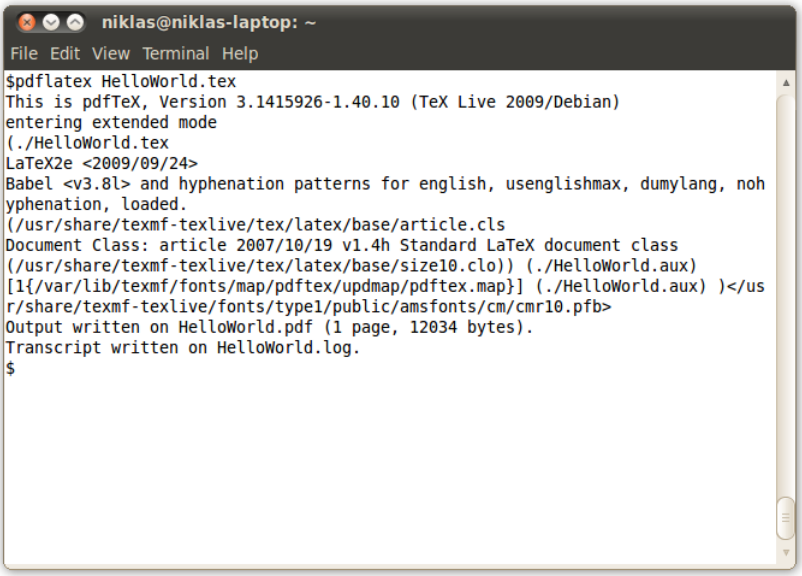
- Öppna en terminal, exempelvis genom att högerklicka på skrivbordet och välja ”Open terminal” i den meny som kommer upp.
- Gå till den mapp i vilken du har filen `HelloWorld.tex` [3].
- Ange kommandot `pdflatex HelloWorld.tex`.

[2] `http://miktex.org/download`

[3] Använd kommandot ”ls” för att visa allt innehåll i den mapp du står i och kommandot ”cd” för att gå till en mapp (som ligger i den mapp du för tillfället befinner dig i). För mer information, sök efter ”linux terminal file navigation” på internet.



Kommandot `pdflatex HelloWorld.tex` skapar filen `HelloWorld.pdf`. Att kompileringen genomfördes med ett lyckat resultat ser vi genom utskriften nedan.
`Output written on HelloWorld.pdf (1 page, 12034 bytes).`



1.2.3 Vanliga orsaker till kompileringsfel

Det finns ett flertal vanliga orsaker till kompileringsfel, varav de flesta nämns på följande sida: `http://en.wikibooks.org/wiki/LaTeX/Errors_and_Warnings`. Exempel på fel som är relativt lätta att korrigera är felstavade kommandon, paket som använts men ej finns på datorn, start och slut på omgivningar som inte matchas osv.

Ett annat fel som är lätt att göra är att av misstag använda ett så kallat *reserverat tecken*, dvs. ett tecken som har en speciell funktion i LaTeX. Om man använder något av dessa tecken direkt i sin text får man kompileringsfel, bortsett från %–tecknet som helt enkelt kommenterar bort all text som står efter det. Vilka dessa reserverade tecken är, och hur man gör för att använda dem i sin text visas i tabell 1.1.

Tecken	*Används till att ...*	*Kommando*
%	påbörja kommentarer	`\%`
$	påbörja eller avsluta matematiska uttryck	`\$`
\	markera att ett kommando börjar	`\backslash`
_	påbörja subskript i ekvationer	`\_{}`
{ och }	ange parametrar till LaTeX-kommandon	`\{` och `\}`
#	använda parametrar när man definierar egna kommandon	`\#`
^	påbörja superskript i ekvationer	`\^{}`
~	göra mellanrum som ej får brytas	`\~{}`
&	separera kolumner i tabeller	`\&`

Tabell 1.1: Tecken som kan ge problem i LaTeX; om något av tecknen i den första kolumnen används i ett sammanhang där de inte är en del av ett kommando så går det ofta inte att kompilera sin kod. Använd motsvarande teckenkombination ur den tredje kolumnen istället.

Ett annat väldigt vanligt fel för framför allt Windows-användare, som inte nämns lika ofta, är problem med dokumentformat.

UTF 8

I början av varje LaTeX-dokument behöver man ange vilken teckenkodning man använder. För att kunna använda svenska bokstäver så importerar man mer eller mindre alltid de två paketen *fontenc* respektive *inputenc*:

```
\usepackage[T1]{fontenc}
\usepackage[\smallcaps{utf}\,8]{inputenc}
```

Det första av paketen ovan; paketet *fontenc*, anger att vi vill använda tecknen i teckenpaketet T1 för att typsätta vårt dokument. Teckenpaketet T1 innehåller bland annat bokstäverna *å*, *ä* och *ö*. Även om man skriver dokument på engelska så bör man importera det här paketet för att exempelvis kunna skriva namn så som *Gödel*, *Hölder*, *Erdős*, *Möbius* och *Itō*.

Det andra av de två paketen ovan anger att formatet på alla filer med LaTeX-kod som dokumentet använder har teckenkodningen UTF 8. Om de inte har det, vilket ofta inte är standard i Windows, så går dokumentet som helhet oftast inte att kompilera, eller alternativt blir många av tecknen, och då speciellt å, ä och ö, fel. Ett typiskt felmeddelande när det här har hänt är:
`! Package inputenc Error: Unicode char \u8:??? not set up for use with LaTeX.`
Det enklaste att göra för att undvika det här felet är att redan innan man börjar skriva i en fil kontrollera att dess teckenkodning är just UTF 8. I Windows gör man enklast det här via den editor man valt att använda. I MikTeX anger man exempelvis teckenkodning via
Edit -> Preferences ... -> Editor -> Encoding, och i Scite via *File -> Encoding*.

1.3 Tilläggspaket

LaTeX består av ett grundprogram som innehåller dess grundläggande funktionalitet samt en mängd tilläggspaket som lägger till funktionalitet till LaTeX. Varje gång vi i det här dokumentet ber dig lägga till kommandot `\usepackage{***}` i ditt dokuments preambel så anges att ett visst paket skall användas. Funktionalitet som läggs till kan exempelvis vara möjligheten att ha färg på texten i dokumentet, att göra länkar klickbara eller att använda något särskilt typsnitt. Vissa av paketen kommer att följa med din grundinstallation av LaTeX, medan vissa måste installeras separat. Mer om hur man installerar extra paket i Linux kan du läsa här: `http://en.wikibooks.org/wiki/LaTeX/Packages/Installing_Extra_Packages`. Om du använder MikTeX i Windows så sker paketinstallationen automatiskt om ett paket du försöker använda inte finns på datorn, givet att du är anluten till internet.. Se även den med installationen följande programvaran *Package Manager*.

Det finns ett antal paket som man kan räkna med att alltid behöva importera för att kunna skriva en rapport av något slag. Dessa paket sköter saker så som teckenkodning, svenska bokstäver osv. Nedan listar vi det minimum av paketimporter som man mer eller mindre alltid vill göra.

fontenc `\usepackage[T1]{fontenc}` Möjliggör användning av svenska symboler.

inputenc `\usepackage[UTF8]{inputenc}` Anger vilket format källkodsfilerna har. Oftast vill vi ange att formatet är UTF 8, eftersom att vi då kan använda svenska tecken.

babel `\usepackage[swedish]{babel}` Anger vilket språk rubriker på exempelvis innehållsförteckningen skall vara på. När man skriver på engelska kan man helt bortse från den här importen eftersom att rubriker som standard blir på engelska.

amsfonts `\usepackage{amsfonts}` Innehåller matematiska symboler.

amsmath `\usepackage{amsmath}` Innehåller omgivningar och kommandon för att skriva matematiska formler.

För att kunna använda de funktioner som vi beskriver i det här dokumentet behövs importer av ett antal ytterligare paket. Hur detta går till beskrivs i samband med varje avsnitt.

För att underlätta för den som skriver sitt första LaTeX-dokument så har vi gjort et paket (en så kallad *stil*-fil), `minapaketochkommandon.sty`, vilken innehåller importer av de paket vi nämner i det här dokumentet. Det innebär att om ni väljer att använda vårt paket, så räcker det att ni använder nedanstående dokumentskal, dvs. ni kan då bortse från alla andra uppmaningar om att lägga kommandon i er preambel. Om du vill använda paketet så behöver filen `minapaketochkommandon.sty` ligga i samma mapp som den LaTeX-fil du kompilerar. Om du senare vill använda ett annat paket än de vi nämnt i det här dokumentet kan du antingen välja att lägga till det i din preambel eller i vår *.sty*-fil, eller alternativt göra en egen[4].

Om du väljer att använda vårt paket så behöver din preambel alltså nu bara se ut som den i exemplet nedan (här markerad med svart text), givet att filen `minapaketochkommandot.sty` ligger i samma mapp som det dokument du kompilerar.

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

\usepackage{minapaketochkommandon}

\begin{document}
Hello World
\end{document}
```

Till vårt paket finns det två möjliga parametrar, *swedish* och *english*. Om du inte anger någon parameter, vilket vi inte gjorde ovan, eller anger parametern *swedish*, så blir titlar för innehållsförteckningar, källförteckningar och namn på satser, korrelarium och liknande på svenska. Om parametern *english* används, liksom nedan, blir motsvarande saker automatiskt på engelska.

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

\usepackage[english]{minapaketochkommandon}

\begin{document}
Hello World
\end{document}
```

För mer information om stil-filer, se avsnitt 8.2.

[4]Se även avsnitt 8.2

1.4 Utdaterade kommandon

LaTeX som programmeringsspråk bygger på programmeringsspråket TeX, men lägger till fler kommandon och förbättrar en del TeX-kommandon. Till det tillkommer sedan ett flertal paket som man nästan alltid använder, så som exempelvis *amsmath*, vilka lägger till ytterligare kommandon. Både LaTeX, och de flesta paket finns dessutom i ett flertal olika versioner. I vissa fall finns det därför flera kommandon som gör nästan samma sak och som alla går att använda. Generellt sett rekommenderas att man använder så *nya* kommandon som möjligt; dels för att få ett så bra resultat som möjligt men också för att inte riskera att de LaTeX-filer man har inte går att kompilera efter att man uppdaterat sin LaTeX-distribution. Tabellen nedan samlar några kommandon som man av sådana skäl inte bör använda tillsammans med de kommandon man bör använda istället. En mer omfattande sammanställning av kommandon man bör undvika att använda finns här: `ftp://ftp.dante.de/tex-archive/info/l2tabu/english/l2tabuen.pdf`.

Kommando att undvika	*Kommando att använda istället*
`$ 1+2 $`	`\( 1+2 \)`
`$$ 1+2 $$`	`\[ 1+2 \]`
`{\bf malin}`	`\textbf{malin}`
`{\it malin}`	`\textit{malin}`
`{\cal F}`	`\mathcal{F}`

Kapitel 2

Rapporter

2.1 Vanliga kommandon i rapporter

I det här avsnittet listar vi snabbt de grundläggande kommandon som man behöver kunna för att kunna göra saker som rubriker, innehållsförteckningar, fotnoter och liknande i LaTeX. Du som redan har skrivit en del i LaTeX kan därför helt hoppa över det här avsnittet.

Vi visar i det här avsnittet bara de mest grundläggande kommandona för att göra de olika standardelementen i ett dokument. Givetvis går det typsatta resultatet att anpassa och ändra med hjälp av olika inställningar och paket i LaTeX, men detta tar vi inte upp här.

2.1.1 Rubriker

I LaTeX används kommandona `\section`, `\subsection` och `\subsubsection` för att skapa rubriker på olika nivåer. I dokumentet kan det exempelvis se ut som i exemplet nedan.

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

\begin{document}

\section{Huvudrubrik}
\subsection{En underrubrik}
\subsubsection{Ytterligare en rubrik}

\end{document}
```

2.1.2 En titelsida

De flesta dokument har en titel som finns allra först i dokumentet. Istället för att göra en titel med hjälp av en varnlig rubrik så finns ett speciellt kommando i LaTeX som är till just för att göra titlar. Nedan visas ett minimalt exempel.

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

\title{<Dokumentets titel>}
\author{<Författare>}
\date{<Datum>}

\begin{document}

\maketitle

\end{document}
```

Observera särskilt att kommandona `\title`, `\author` och `\date` ligger i dokumentets preambel, medan kommandot `\maketitle` inte gör det. I stället för att ange ett datum till kommandot `\date` ovan så kan man automatiskt använda dagens datum genom att använda kommandot `\today`, dvs. skriva `\date{\today}`. Datumet anpassas då automatiskt till det datum det är den dag du kompilerar dokumentet.

Resultatet av kommandona ovan syns nedan.

<Dokumentets titel>

<Författare>

<Datum>

2.1.3 Innehållsförteckning

För att göra en innehållsförteckning i LaTeX skriver lägger man kommandot `\tableofcontents` på det ställe i sitt dokument där man vill att den skall placeras. När man kompilerar dokumentet samlas då alla rubriker från dokumentet ihop och en innehållsförteckning skapas.

2.2 Att dela upp ett dokument i mindre delar

När man skriver långa dokument i LaTeX, exempelvis kandidatarbeten, så är det ofta en dålig idé att skriva all text i samma fil. Det finns nämligen många fördelar med att dela upp ditt dokument i flera mindre delar:

- Eftersom att all text skrivs som källkod, och det hela tiden är källkoden man arbetar med istället för det faktiska dokumentet, blir texten snabbt oöverskådlig om man har all text i samma fil. Om man delar upp den så blir det betydligt lättare att hitta i den.
- Om man under arbets gång vill byta plats på avsnitt är det mycket lättare att flytta på kommandona som läser in texten från en annan fil till huvuddokumentet än vad det är att flytta motsvarande text.
- Man får en bättre överblick av sin disposition.
- Om man är flera författare kan var och en skriva på sin del i en egen fil. Huvuddokumentet sammanfogar sedan alla författares delar och lägger dem i rätt ordning, vilket är lättare än att klippa och klistra text manuellt. Flera olika författare kan dessutom skriva samtidigt på varsin del.

Vi kommer nu att visa ett minimalt exempel för att det skall bli lätt att förstå hur man gör.

Antag att vi har följande text i en LaTeX-fil:

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

\section{Inledning}
Sed ut perspiciatis, unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam eaque ipsa, quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt, explicabo.

\section{Avslutning}
Nemo enim ipsam voluptatem, quia voluptas sit, aspernatur aut odit aut fugit,
sed quia consequuntur magni dolores eos, qui ratione voluptatem sequi nesciunt,
neque porro quisquam est, qui dolorem ipsum, quia dolor sit amet, consectetur,
adipisci[ng] velit, sed quia non numquam [do] eius modi tempora inci[di]dunt,
ut labore et dolore magnam aliquam quaerat voluptatem.

\end{document}
```

Vi skapar nu två helt nya dokument som vi lägger i samma mapp som vår huvudfil och sparar dem med namnen `inledning.tex` och `avslutning.tex`.

I filen `inledning.tex` lägger vi texten:

```
\section{Inledning}
Sed ut perspiciatis, unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam eaque ipsa, quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt, explicabo.
```

och på motsvarande sätt lägger vi följande text i filen `avslutning.tex`:

```
\section{Avslutning}
Nemo enim ipsam voluptatem, quia voluptas sit, aspernatur aut odit aut fugit,
sed quia consequuntur magni dolores eos, qui ratione voluptatem sequi nesciunt,
neque porro quisquam est, qui dolorem ipsum, quia dolor sit amet, consectetur,
adipisci[ng] velit, sed quia non numquam [do] eius modi tempora inci[di]dunt,
ut labore et dolore magnam aliquam quaerat voluptatem.
```

Det räcker då att vi i vår huvudfil skriver nedanstående text för att få exakt samma resultat som tidigare.

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

\input{inledning.tex}
\input{avslutning.tex}

\end{document}
```

När man kompilerar sin huvudfil så läses innehållet från filerna `inledning.tex` `avslutning.tex` och lägger det på det ställe där man skrivit `\input{inledning.tex}` och `\input{avslutning.tex}`. Man behöver därför bara kompilera huvuddokumentet, dvs. det dokument som innehåller kommandona `\begin{document}` och `\end{dokument}` så infogas all annan text automatiskt.

Generellt så gäller alltså att om vi lägger en del av vår kod i en fil med namnet *filnamn.tex* så kan vi byta ut motsvarande text i vår huvudfil mot kommandot `\input{filnamn.tex}`. Kommandot `\input{filnamn.tex}` läser innehållet från filen `filnamn.tex` och lägger det på motsvarande ställe i den fil man gett kommandot.

2.3 Kompilering av delar av ett dokument

Då man skriver långa dokument i LaTeX så kan det ibland vara skönt att arbeta med bara en del av dokumentet i taget för att slippa kompilera hela dokumentet när man egentligen bara arbetar med att litet avsnitt av det. Genom att använda kommandot `\input` (se föregående avsnitt) så

kan vi dela upp dokumentet i flera mindre delar, men inte kompilera varje del för sig. För att göra det så behöver vi använda paketet *subfiles*, och syftet med det här avsnittet är att visa hur det fungerar.

Vi kommer nu att återanvända samma exempel som vi använda i föregående avsnitt, men använder istället de tre filerna nedan:

`inledning.tex`:

```
\documentclass[huvudfil.tex]{subfiles}
\begin{document}

\section{Inledning}
Sed ut perspiciatis, unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam eaque ipsa, quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt, explicabo.

\end{document}
```

`avslutning.tex`:

```
\documentclass[huvudfil.tex]{subfiles}
\begin{document}

\section{Avslutning}
Nemo enim ipsam voluptatem, quia voluptas sit, aspernatur aut odit aut fugit,
sed quia consequuntur magni dolores eos, qui ratione voluptatem sequi nesciunt,
neque porro quisquam est, qui dolorem ipsum, quia dolor sit amet, consectetur,
adipisci[ng] velit, sed quia non numquam [do] eius modi tempora inci[di]dunt,
ut labore et dolore magnam aliquam quaerat voluptatem.

\end{document}
```

`huvudfil.tex`:

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

\begin{document}

\subfile{inledning.tex}
\subfile{avslutning.tex}
```

```
\end{document}
```

Det finns nu ett antal saker som ser lite annorlunda ut jämfört med när vi använde kommandot `\input` i föregående avsnitt:

1. Vi har angett att vi vill använda paketet *subfiles* genom att skriva `\usepackage{subfiles}` i vår preambel.
2. Vi har skrivit `\subfile` istället för `\input` i vår huvudfil för att inkludera innehållet i `avslutning.tex` och `avslutning.tex`.
3. Vi har med `\begin{document}` och `\end{document}` även i filerna `avslutning.tex` och `avslutning.tex`, dvs. de filer som innehåller text vi vill importera till huvuddokumentet
4. Högst upp i filerna `avslutning.tex` och `avslutning.tex` finns nu raden `\documentclass[huvudfil.tex]{subfiles}`, som framför allt anger vilken fil som är dess *huvudfil*.

Om vi nu kompilerar huvudfilen, ovan kallad `huvudfil.tex` så blir det ingen som helst skillnad från om vi hade haft all text i huvuddokumentet, eller alternativt använt kommandot `\input` istället. Den stora skillnaden mot att använda kommandot `\input` är dock att vi nu kan kompilera även delfilerna `avslutning.tex` och `avslutning.tex`. Den första raden `\documentclass[huvudfil.tex]{subfiles}` anger då att delfilen skall kompileras med innehållet i preamblen från huvudfilen, så vi behöver bara ange vilka paket vi vill använda där.

2.4 Mappar och relativa sökvägar

När man skriver ett dokument så ökar antalet filer som används i dokumentet snabbt då dokumentets storlek ökar. Det är därför viktigt att redan från början organiserar den mapp i vilken alla filer som hör till arbetet ligger på ett vettigt sätt. Eftersom att man kan ange relativa sökvägar i LaTeX behöver man inte ha alla filer i samma mapp. Om vi i rapporten, från filen *resultat.tex*, vill inkludera bilden som ligger i mappen *Bilder*, som i sin tur ligger i samma mapp som vår .tex-fil, skriver vi `\includegraphics{Bilder/bild1.png}`, dvs. vi kan använda relativa sökvägar.

Kapitel 3

Matematik

I det här avsnittet beskriver vi olika sätt på vilka man kan ange matematiska uttryck i sitt dokument. Observera att vi inte går igenom specifik syntax, eftersom att det finns mycket dokumentering av detta på internet[1], utan snarare går igenom de olika omgivningar man kan välja att använda för att skriva ekvationer, samt den grundläggande *grammatik* man behöver känna till för att förstå hur matematiska formler byggs upp i LaTeX.

LaTeX skiljer mellan två olika typer av matematiska ekvationer/uttryck; de som finns i flytande text och de som är på en egen rad. Vi kommer i det här avsnittet att börja med att gå igenom det minimum man behöver kunna för att kunna förstå hur matematiska uttryck byggs upp i LaTeX, för att sedan fortsätta med att gå igenom de olika sorters omgivningar man kan välja mellan och sedan avsluta med att visa hur man kan använda sådana omgivningar som en del i satser, korrelarium, lemman osv. som en del av sin text.

I samtliga avsnitt av detta dokument kommer vi att anta att paketen *amsmath* och *amsfonts* redan är inporterade till dokumentet, vilket du gör genom att skriva nedanstående kommandon i din preambel eller genom att importera vårt matematikpaket (se avnitt 8.2).

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

3.1 Matematisk grammatik i LaTeX

Matematiska uttryck i LaTeX kan sägas bestå av tre typer av kommandon; kommandon som anger att en symbol skall sättas in, och kommandon som anger vart symboler skall placeras och kommandon som formatterar symboler.

[1] Se exempelvis `http://en.wikibooks.org/wiki/LaTeX/Mathematics`.

De kommandon som anger att en symbol skall sättas in finns i två olika typer; de symboler som redan finns på tangentbordet skriver man helt enkelt genom att skriva den symbolen. Exempel på sådana symboler är paranteser, hakparanteser, *, siffror och vanliga bokstäver. Här behöver man dock vara uppmärksam på att de tecken som tas upp i tabell 1.1 inte går att använda. För dessa få man i stället använda namn, precis som för de flesta andra symboler. Denna andra typ av kommandon för tecken är de som inte finns med på vanliga tangentbord. Exempel på sådana tecken är $\sum$, $\ll$, $\approx$, α osv. I LaTeX skriver man dessa tecken genom att direkt efter tecknet \ skriva tecknets namn. De exempel vi just nämnts skrev vi exempelvis genom att skriva `\( \sum \), \( \ll \), \( \approx \), \( \alpha \)`. Om man söker efter *LaTeX symbols* på internet så hittar man många sidor med listor med symboler respektive vilka namn de har i LaTeX.

Nedanstående lista sammanställer några av de vanligaste placeringskommandona i LaTeX:

Tecken	*Syfte*	*Kodexempel*	*Exempel*
`^`	Superskript, dvs. när man vill skriva en liten siffra snett ovanför en annan siffra	`x^2`	x^2
`_`	Subskript, dvs. för att skriva en liten siffra snett nedanför en stor	`a_1, a_2`	a_1, a_2
`{}`	Används för att para ihop symboler, för att exempelvis markera att flera symboler skall vara nedsänkta tillsammans, eller att ett kommando skall tillämpas på en specifik grupp symboler	`x_{11}, x_{12}` (jämför med `x_12` (**fel**))	x_{11}, x_{12} (x_12)
`\frac{}{}`	Används för att skriva bråk	`\frac{x^2+1}{x-1}`	$\frac{x^2+1}{x-1}$

Formatteringskommandon är kommandon vars huvudsakliga syfte är att ändra stilen på det som just skrivits. Dessa kan användas för att exempelvis skriva $x \in \mathbb{R}$ i stället för $x \in R$ för att markera att x är ett reellt tal, eller för att kunna skriva de symboler som representerar vektorer med fet stil. Några vanliga sådana formatteringskommandon listas i tabellen nedan. Observera att vi i samtliga fall mellan måsvingarna anger vilka symboler det är som skall formatteras.

Tecken	*Syfte*	*Kodexempel*	*Exempel*
`\mathbf{}`	Fetstil i ekvationer	`\mathbf{u}`	$\mathbf{u}$
`\mathbb{}`	Det typsnitt som ofta används för vanliga mängder	`\mathbb{R}`	$\mathbb{R}$
`\mathcal{}`	Kalligrafisk stil, används exempelvis ofta i algebra	`\mathcal{F}`	$\mathcal{F}$

3.2 Ekvationer i löpande text

Om man vill skriva matematiska tecken eller en ekvation i flytande text så gör man det genom att skriva motsvarande matematiska text mellan de två tecknenkombinationerna `\(` och `\)`. Exempelvis så ger uttrycket `\( \pi \approx 3.1415 \)` resultatet $\pi \approx 3.1415$, vilket är den typ av uttryck som kan vara intressanta att ha med i flytande text. En ekvation som finns med i flytande text kallas i LaTeX-terminologi för en *inline equation*.

Ett annat sätt att ange att man vill ha en ekvation i flytande text visas nedan. Vilket av de två alternativen man väljer är helt och hållet en smaksak; för resultatet blir detsamma.

```
\begin{math}
\pi \approx 3.1415
\end{math}
```

Ett problem med att låta ekvationer stå med i löpande text är att LaTeX då kan dela upp dem på två rader om den tycker att det behövs, precis som de flesta ordbehandlare gör med vanliga ord. Ett exempel på när detta blir ett problem är nedanstående text:

(Wikipedia): Being an asymptotic formula, Stirling's approximation has the property that $\lim_{n\to\infty} \frac{n!}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n} = 1$

När ekvationer bryts på ställen där det inte alls var tänkt blir de svårlästa. För att undvika att det här händer kan man placera ett extra par måsvingar kring hela sin ekvation i källkoden, så kommer den garanterat inte att brytas:

```
(Wikipedia): Being an asymptotic formula, Stirling's approximation has the
property that
\( { \lim_{n \to \infty}
\frac{n!}{\sqrt{2 \pi n} \left( \frac{n}{e}\right)^n} = 1 }\)
```

Notera alltså det extra paret måsvingar precis efter `\(` respektive precis innan `\)`.

3.3 Ekvationer på en egen rad

Om man skall ha med större matematiska uttryck i sin text så blir texten oftast mer lättläst om man låter ekvationen stå på en egen rad. I LaTeX-terminologi kallas detta för *displayed math*. Det finns flera sätt att åstadkomma detta, men det finns i detta fall faktiska skillnader mellan de olika sätten.

Det första och enklaste sättet att skriva en ekvation på en egen rad är genom att skriva sin matematiska text mellan teckenkombinationerna `\[` och `\]`. Ett exempel på detta är uttrycket:

```
\[
  E[\chi] = \sum_mE[\chi_m] = \sum_m {1 \over 2} = {n \over 2}
\]
```

vilket ger resultatet

$$E[\chi] = \sum_m E[\chi_m] = \sum_m \frac{1}{2} = \frac{n}{2}$$

Observera att i samtliga fall nedan så hamnar uttrycket på en egen rad oavsett om man har det på en egen rad i .tex-filen eller ej.

Ett första alternativ till syntaxen ovan som ger exakt samma resultat är att skriva

```
\begin{displaymath}
  E[\chi] = \sum_mE[\chi_m] = \sum_m {1 \over 2} = {n \over 2}
\end{displaymath}
```

En nackdel med båda sätten ovan är att det blir knepigt om man vill kunna referera till något av uttrycken ovan senare i texten. Det är ju exempelvis vanligt i matematisk litteratur att man skriver *från exvation 4.7 följer att...* så vi skulle vilja ha något sätt att kunna göra det med den här typen av ekvationer. Genom att skriva *equation* i stället för *displaymath* så får vi resultatet

$$E[\chi] = \sum_m E[\chi_m] = \sum_m \frac{1}{2} = \frac{n}{2} \tag{3.1}$$

dvs. ett liten nummer inom parantes vid den högra marginalen. Ekvationen har nu blivit tilldelad ett nummer. Man skulle nu kunna referera till ekvationen genom att använda den siffran direkt i texten, men eftersom att man ibland lägger till en ny ekvation innan den aktuella ekvationen så kan det hända att ekvationen vid något senare tillfälle byter nummer, så det är ingen bra idé. Lösningen är att ge ekvationen ett eget namn. Vi skulle exempelvis kunna kalla vår ekvation för *ekv1*. Det är viktigt att vi inte har två ekvationer med samma namn och att namnet inte innehåller blanksteg. Vi tilldelar ekvationen namnet genom att ändra vår ekvation i källkoden till följande:

```
\begin{equation}
  E[\chi] = \sum_m E[\chi_m] = \sum_m {1 \over 2} = {n \over 2}
  \label{ekv1}
\end{equation}
```

När vi sedan vill referera till ekvationen skriver vi `\ref{ekv1}`. Vi kan då exempelvis skriva `ekvation \ref{ekv1}` för att få utskriften *ekvation 3.1*, där siffran ändras automatiskt om ekvationen skulle byta nummer.

Det finns dock en egenskap som *equation*-omgivningen saknar. Om man vill skriva långa ekvationer så behöver man ibland använda flera rader. En lösning skulle givetvis kunna vara att använda flera

equation-omgivningar efter varandra, men det är smidigare att använda omgivningen *split*:

```
\begin{equation}
  \begin{split}
    \mathbb{E}[\chi_v] = p + (1-p)(1 - p^{d_v} - (1-p)^{d_v}) \geq \\
     p + (1-p)(1 - p^{n^\epsilon} - (1-p)^{n^{\epsilon}}) = \\
     1 - (1-p)( p^{n^\epsilon} + (1-p)^{n^{\epsilon}})
  \end{split}
\end{equation}
```

Tecknen \\ anger att vi vill ha en radbrytning; och resultatet blir

$$\begin{aligned} \mathbb{E}[\chi_v] = p + (1-p)(1 - p^{d_v} - (1-p)^{d_v}) \geq \\ p + (1-p)(1 - p^{n^\epsilon} - (1-p)^{n^\epsilon}) = \\ 1 - (1-p)(p^{n^\epsilon} + (1-p)^{n^\epsilon}) \end{aligned} \tag{3.2}$$

Observera speciellt att vi bara får ett ekvationsnummer.

En annan fördel med att använda en *split*-omgivning istället för flera *equation*-omgivningar efter varandra är att vi kan använda &-tecknet för att styra hur ekvationerna placeras i sidled relativt varandra. Antag exempelvis att vi har följande uttryck:

$$\begin{aligned} \mu(A_i) = \\ \mu((A_i \cup B) \cup (A_i \backslash B)) = \\ \mu(A_i \cup B) + \mu(A_i \backslash B) \geq \\ \mu(A_i \cap B) = \\ \infty \end{aligned} \tag{3.3}$$

I LaTeX-filen motsvarar det följande kod:

```
\begin{equation}
  \begin{split}
    \mu(A_i) = \\
    \mu ((A_i \cup B) \cup (A_i \backslash B)) = \\
    \mu (A_i \cup B) + \mu (A_i \backslash B) \geq \\
    \mu (A_i \cap B) = \\
    \infty
  \end{split}
\end{equation}
```

Som standard blir de olika raderna inuti en *split*-omgivning automatiskt högerjusterade relativt varandra. Vi kanske istället vill att det första μ-tecknet på varje rad skall hamna under varandra;

vilket vi kan lösa genom att lägga till ett &-tecken framför varje μ. Kod respektive resultat blir då:

```
\begin{equation}
  \begin{split}
    &\mu(A_i) = \\
    &\mu ((A_i \cup B) \cup (A_i \backslash B)) = \\
    &\mu (A_i \cup B) + \mu (A_i \backslash B) \geq \\
    &\mu (A_i \cap B) = \\
    &\infty
  \end{split}
\end{equation}
```

$$\begin{aligned}
&\mu(A_i) = \\
&\mu((A_i \cup B) \cup (A_i \backslash B)) = \\
&\mu(A_i \cup B) + \mu(A_i \backslash B) \geq \\
&\mu(A_i \cap B) = \\
&\infty
\end{aligned} \tag{3.4}$$

Vi kan givetvis även välja att matcha positionerna av valfria tecken i ekvationen:

```
\begin{equation}
  \begin{split}
    \mu(A_i) &= \\
    \mu ((A_i \cup B) &\cup (A_i \backslash B)) = \\
    \mu (A_i \cup B) + \mu (A_i \backslash B) &\geq \\
    &\mu (A_i \cap B) = \\
    &\infty
  \end{split}
\end{equation}
```

$$\begin{aligned}
\mu(A_i) &= \\
\mu((A_i \cup B) \cup (A_i \backslash B)) &= \\
\mu(A_i \cup B) + \mu(A_i \backslash B) &\geq \\
\mu(A_i \cap B) &= \\
\infty
\end{aligned} \tag{3.5}$$

Om man vill använda *equation*-omgivningen men inte få radnummer så kan man skriva *equation** överallt istället för *equation*.

Den sista omgivning för matematisk text vi kommer att ta upp är omgivningen *align*. Skillnaden mellan den här omgivningen och omgivningen *equation* är att den här omgivningen är gjord för att kunna presentera flera olika ekvationer i samma omgivning, vilket gör att varje rad i en *align*-omgivning får ett eget ekvationsnummer. Ett exempel på ett fall när detta kan vara rimligt i fallet nedan, vars motsvarande LaTeX-kod precenteras nedanför.

$$\lim_{n\to\infty} \frac{n!}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n} = 1 \Leftrightarrow \tag{3.6}$$

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{3.7}$$

```
\begin{align}
  \lim_{n \to \infty} \frac{n!}{ \sqrt{2\pi n} \left( \frac{n}{e}\right)^n} =
       1 \Leftrightarrow \\
  n! \sim \sqrt{2 \pi n} \left( \frac{n}{e} \right)^n
\end{align}
```

Observera alltså att vi nu fått ett ekvationsnummer per rad. Om man vill bryta en lång ekvation inuti en alignomgivning så skall man, precis som i *equation*-omgivningen, placera en *split*-omgivning kring den ekvation man vill dela upp på två rader för att få rätt radavstånd och ett gemensamt ekvationsnummer.

Man kan även i en *align*-omgivning skriva `\nonumber` på en specifik rad precis före `\\` för att inte få ekvationsnummer på just den raden. Om man vill referera till, och därmed namnge, en specifik ekvation i en *align*-omgivning så placerar man `\label{ekvationsnamn}` precis före `\\`. Ett exempel på både rader utan en siffra och referenser till specifika rader i en *align*-omgivning ses nedan:

```
\begin{align}
  \lim_{n \to \infty} \frac{n!}{ \sqrt{2\pi n} \left( \frac{n}{e}\right)^n} =
       +1 \Leftrightarrow \nonumber \\
  n! \sim \sqrt{2 \pi n} \left( \frac{n}{e} \right)^n \label{rad1}
\end{align}
```

$$\lim_{n\to\infty} \frac{n!}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n} = 1 \Leftrightarrow$$

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{3.8}$$

Nu kan vi referera till ekvationerna på den andra raden ovan genom att skriva `ekvation \ref{rad1}`,

och får då resultatet *ekvation 3.8.*

3.4 Justering av storleken av paranteser

När man skriver matematiska uttryck i LaTeX så märker man ganska fort att det finns ett behov av att ha paranteser i olika storlekar. Det finns huvudsakligen två anledningar att använda mer än en storlek på paranteser; dels gör det ekvationerna mer läsbara om paranteser som inte hör ihop har olika storlek, men ekvationerna blir dessutom ganska mycket snyggare om paranteserna är lika höga som det uttryck som står mellan dem.

Det finns huvudsakligen två olika sätt att ändra parantesers storlek i LaTeX:

- ange att man vill ha just en parantes som är större
- låta LaTeX anpassa parantesens storlek automatiskt efter innehållet

För att ange att en vänsterparantes större; skriv `\bigl(`, `\Bigl(`, `\biggl(` eller `\Biggl(` beroende på hur stora paranteser du vill ha, och ange motsvarande `\bigr)`, `\Bigr)`, `\biggr)` eller `\Biggr)` för större högerparanteser. Motsvarande fungerar även med andra typer av paranteser så som hakparanteser och måsvingar; vi kan exempelvis skriva `\bigl\{`, `\Bigl\{`, `\biggl\{` och `\Biggl\{` eller `\bigl\[`, `\Bigl\[`, `\biggl\[` och `\Biggl\[`.

Om vi istället för att specificera storleken på en parantes med de kommandon som nämndes ovan istället skriver `\left(` respektive `\right)` så anpassas parantesernas storlek automatiskt efter det som står mellan dem. Detta gör att man, om man inte medvetet ändrar storleken på en parantes med hjälp av något av kommandona ovan, i princip alltid vill använda dessa kommandon i stället för att bara använda paranteser direkt; då anpassas storleken om det skulle behövas, och man behöver inte tänka på det alls medan man gör sin fil. Observera dock att den här typen av paranteser bara kan läggas in i par, dvs. filen kommer inte att gå att kompilera om det inte finns lika många höger- som vänster-paranteser på en rad i en ekvation.

Nedan ses några exempel på några bra och dåliga val av paranteser. För varje exempel visas först hur formeln blir om man bara använder en parantes utan ytterligare storlekskommandon. På samtliga övriga rader förutom den sista har vi låtit LaTeX justera storleken på paranteserna automatiskt med hjälp av kommandona `\left` och `\right`. På den sista raden har vi själva manuellt valt storleken på varje par av paranteser.

$$n! \sim \sqrt{2\pi n}(\frac{n}{e})^n$$

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

```
\[
  n! \sim \sqrt{2 \pi n} ( \frac{n}{e})^n
\]

\[
  n! \sim \sqrt{2 \pi n} \left( \frac{n}{e} \right)^n
\]
```

$$
  n! \sim \sqrt{2 \pi n} (\frac{n}{e})^n
$$

$$
  n! \sim \sqrt{2 \pi n} \left(\frac{n}{e} \right)^n
$$

```
\[
  \int_0^\infty \frac{-2x}{(1+x^2)^2} \, dx =
        [\frac{1}{1+x^2}]_0^\infty
\]

\[
  \int_0^\infty \frac{-2x}{(1+x^2)^2} \, dx =
        \left[\frac{1}{1+x^2} \right]_0^\infty
\]
```

$$
  \int_0^\infty \frac{-2x}{(1+x^2)^2} \, dx =
        [\frac{1}{1+x^2}]_0^\infty
$$

$$
  \int_0^\infty \frac{-2x}{(1+x^2)^2} \, dx =
        \left[\frac{1}{1+x^2} \right]_0^\infty
$$

```
\[
  y = 0 + \frac{d}{dx} (
    1 + 2\frac{d}{dx} (
      1 + 3\frac{d}{dx} (
        1 + 4\frac{d}{dx} (
```

$$
  y = 0 + \frac{d}{dx} (
    1 + 2\frac{d}{dx} (
      1 + 3\frac{d}{dx} (
        1 + 4\frac{d}{dx} (
$$

```
        1 + 5\frac{d}{dx} (
          \cdots
        )
      )
    )
  )
)
\]

\[
  y = 0 + \frac{d}{dx} \left(
    1 + 2\frac{d}{dx} \left(
      1 + 3\frac{d}{dx} \left(
        1 + 4\frac{d}{dx} \left(
          1 + 5\frac{d}{dx} \left(
            \cdots
          \right)
        \right)
      \right)
    \right)
  \right)
\]

\[
  y = 0 + \frac{d}{dx} \Biggl(
    1 + 2\frac{d}{dx} \biggl(
      1 + 3\frac{d}{dx} \Bigl(
        1 + 4\frac{d}{dx} \bigl(
          1 + 5\frac{d}{dx} (
            \cdots
          )
        \bigr)
      \Bigr)
    \biggr)
  \Biggr)
\]
```

3.5 Finjusteringar av avstånd

När man skriver matematiska formler i LaTeX så tolkas blanksteg som en symbol som separerar olika symboler som skall tolkas, och inte som ett faktiskt mellanrum. Om man vill ha ett mellanrum någonstans i en formel så behöver man därför specificera även det med hjälp av ett kommando. Det finns flera sådana kommandon att välja mellan beroende på hur mycket mellanrum man vill ha:

Mellanrumsstorlek	*Kommando*	*Resultat*
Negativt litet mellanrum	`$|\!|$`	$\|\!\|$
Inget mellanrum	`$||$`	$\|\|$
Litet mellanrum	`$|\,|$`	$\|\,\|$
Mer mellanrum	`$|\:|$`	$\|\:\|$
Mycket mellanrum	`$|\;|$`	$\|\;\|$
Mycket mer mellanrum	`$|\quad|$`	$\|\quad\|$
Mycket mycket mer mellanrum	`$|\qquad|$`	$\|\qquad\|$

Nedan ges exempel på när man kan behöva lägga till horisontella mellanrum manuellt i matematiska formler för att få ett riktigt bra resultat.

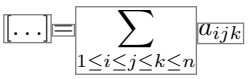
LaTeX-kod	*Uttryck med korrekta mellanrum*	*Uttryck utan extra mellanrum*
`\sqrt{2} \, x`	$\sqrt{2}\,x$	$\sqrt{2}x$
`x^3 \! / 2`	$x^3\!/2$	$x^3/2$
`\Gamma_{\! 2}`	$\Gamma_{\!2}$	Γ_2
`\int \! \! \! \int dx \, dy`	$\iint dx\,dy$	$\int\int dxdy$

En vanlig anledning till att man sätter in horisontella avstående i en formel manuellt är att man har gränser, till exempelvis en integral, som tar upp mycket plats. Om man bara vill göra en lite justering, som i exemplen ovan, är det oftast enklast att använda antingen något av kommandona ovan, eller kommandot `\hspace{}`. En nackdel med båda dessa, om man vill göra en större justering, är att det ofta krävs mycket testande för att få en ekvation att se *rätt* ut. Som exempel på det här problemet kommer vi att använda följande LaTeX-kod, vars resultat visas nedan.

```
\[
  [\ldots] = \sum_ {1 \leq i \leq j \leq k \leq n} a_{ijk}
\]
```

$$[\ldots] = \sum_{1 \leq i \leq j \leq k \leq n} a_{ijk}$$

Ovanstående ekvation, som är resultatet av vår exempelkod, har för stora mellanrum på båda sidorna av summan; ett resultat av att vårt subskript tar ganska mycket plats. För att se vad som har hänt ses nedan samma uttryck så som LaTeX ser det internt, LaTeX lagrar nämligen varje symbol som en ruta:



Det här gör att ju längre subskript vi har, desto längre bort kommer summanden, liksom uttrycket innan summan, att hamna.

Vår första lösning på problemet är att lägga till ett negativt horisontellt utrymme påbåda sidorna om summan. Nackdelen med den här metoden är framför allt att man måste testa sig fram till hur mycket negativt utrymme man behöver:

```
\[
  [\ldots] = \hspace{-0.8em} \sum_ {1 \leq i \leq j \leq k\leq n}
    \hspace{-0.8em} a_{ijk}
\]
```

$$[\ldots] = \sum_{1\leq i\leq j\leq k\leq n} a_{ijk} \qquad\qquad \boxed{[\ldots]} = \boxed{\sum_{1\leq i\leq j\leq k\leq n}} \boxed{a_{ijk}}$$

Observera nu att rutan som innehåller termen a_{ijk} nu har flyttat in över rutan som tidigare låg på dess vänstra sida, så att de delvis överlappar. Vi kan även göra motsvarande sak genom att använda det negativa utrymme som ges av kommandot `\!`. Koden med exakt samma resultat som oven visas nedan. Det här exemplet har exakt samma nackdel som i föregående exempel, men med den ytterligare nackdelen att de många små kommandona efter varande blir oöverskådliga.

```
\[
  [\ldots] = \!\!\!\!\! \sum_ {1 \leq i \leq j \leq k\leq n}
    \!\!\!\!\! a_{ijk}
\]
```

En tredje, och i det här fallet bättre lösning, är att använda kommandot `\mathclap{}`. Detta kommando läggs kring subindexet, och tar då bort dess horisontella bidrag till summans ruta. Nedan syns vårt exempel, liksom motsvarande resultat.

```
\[
  [\ldots] = \sum_ {\mathclap{1 \leq i \leq j \leq n}} a_{ijk}
\]
```

$$[\ldots] = \sum_{1\leq i\leq j\leq n} a_{ijk} \qquad\qquad \boxed{[\ldots]} = \boxed{\sum_{1\leq i\leq j\leq k\leq n}} \boxed{a_{ijk}}$$

Observera nu alltså att rutan som tidigare innehöll både summan och dess subindex nu har blivit mindre, så att den nu har samma höjd som tidigare, men bara tar hänsyn till bredden av summatecknet.

Ett liknande kommando, men som istället tar bort extra vertikalt avstånd är kommandot `\smash{}`. Det huvudsakliga användningsområdet för kommandot `\smash{}` är att ta bort de extra radavstånd som LaTeX lägger till i texter som innehåller *höga* matematiska uttryck. För att inte få olika radavstånd på olika rader i texten beroende på hur höga dessa uttryck är, lägger man kommandot `\smash{}` kring varje matematiskt uttryck som är för högt. Antag exempelvis att vi har följande kod:

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua
\( f(y) = \frac{1}{1+\frac{1}{y}} \). Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis
aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu
fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.
```

Om vi kompilerar den här koden direkt får vi följande resultat:

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua $f(y) = \frac{1}{1+\frac{1}{y}}$*. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.*

Vad som hänt är att LaTeX, för att kompensera för att vårt matematiska uttrcykt är lite för högt, har ökat radavståendet mellan rad 2 och rad 3, vilket kan störa intrycket en sida ger. För att undvika det här, kan vi lägga in den del av uttrycket som är för hög i kommandot `\smash{}`. Vår nya kod blir då följande:

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua
\( f(y) = \smash{\frac{1}{1+\frac{1}{y}}} \). Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu
fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum.
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua $f(y) = \frac{1}{1+\frac{1}{y}}$*. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.*

Vårt sista exempel på när man kan vilja finjustera matematiska avstånd hanterar det motsatta problemet, nämligen när man vill lägga till extra vertikalt avstånd i en matematiskt ekvation. När man använder *align-* eller *split*-omgivningar blir mellanrummet mellan raderna i ekvationerna lite för litet, så att innehållet i ekvationerna blir svårt att läsa. Exemplet nedan visar hur du kan lägga in lite extra mellanrum i en sådan ekvation.

```
\begin{align*}
    \lim_{n \to \infty} \frac{n!}{\displaystyle \sqrt{2\pi n}
      \left( \frac{n}{e}\right)^n} = 1 \Leftrightarrow
    \\[2mm]
    \lim_{n \to \infty} \frac{n!}{\displaystyle \sqrt{2\pi n}
      \left( \frac{n}{e}\right)^n} = 1
\end{align*}
```

3.6 Definitioner, satser, korrelarium, lemman, propositioner och bevis

Matematiska skrifter innehåller ofta definitioner, satser, bevis och liknande. Dessa finns det naturligtvis stöd för i LaTeX, och följande exempel visar hur man enkelt använder dessa.

För att kunna använda detta stöd måste man tala om för LaTeX vilka omgivningar man vill använda i sitt dokument. De vanligaste omgivningarna i matematisk texter är *theorem*, *definition*, *lemma*, *proposition*, och *corollary*. Dessa definieras innan `\begin{document}` med följande kommandon:

```
\usepackage{amsthm}
\newtheorem{theorem}{Sats}[section]
\newtheorem{definition}[theorem]{Definition}
\newtheorem{lemma}[theorem]{Lemma}
\newtheorem{proposition}[theorem]{Proposition}
\newtheorem{corollary}[theorem]{Korollarium}
```

Observera att vi i samtliga fall inom det första paret av måsvingar anger med vilket namn vi vill anropa kommandot, och inom det andra paret av måsvningar anger vilket namn vi vill skall visas i dokumentet för respektive omgivning (därav de svenska orden). Inom hakparanteserna anger vi hur de olika satserna,korrelariumen och definitionerna skall numreras; ovan har vi angett att satserna skall numreras efter vilket avsnitt de ligger i, så att den tredje satsen i avsnitt 2 får nummer 2.3, och att definitionerna, lemmana osv. skall använda samma räknare.

Ovanstående rader är allt som krävs för att enkelt kunna skapa definitioner, satser och lemman m.m. i ditt dokument. Vi visar med några exempel. Nedanför varje grå ruta står resultatet av den källkod som finns i rutan.

Vi börjar med ett exempel som visar hur man skriver en definition, och visar sedan exempel på en sats och ett korrelarium.

```
\begin{definition}
  Ett {\bf primtal} är ett heltal större än 1 som enbart har $\pm 1$ och
  $\pm p$ som delare.
\end{definition}
```

Definition 3.6.1. Ett **primtal** är ett heltal större än 1 som enbart har ± 1 och $\pm p$ som delare.

```
\begin{theorem}
  Om $p$ är ett primtal så är $\sqrt[n]{p}=p^{\frac{1}{n}}$ irrationellt för
  alla heltal $n \geq 2$.
\end{theorem}
```

Sats 3.6.2. *Om p är ett primtal så är $\sqrt[n]{p} = p^{\frac{1}{n}}$ irrationellt för alla heltal $n \geq 2$.*

```
\begin{corollary}
  $\sqrt{2}$ är irrationellt. Detta följer direkt av satsen med $n=p=2$
\end{corollary}
```

Korollarium 3.6.3. *$\sqrt{2}$ är irrationellt. Detta följer direkt av satsen ovan med $n = p = 2$*

Ofta vill man inte bara presentera påståenden, utan till dessa även ange bevis. Bevis lägger man till i sitt dokument genom att skriva:

```
\begin{theorem}
  Om \( p\) är ett primtal så är \( \sqrt[n]{p}=p^{\frac{1}{n}}\) irrationellt
  för alla heltal \( n \geq 2\).
\end{theorem}

\begin{proof}
  Antag att \( p^{\frac{1}{n}} \) är ett rationellt tal. Då finns det
  positiva heltal \( a\) och \( b\) med \(SGD(a,b)=1\) så att
  \[
    p^{\frac{1}{n}}=\frac{a}{b} \Leftrightarrow p=\frac{a^n}{b^n}
    \Leftrightarrow p b^n = a^n
  \]

  Eftersom \( p\) delar VL måste även \( p\) dela HL, dvs \( p \bar a^n\).
```

```
  Detta innebär att \( p \bar a\), dvs \( a= pm\) för något heltal \( m\),
  vilket ger att \( p b^n = a^n = (pm) ^n = p^n m ^n\), dvs.
  \( b^n = p^{n-1} m\), så \( p \bar b^n\). Men då är \( p\) en gemensam faktor
  för \( a\) och \( b\), vilket motsäger att \( SGD(a,b)=1\). Vårt antagande
  var därför felaktigt, och satsen är därmed bevisad.
\end{proof}
```

Sats 3.6.4. *Om p är ett primtal så är $\sqrt[n]{p} = p^{\frac{1}{n}}$ irrationellt för alla heltal $n \geq 2$.*

Bevis. Antag att $p^{\frac{1}{n}}$ är ett rationellt tal. Då finns det positiva heltal a och b med $SGD(a, b) = 1$ så att $p^{\frac{1}{n}} = \frac{a}{b} \Leftrightarrow p = \frac{a^n}{b^n} \Leftrightarrow pb^n = a^n$.

Eftersom p delar VL måste även p delar HL, dvs $p|a^n$. Detta innebär att $p|a$, dvs $a = pm$ för något heltal m. Detta ger att $pb^n = a^n = (pm)^n = p^n m^n$, dvs $b^n = p^{n-1}m$, dvs $p|b^n$. Men då är p en gemensam faktor för a och b, vilket motsäger att $SGD(a, b) = 1$. Vårt antagande var därför felaktigt, och satsen är därmed bevisad. □

Om man presenterar beviset direkt efter satsen så gör det ofta ingenting att bevisen inte numreras. Ibland har man dock lemman med bevis mellan en sats och dess bevis, och behöver därför ha någon form av numrering av bevisen. En lösning är då att skriva följande:

```
\begin{theorem}
  Om \( p\) är ett primtal så är \( \sqrt[n]{p}=p^{\frac{1}{n}}\) irrationellt
  för alla heltal \( n \geq 2\).
  \label{minsats}
\end{theorem}


\begin{proof}[Bevis av sats \ref{minsats}]
  Antag att \( p^{\frac{1}{n}} \) är ett rationellt tal. Då finns det
  positiva heltal \( a\) och \( b\) med \(SGD(a,b)=1\) så att
  \[
    p^{\frac{1}{n}}=\frac{a}{b} \Leftrightarrow p=\frac{a^n}{b^n}
    \Leftrightarrow p b^n = a^n
  \]

  Eftersom \( p\) delar VL måste även \( p\) dela HL, dvs \( p \bar a^n\).
  Detta innebär att \( p \bar a\), dvs \( a= pm\) för något heltal \( m\),
  vilket ger att \( p b^n = a^n = (pm) ^n = p^n m ^n\), dvs.
  \( b^n = p^{n-1} m\), så \( p \bar b^n\). Men då är \( p\) en gemensam faktor
  för \( a\) och \( b\), vilket motsäger att \( SGD(a,b)=1\). Vårt antagande
  var därför felaktigt, och satsen är därmed bevisad.
```

```
\end{proof}
```

Sats 3.6.5. *Om p är ett primtal så är $\sqrt[n]{p} = p^{\frac{1}{n}}$ irrationellt för alla heltal $n \geq 2$.*

Bevis av sats 3.6.5. Antag att $p^{\frac{1}{n}}$ är ett rationellt tal. Då finns det positiva heltal a och b med $SGD(a, b) = 1$ så att $p^{\frac{1}{n}} = \frac{a}{b} \Leftrightarrow p = \frac{a^n}{b^n} \Leftrightarrow pb^n = a^n$.

Eftersom p delar VL måste även p delar HL, dvs $p|a^n$. Detta innebär att $p|a$, dvs $a = pm$ för något heltal m. Detta ger att $pb^n = a^n = (pm)^n = p^n m^n$, dvs $b^n = p^{n-1}m$, dvs $p|b^n$. Men då är p en gemensam faktor för a och b, vilket motsäger att $SGD(a, b) = 1$. Vårt antagande var därför felaktigt, och satsen är därmed bevisad. □

Givetvis kan man göra motsvarande saker med korrelarium, lemman osv.

Kapitel 4

Referenser

De referenser man gör i skrift är oftast av något ett av följande slag:

- referenser till externa källor, exempelvis böcker, artiklar, hemsidor osv
- referenser inom det egna dokumentet

Referenser till externa källor används ofta när man hänvisningar till satser, bevis eller påståenden som inte anses vara allmänt kända. Behov av referenser inom det egna dokumentet kan uppstå exempelvis genom att man i löpande text vill diskutera innehållet i en figur eller en sats i sitt dokument.

4.1 Referenser inom dokumentet

4.1.1 Att märka och referera

Ofta vill man kunna hänvisa till satser/bevis/avsnitt/figurer/tabeller m.m. i sitt dokument, exempelvis genom att skriva *i figur 3 på sidan 8 såg vi att....* Detta blir naturligtvis väldigt omständigt att göra om man själv ska hålla koll på figurernas nummer och sidor, men som tur är finns det inbyggda kommandon för att göra detta på ett smidigt sätt.

Objekt som ska kunna refereras till måste märkas, och detta görs med kommandot `\label{namn}`, där `namn` är det namn du väljer att ge det du vill referera till. Försök att tilldela objekten beskrivande namn. Att kalla något *figur1* är en typiskt dålig idé, medan *figur:simulering-1* ofta är ett bättre namn för att du lätt ska kunna hålla koll på vilka referenser som pekar till vilka objekt.

I princip alla omgivningar (*figure, equation, align, definition, theorem* osv.) går att referera till, dvs. allt som ligger innanför `\begin{***}` och `\end{***}`, och `\label{referensnamn}` placeras

då inuti omgivningen, dvs. mellan `\begin{***}` och `\end{***}`. Observera dock att man för att kunna referera till en figur (omgivningen *figure*) måste ha en beskrivning under bilden/tabellen (fås med kommandot `\caption{figurebeskrivning}`) och placera kommandor `\label{***}` under kommandot `\caption{***}`.

När man refererar till ett avsnitt så lägger man kommandor `\label{***}` vartsomhelst i avsnittet.

De objekt som är markerade med en `label` kan sedan refereras till, och beroende på vilken information om objektet man vill ha används olika kommandon:

- Objektets nummer refereras till med kommandot `\ref{namn}`.
  Detta är lämpligt om man vill skriva *...i figur 1 ser vi....*
- Objektets sidnummer refereras till med kommandot `\pageref{namn}`.
  Detta är lämpligt om man vill skriva *... i figur 1 på sidan 5 såg vi....*

4.1.2 Smartare referenser

När man skriver riktigt långa dokument så samlar man snabbt på sig många omgivningar som man har gett ett namn med kommandot `\label{}`. Oftast är det helt oproblematiskt att använda kommandot `\ref`, men det bygger på att man håller koll på vad för typ av objekt man refererar till. Ett exempel på när detta blir särskilt problematiskt är när man använder paketet *amsthm*. Antag exempelvis att man i ett avsnitt har en sats, som har namnet `thm:min_sats`, och att vi har refererat till den vid ett flertal tillfällen i vårt dokument genom att skriva `sats \ref{thm:min_sats}`. Antag att vi senare bestämmer oss för att satsen istället skall vara en proposition. Vi måste då ändra ordet *sats* till *proposition* vid varje tillfälle var vi har refererat till satsen. Det är givetvis inte svårt att göra, men det finns en viss risk att man glömmer bort något ställe vid något tillfälle, och texten blir då svår att förstå. I det här avsnittet kommer vi att beskriva hur man använder paketet *cleveref*.

Vi kommer att börja med att beskriva hur du förbereder ditt dokument på att du skall använda paketet, dvs. vad som behöver stå i din preambel eller sty-fil, och därefter visa hur du använder paketet. I paketet som följer som bilaga till det här dokumentet har vi redan gjort alla förberedelser som krävs, så därför kan du hoppa över den första delen av det här avsnittet om du använder det.

Lägg nu följande i din huvudfils preambel (em du skriver så engelska så ta bort argumentet *swedish*):

```
\usepackage[swedish]{cleveref}
```

Om du inte använder paketet *amsthm* så är detta allt som krävs för att kunna använda paketet *cleveref*. Om du har använt paketet *amsthm*, och har definierat egna omgivningar med hjälp av kommandot `\newtheorem`, så behövs ytterligare lite kod i din preambel för att kommandona nedan ska fungera, oavsett om du skriver på svenska eller på engelska. Antag exempelvis att du har definierat en omgivning för korollarium i din preambel (se section 3.6):

```
\newtheorem{corollary}[theorem]{Korollarium}
```

Antag även att du använt den någonstans i ditt dokument genom att skriva:

```
\begin{corollary}
  Från definitionen av addition av binära tal följer direkt att

  \begin{align}
    0+0=0 \label{eq00}\\
    0+1=1 \label{eq01}\\
    1+0=1 \label{eq10}\\
    1+1=2 \label{eq11}
  \end{align}

  \label{mitt_korrelarium}
\end{corollary}
```

Korollarium 4.1.1. *Från definitionen av addition av binära tal följer direkt att*

$$0 + 0 = 0 \tag{4.1}$$
$$0 + 1 = 1 \tag{4.2}$$
$$1 + 0 = 1 \tag{4.3}$$
$$1 + 1 = 2 \tag{4.4}$$

Eftersom att omgivningen för korrelarium inte följer med paketet *amsthm* så måste vi berätta för paketet *cleveref* vad vår omgivning skall ha för namn när den refereras till (om vi inte gör det så kommer paketet *cleveref* att kalla alla korrelarium för satser (och motsvarande på engelska), vilket ju sällan är önskvärt). Lägg därför till följande kod i din preambel, efter koden ovan.

```
\crefname{corollary}{korollarium}{korollarium}
\Crefname{corollary}{Korollarium}{Korollarium}
```

Kommandona `\crefname` respektive `\Crefname` anger inställningar för vilka namn olika omgivningar skall ha i din text när du refererar till dem. Den första parametern till dessa kommandon anger vilken omgivning vi vill ange ett namn för. Den andra parametern anger namnet för omgivningen när vi vill referera till bara ett objekt och den tredje parametern anger vad namnet för omgivningen skall vara när vi refererar till flera objekt samtidigt när vi exempelvis vill skriva *korrolarium 1.3-1.5*. I båda fallet skall namnen som är parametrar till `\crefname` skrivas med bara gemener och namnen som är parametrar till `\Crefname` börja med versal.

Paketet cleveref ger oss nu tillgång till ett antal nya kommandon, vilka sammanfattas i table 4.1.

Kommando	*Exempelkod*	*Resultat*
`\cref`	`\cref{mitt_korrelarium}`	korollarium 4.1.1
	`\cref{eq00}`	equation (4.1)
	`\cref{eq00,eq01,eq10}`	equations (4.1) to (4.3)
	`\cref{eq00,eq01,eq11}`	equations (4.1), (4.2) and (4.4)
`\Cref`	`\Cref{min_tabell}`	Equation (4.1)
	`\Cref{eq00,eq01,eq10}`	Equations (4.1) to (4.3)
`\namecref`	`\namecref{mitt korrelarium}`	korollarium
	`\namecref{eq00}`	equation
`\nameCref`	`\nameCref{mitt korrelarium}`	Korollarium
	`\nameCref{eq00}`	Equation

Tabell 4.1: Exempel på kommandon du kan använda om du använder paketet *cleveref*.

4.2 Citering och källförteckning

Skapa en fil med ändelsen *.bib* där du placerar information om de externa källor du refererar till i din text. Filen kan exempelvis se ut som följande:

```
@BOOK{referens1,
   author = "N. J. A. Sloane and Simon Plouffe",
   title = "The Encyclopedia of Integer Sequences",
   publisher = "Academic Press",
   year = "1995"
}

@article{roginskaya,
   author = "Maria Roginskaya",
   title = "Asymptotic properties of harmonic and {M}-harmonic functions
              on the boundary of the unit ball",
   year = "2003",
   journal = "Journal of {M}athematical {S}ciences",
   volume = "115",
   number = "2",
}
```

Hänvisningar till referenserna från din text görs med kommandot `\cite{referensnamn}`. Om du exempelvis vill referera till *The Encyclopedia of Integer Sequences* som ligger i din `.bib`-fil enligt ovan skriver du `\cite{referens1}`, vilket ger följande utskrift i den löpande texten: [2], medan kommandot `\cite{roginskaya}` ger utskriften [1].

Notera att namnen *referens1* respektive *roginskaya* kan vara vilka namn du vill. Ett bra tips är att döpa referenserna så namnen säger något om vilken referens det är - det blir lättare att hålla koll på då!

Där du vill infoga referenslistan i rapporten skriver du följande:

```
% Det utseende/vilken standard för referenserna som ska användas
\bibliographystyle{amsplain}

% Namnet på .bib-filen, fast utan filändelsen:
\bibliography{filnamn}
```

För att kompilera och få alla referenser rätt i **Linux**, ange följande kommandon:

```
pdflatex dokumentnamn.tex
bibtex dokumentnamn.aux
```

```
pdflatex dokumentnamn.tex
pdflatex dokumentnamn.tex
```

Första `pdflatex`-kommandot kompilerar dokumentet och skapar bl.a. filerna `dokumentnamn.pdf` och `dokumentnamn.aux`. Filen `dokumentnamn.aux` innehåller namnen på de referenser som använts. Kommandot `bibtex dokumentnamn.aux` kopplar samman referensnamnen från `dokumentnamn.aux` med informationen om motsvarande referenser från `dokumentnamn.bib`-filen och lagrar denna i filen `dokumentnamn.bbl`. Det tredje kommandot, `pdflatex dokumentnamn.tex`, sätter ut siffror/bokstäver på de ställen där man refererar i texten, och det sista kommandot korrigerar sidnummer i de fall de förekommer samt formaterar referenslistan och infogar den.

I de flesta **Windows-program** som sköter kompileringen kan vi välja att även kompilera med *bibtex*. Om man använder Windows och har installerat LaTeX kan man även lösa det här genom att ange följande kommando i kommandoprompen, i den mapp där filet `dokumentnamn.tex` ligger:

```
texify dokumentnamn.tex
```

Resultatet av kompileringen ovan blir nu:

Litteraturförteckning

[1] Maria Roginskaya, *Asymptotic properties of harmonic and M-harmonic functions on the boundary of the unit ball*, Journal of Mathematical Sciences **115** (2003), no. 2.

[2] N. J. A. Sloane and Simon Plouffe, *The encyclopedia of integer sequences*, Academic Press, 1995.

Notera att om man inte bryr sig om att få referenserna rätt, exempelvis om man håller på och ändrar en ekvation och bara vill se till så att den ser bra ut, så kan man givetvis enbart kompilera med kommandot `pdflatex dokumentnamn.tex` en gång. Det enda som händer då är att referenserna inte kommer att hamna rätt, men allt annat typsätts som det ska. Detta är tidssparande vid kompilering av stora arbeten som innehåller många referenser: det finns ingen anledning att köra flera kommandon så att alla referenser hamnar rätt om man bara ska se till att en figur hamnar rätt.

Om man är flera författare som skriver varsin del av ett arbete har varje författare antagligen egna referenser för varje del. Varje person kan då lagra sina referenser i en egen .bib-fil, och dessa sammanfogas sedan av latexkompilatorn. Givet att tre personer har skrivit varsin del av rapporten

och har sina referenser i filerna `ref1.bib`, `ref2.bib` respektive `ref3.bib` skriver man följande för att få referenslistan korrekt:

```
\bibliographystyle{amsplain}
\bibliography{ref1,ref2,ref3}
```

Detta fungerar precis som man förväntar sig: latexkompilatorn letar efter referenser i filerna `ref1.bib`, `ref2.bib` och `ref3.bib`, och sammanfogar dessa till en referenslista. Notera även att ordningen på referenserna i bib-filerna inte spelar någon som helst roll. Referenslistan kommer att sortera och numrera referenserna efter den ordning de förekommer i rapporten.

Beroende på om det ni refererar till är en bok, artikel, osv så sparar man informationen i sin ḃib-fil på olika sätt. Nedan listar vi de vanligaste referenssorterna. Till varje referenssort finns det några fält som är obligatoriska och några som man inte behöver använda (`http://en.wikibooks.org/wiki/LaTeX/Bibliography_Management`).

Det finns många olika layouter man kan välja att ha på sina referenser, och vilken layout man vill ha anges med kommandot: `\bibliographystyle{amsplain}`. I exemplet ovan använde vi layouten `amsplain`, men det finns många fler att välja på - sök på internet efter *latex bibliography styles*. Ofta har olika universitet egna regler som styr vilka layouter man får lov att välja bland.

4.3 Nomenklaturlista

När man skriver ett långt dokument med många specialsymboler kan det vara smidigt att ha en lista med alla notationer man använder.

För att kunna lägga till en nomenklaturlista i ditt dokument; lägg följande kommandon i din preambel[1]

```
\usepackage[refpage, noprefix]{nomencl}
\usepackage{makeidx}

% Anger vad vi vill ha för rubrik på nomenklaturlistan
\renewcommand{\nomname}{Nomenklaturlista}

% Definitionerna nedan gör att nomenklaturlistan ges precis
% samma utformning som innehållsförteckningen

\makeatletter
\newcommand \Dotfill {
  \leavevmode \leaders \hb@xt@ .75em{\hss .\hss }\hfill \kern \z@
```

[1] Om du använder vårt paket *mina_paket* så behöver du inte göra det, eftersom att vi då redan gjort det åt dig.

```
}
\makeatother

\renewcommand*{\pagedeclaration}[1]{
  \unskip\Dotfill \makebox[6mm][r]{\hyperpage{#1}
}

% Ser till att kompilatorn sparar information om när vi använt kommandot
% \nomenclature
\makenomenclature
\makeindex
```

När vi nu introducerar nya kommandon i vår text så kan vi använda kommandot `\nomenclature` för att se till att en definiton hamnar i vår nomenklaturlista. Kommandot `\nomenclature` har totalt tre parametrar:

- Den första parametern anger hur notationen skall sorteras i nomenklaturlistan. Den parametern behövs eftersom att paketet annars skiljer på grekiska bokstäver, vanliga bokstäver, mathematiska symboler osv. I exemplet nedan har vi därför angett att bokstaven $\mathbb{N}$ skall sorteras som om den vore bokstaven N.
- Den andra parametern anger vilken symbol som det är vi förklarar. I fallet med $\mathbb{N}$ skriver vi därför `\( \mathbb{N} \)`, vilket ju är LaTeX-syntax för att visa bokstaven $\mathbb{N}$ i vårt dokument.
- Den tredje parametern innehåller en kort definition av symbolen som står som andra parameter.

Vi visar nu ett litet exempel.

```
\nomenclature[Z]{\( \mathbb{Z} \)}{De hela talen} Vi kommer att använda
  $\mathbb{Z}$ för att beteckna de hela talen.

\nomenclature[N]{\( \mathbb{N} \)}{De naturliga talen} Vi kommer att använda
  $\mathbb{N}$ för att beteckna de naturliga talen.
```

I vårt dokument syns det inte alls att vi använt kommandot `\nomenclature`, utan vi ser bara det som står utanför det kommandots parametrar:

Vi kommer att använda $\mathbb{Z}$ för att beteckna de hela talen.

Vi kommer att använda $\mathbb{N}$ för att beteckna de naturliga talen.

Notera särskilt att det inte syns att man använt kommandot `\nomenclature` i pdf-filen. Det är dock ändå viktigt att man verkligen lägger kommandot `\nomenclature` på det ställe i sin text

där man definierar symbolen, eftersom att man i nomenklaturlistan får en sidhänvisning till vart i dokumentet symbolen definieras mer ordentligt.

Skriv nedansående på den plats i ditt dokument när du vill att din nomenklaturlista skall ligga.

```
\printnomenclature[2cm]
```

För att få nomenklaturlistan att synas i ditt dokument, kompilera dokumentet i **Linux** genom att ange följande kommandon:

```
pdflatex huvudfil.tex
bibtex huvudfil.aux
makeindex huvudfil.nlo -s nomencl.ist -o huvudfl.nls
pdflatex huvudfil.tex
pdflatex huvudfil.tex
```

Notera alltså särskilt att den tredje raden har tillkommit jämfört med de gånger vi kompilerat via terminalen tidigare.

Om man brukar kompilera sina dokument via programmet MikTEX/ TEXWorks eller TEX Live i Windows (eller Linux) så kan man kompilera nomenklaturlistan helt automatiskt, givet att koden nedan läggs till i preambeln, precis innan `\makenomenclature`, i huvudfilen, som vi antar har namnet `huvudfil.tex`, eller på motsvarande ställe i din sty-fil.

```
\def\execute{%
\begingroup
\catcode`\%=12
\catcode`\\=12
\executeaux}
\def\executeaux#1{\immediate\write18{#1}\endgroup}

\execute{makeindex huvudfil.nlo -s nomencl.ist -o huvudfil.nls}

\makenomenclature
\makeindex
```

Notera att vi alltså på två olika ställen anger namnet på vår huvudfil, och att det måste ändras om vi byter namn på vår huvudfil. Notera också att argumentet till kommandot `\execute` är precis samma rad som den tredje raden i den sekvens kommandon vi använde ovan för att kompilera vårt dokument. När koden ovan ligger i vår preambel/sty-fil räcker det att kompilera dokumentet med kompilatorn *pdflatex* via MikTEX/TEXWorks grafiska gränssnitt för att även nomenklaturlistan skall kompileras.

Vår nomenklaturlista är nu med i vårt dokument och ser ut enligt följande:

Nomenklaturlista

Observera särskilt att symbolerna i vår nomenklaturlista är sorterade efter vad vi angav att de skulle sorteras efter, och inte efter i vilken ordning de dyker upp i dokumentet.

Kapitel 5

Layout

I det här avsnittet nämner vi ett antal olika knep som gör dokumentet lite snyggare, och framför allt tips som kan fixa några av de saker som kan vara lite problematiska när man arbetar med LaTeX.

5.1 Positionering av bilder

Det här stycket kommer att förutsätta att du redan har en bild infogad med hjälp av följande kommandon:

```
\begin{figure}[]
  \includegraphics{Bilder/bild.png}
\end{figure}
```

Ett vanligt problem som många har när de börjar arbeta med LaTeX är att de ej har kontroll över vart bilderna hamnar. När man infogar en bild så försöker LaTeX att placera bilden på ett så *bra* ställe som möjligt. Anledningar till att LaTeX flyttar på en bild kan exempelvis vara att undvika vitt utrymme. Det kan dock ibland innebära att bilder hamnar på helt andra ställen än man tänkt sig, och ibland dessutom i en annan ordning än den man tänkt sig. Innanför klamrarna efter `\begin{figure}` kan man föreslå var bilder skall placeras; om man vill ha dem på en egen sida (p); i närheten av vart man placerade dem i källkoden (h), högst upp på sidan (t) eller längst ner på sidan (b). Det är dock ingen garanti för att det är där bilderna faktiskt placeras...

En, *fungerande*, lösning är att importera paketet *float*, dvs. skriv:
`\usepackage{float}` innan `\begin{document}` och modifiera sedan *figure*-omgivningen ovan till följande:

```
\begin{figure}[H]
  \includegraphics{Bilder/bild.png}
\end{figure}
```

Observera att vi har använt ett stort H här, dvs. det här är inte samma kommando som h

Använd dock *H* sparsamt och endast när arbetet är nästan klart. Anledningen till att man inte bör göra det direkt är att det faktiskt finns poänger med att inte specificera ekakt var en bild skall hamna. Om du exempelvis lägger till mer text innan bilden så kanske *just precis här* inte är en jättebra placering av bilden längre. En av poängerna med att använda just LaTeX är ju att LaTeX gör den här typen av typsättning åt en, och oftast riktigt snyggt. Ofta blir dessutom slutresultatet bättre om man väljer att låta LaTeX bestämma vart figurerna skall vara och refererar till dem genom att skriva *i figur 2.3 ser vi att [...]* istället för att skriva *i bilden nedan ser vi att [...].*

5.2 Radbrytningar

LaTeX bryter rader där LaTeX tycker att det passar bäst. Nedan listas ett antal fall när vad LaTeX tycker är bäst inte blir så snyggt, och hur man kan se till att det inte händer.

Definitiv radbrytning (\\)
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Genom att använda kommandot \\ så kan vi ange precis vart raderna skall brytas:

```
Lorem ipsum dolor sit amet,\\
consectetur adipisicing elit,\\
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.\\
Ut enim ad minim veniam,\\
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat.
```

Lorem ipsum dolor sit amet,
consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Att undvika radbrytning (~)
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut ***figur***

1 *labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.*

Det är inte särskilt snyggt att *figur* och *1* hamnar på olika rader. För att undvika det så kan vi använda specialsymbolen ~ (tilde) som i LaTeXmarkerar ett mellanrum där en radbrytning inte får placeras. Genom att skriva `figur~\ref{figurnamn}` i stället för `figur \ref{figurnamn}` så talar vi om att raden inte får brytas där, och får resultatet:

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut ***figur 1*** *labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.*

Det är alltså en bra vana att alltid skriva ~ istället för ett vanligt mellanrum när man refererar.

Tillåt radbrytning (`\-`)
Unus duo tres quattuor quinque sex septem octo novem decem undecim duodecim tredecim ***quattrodecim*** *quindecim septendecim duedeveginti uneviginti viginti*

När LaTeX avgör hur många ord som skall vara på en given rad innan radbrytning så försöker den i första hand att inte bryta ord alls, men gör det ibland om man placerar ett långt ord där ett radbyte normalt sett placerats. Anledningen till det är att man vill undvika att placera vad som estetiskt sett kan anses vara för få ord på någon rad. För att förebygga problemet, eller alternativt, fixa det när man ser att det skett, så kan man använda kommandot `\-`. I kodexemplet nedan har vi angett att vi föredrar att radbrytning sker mellan de två stavelserna *quattro* respektive *decim*.

```
Unus duo tres quattuor quinque sex septem octo novem decem undecim duodecim
tredecim quattro\-decim quindecim septendecim duedeveginti uneviginti viginti
```

Unus duo tres quattuor quinque sex septem octo novem decem undecim duodecim tredecim ***quattrodecim*** *quindecim septendecim duedeveginti uneviginti viginti*

5.3 Sidbrytning

För att få en ny sida, skriver man oftast `\newpage`. Det finns dock ett problem med det; eftersom att vi inte kan styra exakt vart figurer hamnar i LaTeX så finns det en risk att en figur hamnar långt efter vart man avsett placera den. Oftast vill man ju exempelvis inte att bilder som hör till ett avsnitt skall hamna under nästa avsnitt. Genom att skriva `\clearpage` i stället så får man en ny sida, men tvingar dessutom LaTeX att placera alla figurer man hittills använt innan den nya sidan.

5.4 Mellanrum mellan olika stycken

Som standard använder LaTeX en ny rad och en indentering för att markera att ett nytt stycke börjar. Det gör dock att en rapport kan se väldigt kompakt ut. För att ta bort indenteringen och istället markera att nytt stycke med en blank rad så är det enklast att importera paketet *parskip* genom att skriva följande precis innan `\begin{document}`:

```
\usepackage[parfill]{parskip}
```

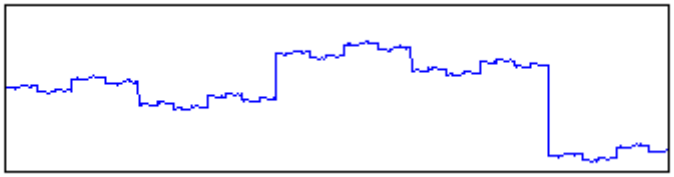
5.5 Horisontella och vertikala mellanrum i dokumentet

Det finns huvudsakligen två orsaker till att man ibland vill använda manuella horisontella eller vertikala mellanrum i sitt dokument:

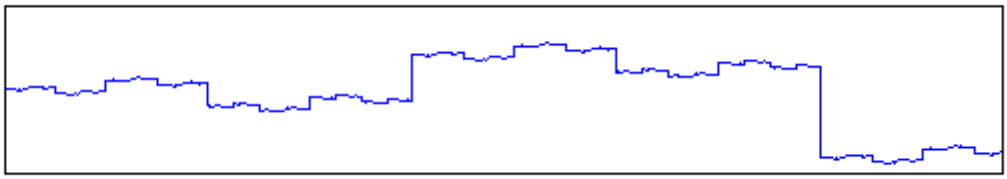
1. Av rent typografiska skäl, dvs. för att man vill åstadkomma en effekt.
2. För att fixa typsättning som inte blev så snygg när dokumentet kompilerades.

Horisontella mellanrum görs med kommandot `\hspace{}`, var vi innanför klamrarna anger hur mycket utrymme vi vill ha. Kommandon `\vspace{}`, för vertikala mellanrum, fungerar på precis samma sätt. Avståndet kan anges i ett antal olika enheter, bland annat mm, cm, pt[1] och px[2].

Ett exempel där det kan vara smidigt att använda kommandot `\hspace{}` är när man vill centrera en lite större bild i sitt dokument. Genom att lägga till kommandot `\centering` inuti en *figure*-omgivning så centreras en bild som är *tillräckligt liten*:

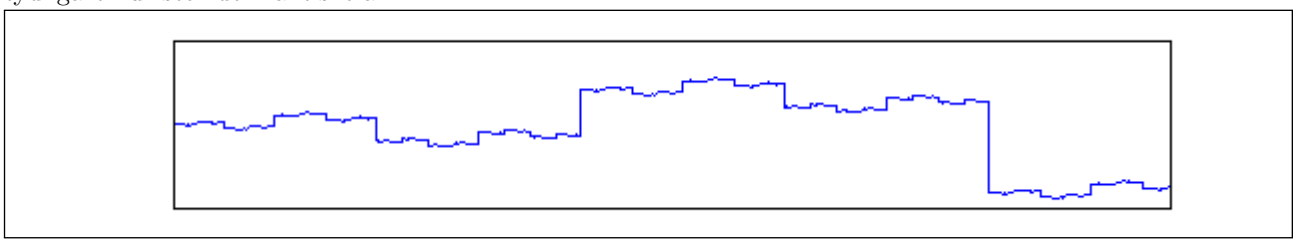


Ofta är dock inte bilden så liten att den får plats helt, vilket gör att kommandot `\centering` inte får någon som helst effekt, och bilden blir osymmetriskt placerad:

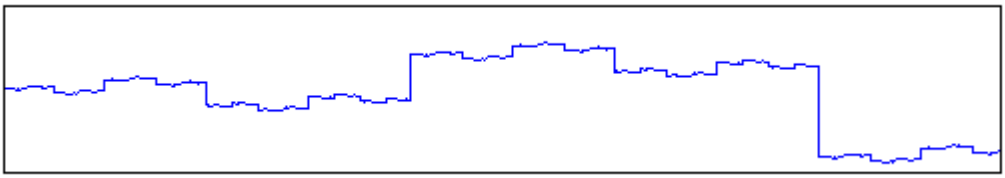


[1] *pt* står för antal punkter på skärmen, och är en vanlig måttenhet i typografisammanhang för att ange storlek på exempelvis bokstäver.
[2] *pt* står för pixlar, vilket är en måttenhet som anger för att exempelvis ange storlek på bilder och datorskärmar.

Anledningen till att det inte går att centrera just den här bilden, som är ritad i MATLAB, är egentligen inte att bilden i sig är för liten, utan att MATLAB som standard lägger på en egen marginal på bilder; vilket kan vara snyggt i MATLAB, men inte fungerar fullt så bra om man direkt importerar bilden från MATLAB. Nedan har vi lagt en ram runt bilden så att det syns lite tydligare hur stor den faktiskt är.



En lösning är att lägga till `\hspace{-3cm}` innanför *figure*-omgivningen:



Notera här att vi alltså centrerar bilden *manuellt*, dvs. man måste ändra `-1.5cm` till det avstånd man behöver dra bort för att bilden skall vara centrerad.

Nedan visas källkoden för exemplen ovan.

```
\begin{figure}[h]
  \centering
  \includegraphics{../Exempel/image_size1.png}
\end{figure}
```

```
\begin{figure}[h]
  \hspace{-1.5cm}
  \includegraphics{../Exempel/image_size2.png}
\end{figure}
```

En mindre manuell lösning, som dock kräver att paketet `adjustbox` importeras i början av dokumentet, är att lägga in bilden i omgivningen `adjustbox`. Kod som ger samma resultat som exemplet med `\hspace` visas nedan.

```
\begin{adjustbox}{center}
  \begin{figure}[h]
    \includegraphics{../Exempel/image_size2.png}
```

```
  \end{figure}
\end{adjustbox}
```

5.6 Färg

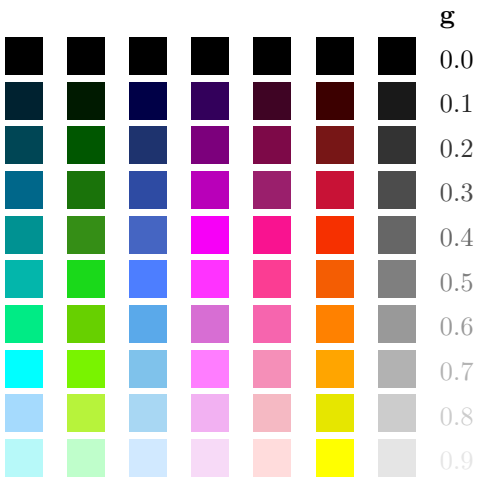
5.6.1 Val av färger

När man skriver dokument som främst är tänkta att läsas på en skärm spelar ofta valet av färger inte så stor roll. Om man misstänker att en läsare kan tänkas vilja skriva ut dokumentet så är det dock viktigt att välja färger så att de ser snygga ut även om dokumentet skrivs ut i gråskala. Detta innebär främst att man inte bör välja färger som blir för ljusa vid utskrift, eftersom att det då är svårt att urskilja vad det står i en text eller vad en figur visar.

Olika kombinationer för skrivare, pdf-läsare och drivrutiner för skrivare gör konverteringen till gråskala på olika sätt, men en tumregel man kan gå efter är följande konverteringsformel från en rgb-trippel till gråskala:

$$\mathbf{g} = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B \tag{5.1}$$

I bilden nedan visas ett exempel på hur *gråa* olika färger blir när de skrivs ut i gråskala.



Tabell 5.1: Tabellen ovan visar hur olika färger blir när de skrivs ut i svartvitt

Denna tabell kan med fördel användas då man väljer färger för sina figurer. Tabellen visar att färger som tydligt skiljer sig åt i färgtryck kan motsvara samma färg i gråskaletryck. Detta bör man givetvis ta hänsyn till, och välja färger som man ser tydlig skillnad på både i färg- och gråskaletryck.

5.6.2 Färger i LaTeX

Det finns huvudsakligen två paket som möjliggör användandet av färger i LaTeX; nämligen paketen *color* och *xcolor*. Paketet *color* möjliggör användande av enklare färger i dokument, medan paketet *xcolor* möjliggör mer avancerad användning av färger. Framför allt erbjuder paketet *xcolor* användning av fler färger än de som specificeras av paketet *color*, nämligen i princip alla färger som kan anges med någon av färgskalorna RGB[3], CMYK, gråskala respektive HTML-färgskala. För att importera de två paketen som hanterar färger skriver vi följande i vår preambel:

```
\usepackage{color}
\usepackageusenames,dvipsnames{xcolor}
```

Vi får nu tillgång till bland annat följande fördefinierade färger:

Yellow	*Lavender*	*SpringGreen*	*SkyBlue*
Goldenrod	*Magenta*	*LimeGreen*	*Cerulean*
Dandelion	*WildStrawberry*	*Green*	*NavyBlue*
Orange	*Red*	*OliveGreen*	*Blue*

Mer om vilka färger som finns fördefinierade i paketen *color* respektive *xcolor* kan du läsa om här:

Avsnittet om färger på Wikibooks `http://en.wikibooks.org/wiki/LaTeX/Colors`

Manualen till paketet xcolor `http://ctan.uib.no/macros/latex/contrib/xcolor/xcolor.pdf`

Färger kan exempelvis användas för att göra färgade rutor eller färga delar av en text, för att anpassa kommandon så som `\todo` eller för att ange vilka färger som skall användas till olika delar av källkod. Vi visar nedan två exempel på kommandon som visar hur man tillfälligt byter färg på sin text respektive gör en färgad ruta med text i. Ytterligare exempel på hur man kan använda färger i sina dokument finns i de avsnitt där motsvarande kommandon dyker upp.

```
\textcolor{WildStrawberry}{byter färg}
```

```
\fcolorbox{WildStrawberry}{Lavender!30}{färgad ruta}
```

Definition av egna färger

Om man vill använda färger som inte har något eget namn i LaTeX (med paketen `color` respektive `xcolor`) så kan man få dem genom att specificera deras rgb-kod. Om man då använder samma färg

[3] LaTeX skiljer här på två olika RGB-skalor. Den skala där varje fäg anges av tre tal mellan 0 och 255 kallas för RGB, medan den där varje färg anges av tre tal mellan 0 och 1 kallas för rgb.

många gånger så kan det vara smidigt att ge den ett namn, vilket går att göra vart som helst i sin kod med hjälp av kommandot `\definecolor`. Vi visar ett antal exempel på sådana definitioner av färger nedan.

```
\definecolor{dark_blue}{RGB}{29,29,102}
\definecolor{mid_blue}{RGB}{51,51,178}
\definecolor{light_blue}{RGB}{153,153,217}
```

Vi kan även *blanda* de färger som redan finns med hjälp av funktionalitet i paketet *xcolor*. Antag att vi exempelvis vill blanda vår nya färg *dark_blue* med vitt, så att vi får 10% blått och 90% vitt. Denna nya färg vill vi kalla *very_light_blue*. Vi kan då använda nedanstående syntax:

```
\colorlet{very_light_blue}{dark_blue!10!white}
```

Vi kan nu använda fyra av våra nya färger till att exempelvis göra listan nedan.

- *dark_blue*
- *mid_blue*
- *light_blue*
- *very_light_blue*

5.7 Elektroniska pdf-filer – klickbara referenser

När en pdf-fil är tänkt att spridas i elektroniskt format så finns det ett antal saker man kan göra för att den skall bli trevligare att läsa. Nedanstående två kommandon gör att innehållsförteckning och alla referenser blir klickbara, vilket gör det enklare att navigera i dokumentet för läsaren.

```
\usepackage{url}
\usepackage{hyperref}
```

Den enda modifiering av koden man behöver göra är i fallet med *url*-adresser, vilka vi behöver skriva som `\url{http://www.chalmers.se/math}` för att de skall bli länkar istället för text i vårt dokument. Notera att vi inte ser någon som helst skillnad i en utskriven version av dokumentet.

Om vi inte gör ytterligare inställningar så får vi paketets standardinställningar, vilket exempelvis innebär en röd ram kring allt som är klickbart, dvs.kring alla refernser, länkar och fotnoter. Det finns dock ett antal inställningar som går att göra för att anpassa hur markeringen av vad som är klickbart. Sådana inställningar görs med kommandot `\hypersetup{}`. Nedan visar vi vilka inställningar vi har använt till det här dokumentet. Notera att oavsett ilka inställningar vi väljer, så försvinner alla markeringar av länkar då dokumentet skrivs ut.

```
\usepackage{url}
\usepackage{hyperref}
\hypersetup{
  colorlinks=false,
```

```
  linkbordercolor=gray,
  citebordercolor=gray,
  urlbordercolor=gray}
```

5.8 En tom sida

Om man skriver längre rapporter så vill man ofta ha första sidan som ett separat blad, dvs. man vill ha en titelsida och sedan en tom sida, innan dokumentets egentliga innehåll börjar. Oavsett hur många gåner man skriver `\newpage` eller `\clearpage` efter varandra så får man ingen blanksida; utan LaTeX betraktar alla kommandon efter det första som överflödiga. En fungerande lösning på problemet, som förmodligen varken är den snyggaste eller kortaste, men som åtminstone inte kräver import av ytterligare paket, är följande:

```
\clearpage
\mbox{}
\clearpage
\end{SaveVerbatim}
```

Om vi importerar paketet *fancyhdr* och innan kommandot `\clearpage` skriver `\thispagestyle{empty}` så blir vi dessutom av med sidans sidnummer, och får därmed en helt tom sida.

Kapitel 6

Källkod

Anledningen till att man vill använda någon form av omgivning runt sin källkod, och inte bara klistra in den i dokumentet som den är ganska enkel; det finns en risk att många av de specialsymboler som ett språk använder försvinner om man bara klistrar in den, och dessutom är risken stor att eventuell indentering och liknande inte tas med. Det finns också en risk att man använder ett kommando eller tecken som betyder någonting i LaTeX och som därmed tolkas som någonting annat än man tänkt. Ett exempel på ett tecken som ger problem är %-tecknet; i MATLAB kan det vara en del av kod genom att det markerar en kommentar, och i många andra språk är det symbolen för modulo-räkning, men kopierar man in det till LaTeX så tolkas det som att allting efter tecknet är en kommentar i LaTeX, och därmed syns varken tecknet eller resterande delen av raden. Antag exempelvis att vi vill klistra in följande Python-kod i vårt dokument för att visa hur man kan skriva en funktion som utför moduloräkning:

```
def modulo(m,n):   # calculates m mod n
    return m % n
```

Om vi klistrar in ovanstående direkt i vår LaTeX-fil så går filen över huvud taget inte att kompilera, eftersom att # är ett specialtecken i LaTeX. Det problemet kan vi lösa genom att byta ut # mot `\#` och vi får då följande resultat:

def modulo(m,n): # calculates m mod n return m

Vi har alltså även tappat radbrytningen. Vi kan lösa även det problemet genom att lägga till \\ i slutet av varje rad, och får då i stället resultatet:

def modulo(m,n): # calculates m mod n
return m

Vi har dock ytterligare ett problem, nämligen att % tolkas som en LaTeX-kommentar. Byter vi ut alla % mot \% får vi vårt första vettiga resultat:

def modulo(m,n): # calculates m mod n
return m % n

Den kod vi använt nu är:

```
def modulo(m,n): \# calculates m mod n\\
    return m \% n
```

dvs. vi har behövt modifiera vår ursprungliga källkod en hel del för att komma hit, och då har vi ändå inte fixat indenteringen. Det kan alltså bli jobbigt redan för källkod enbart bestående av två korta rader kod, så att klistra in större mängder kod direkt i dokumentet är helt enkelt inte en bra idé.

En annan fördel med att använda en källkodsomgivning är att man gärna vill typsnitt på den del av en text som är källkod, så att man ser att det är just källkod den är. Det här avsnittet av dokumentet beskriver tre olika sätt att infoga källkod på.

Innan man använder något av kommandona nedan måste man fundera på vad för typ av källkod det är man vill infoga. Det finns huvudsakligen tre alternativ:

1. Enbart något enstaka kommando skall infogas, exempelvis för att visa vilket kommando i MATLAB som ger en 8x8-matris med ettor.
2. Flera rader kod skall infogas, men egentligen inte kod som man använder, utan fortfarande bara som ett utdrag ur en längre bit kod eller som ett fabricerat exempel.
3. En hel fil kod skall infogas, och dessutom kod som uppdateras under projektets gång. Exempel på det här är källkod som används i kandidatarbetet. Ofta har man kanske en fil med kod i klar ganska tidigt, men uppdaterar den under kursens gång.

6.1 För enstaka ord, exempelvis variabelnamn

När vi bara vill infoga ett enskilt ord av källkod i ett dokument så är det enklast att använda kommandot `\verb==`. Den text vi lägger mellan likhetstecknen kompileras då inte. Ett exempel på hur man kan använda kommandot är följande:

```
För att få en radvektor med 1000 normalfördelade variabler
använde vi kommandot \verb=randn(1,1000)= i MATLAB.
```

Resultatet blir då:
För att få en radvektor med 1000 normalfördelade variabler använde vi kommandot `randn(1,1000)` *i MATLAB.*

Tecknet = kan bytas ut mot de flesta andra tecken, vilket kan vara praktiskt om den källkod man vill infoga använder just det tecknet; kommandona `\verb=randn(1,1000)=`, `\verb+randn(1,1000)+`, `\verb-randn(1,1000)-` och `\verb$randn(1,1000)$` ger exakt samma resultat.

För att kunna använda kommandot `\verb==` behöver vi importera paketet *verbatim*. Klistra in följande kod mellan `\documentclass[...]{...}` och `\begin{document}`:

```
\usepackage{verbatim}
```

6.2 För längre bitar av källkod

När man vill infoga mer än något enstaka ord av källkod så passar omgivningen *verbatim* bättre:

```
\begin{verbatim}
def modulo(m,n): # calculates m mod n
    return m % n
\end{verbatim}
```

Även för att använda detta kommando behöver vi importera paketet *verbatim*; se föregående avsnitt. Om vi gör det så får vi alltså tillgång till både omgivningen *verbatim* och kommandor `\verb==`. Resultatet av kodexemplet ovan blir:

```
def modulo(m,n):   # calculates m mod n
    return m % n
```

Ett alternativ till att använda omgivningen *verbatim* är att använda omgivningen *listings*. För att kunna använda denna behöver vi först importera paketet *listings* genom att mellan `\documentclass[10pt,a4paper]{article}` och `\begin{document}` skriva:

```
\usepackage{listings}
```

På det ställe i texten där vi vill ha källkod skriver vi nu följande text.

```
\begin{lstlisting}
  def modulo(m,n): # calculates m mod n
    return m % n
\end{lstlisting}
```

Resultatet blir då:

```
def modulo(m,n):    # calculates m mod n
    return m % n
```

6.3 Källkod från en fil

Att använda en verbatim-omgivning kan kännas lockande; framförallt för att det är relativt få kommandon som krävs för att infoga källkoden i dokumentet. Fördelarna med att inte göra det är dock flera; genom att använda paketet listings istället så tillkommer flera finesser;

- Vi kan ha radnummer i källkoden, som gör det lättare att hitta till och referera till specifika delar av den
- Vi kan färga källkoden automatiskt
- Vi kan ha källkoden kvar i den fil vi har den, och infoga genom att ange filnamnet i stället för att klistra in texten som finns i filen. Det gör att källkoden i rapporten automatiskt ändras då innehållet i källkodsfilen ändras då man kompilerar sin fil

För att använda paketet listings så behöver man först infoga följande högst upp i sitt dokument; mellan `\documentclass[10pt,a4paper]{article}` och `\begin{document}`:

```
\usepackage{listings}
```

För att sedan använda listings-paketet så skriver vi i filen:

```
\lstinputlisting{filnamn.f90}
```


```
def modulo(m,n):    # calculates m mod n
    return m % n
```

6.3.1 Formattering av källkod

Med paketet *listings* är det enkelt att anpassa utseendet på den källkod man visar i sitt dokument. Man gör det enklast genom att definiera en egen stil för hur källkod skall visas med hjälp av kommandot `\lstdefinestyle` som placeras i dokumentets preambel. Vi visar nedan ett antal exempel på vad för inställningar man kan göra och hur de sedan används.

Anpassning till olika programmeringsspråk

Vårt första exempel visar hur man kan anpassa hur källkoden presenteras beroende på vilket språk den är skriven i. Nedan visas ett antal exempel, som förhoppningsvis gör det tydligt hur det fungerar. Syftet med att ange vilket språk koden är skriven i är att *listings* då känner igen vilka delar av koden som är så kallade nyckelord (ord som markerar start och slut på loopar, variabel definitioner och liknande i olika språk) respektive kommentarer, och formatterar koden så att man skall se tydlig skillnad mellan dessa. Paketet *listings* har stöd för bland annat följande programmeringsspråk:

- bash
- Basic
- C
- C++
- Fortran
- Gnuplot
- HTML
- Java
- Lisp
- Mathematica
- MATLAB
- Octave
- Pascal
- Perl
- PHP
- Python
- R
- Ruby
- SQL
- XML

```
\begin{lstlisting}[language=Fortran]
def modulo(m,n): # calculates m mod n
    return m % n
\end{lstlisting}
```

```
\lstinputlisting[language=Fortran]{filnamn.f90}
```

```
# calculates m mod n
def modulo(m,n):
    return m % n
```

```
\begin{lstlisting}[language=MATLAB]
% calculates m mod n
function remainder = modulo(m,n)
    remainder = mod(m,n);
\end{lstlisting}
```

```
\lstinputlisting[language=MATLAB]{filnamn.m}
```

```
% calculates m mod n
function remainder =  modulo(m,n)
    remainder = mod(m,n);
```

Radnumrering

```
\lstinputlisting[language=MATLAB, numbers=left, numberstyle=\small\color{gray}]
   {filnamn.m}
```

```
1 % calculates m mod n
2 function remainder =  modulo(m,n)
3     remainder = mod(m,n);
```

Parametern `numbers=left` anger att vi vill ha siffror på vänster sida av vår kod. Parametern `numberstyle` kan vi använda för att ställa in utseendet på radsiffrorna. I vårt fall valde vi att använda små grå siffror.

Byte av typsnitt

Givet att man har angett vilket programmeringsspråk den källkod man tänkt infoga är skriven i så finns det fyra olika typer av texter i källkoden vilka man kan ange typsnitt och formattering för: vanlig kod, kommentarer och nyckelord.

Den del av koden som inte är nyckelord eller kommentarer formatteras med hjälp av parametern `basicstyle`, strängar med parametern `stringstyle`, kommentarer med parametern `commentstyle` och nyckelord med parametern `keywordstyle`. Det finns huvudsakligen två saker man kan ändra med de tre grupperna av text, typsnitt och färg. Färgen på respektive grupp ändras med hjälp av kommandot `\color`, och typsnitt med hjälp av något av kommandona i tabellen nedan nedan:

Stil	*Kommando*	*Exempel*
-	`\rmfamily`	Lorem ipsum dolor sit amet
-	`\ttfamily`	`Lorem ipsum dolor sit amet`
-	`\sffamily`	Lorem ipsum dolor sit amet
-	`\bfseries`	**Lorem ipsum dolor sit amet**
-	`\rmfamily\itshape`	*Lorem ipsum dolor sit amet*
-	`\ttfamily\itshape`	*`Lorem ipsum dolor sit amet`*
-	`\scshape`	Lorem ipsum dolor sit amet
-	`\slshape`	*Lorem ipsum dolor sit amet*

Vi visar nedan ett exempel där vi gör följande inställningar:

- All källkod, både nyckelord och vanlig kod vill vi skall skrivas med *skrivmaskinstypsnitt*.
- Alla kommentarer skall skrivas med grå kursiv text
- Alla nyckelord skall skrivas med färgen *Magenta*
- Alla strängar skall skrivas med färgen *LimeGreen*

```
\lstinputlisting[language=MATLAB,
  basicstyle=\ttfamily,
  commentstyle=\rmfamily\itshape,
  stringstyle=\ttfamily\color{Lavender},
  keywordstyle=\ttfamily\color{Magenta}]{filnamn.m}
```

```
% calculates m mod n
function remainder =  modulo(m,n)
    str = 'malin';
    remainder = mod(m,n);
```

Definitiner av formatterings-stilar

Istället för att varje gång vi infogar källkod i vårt dokument skriva hur den skall formatteras, så kan vi definiera en egen formatterings-stil som vi sedan använder när vi vill infoga kod i vårt dokument. Antag exempelvis att vi importerar mycket MATLAB-kod med hjälp av följande formatterings-parametrar (vilket får koden att färgas ungefär som den görs i MATLAB):

```
\lstinputlisting[language=MATLAB, basicstyle=\ttfamily,
  keywordstyle=\color{blue}, commentstyle=\color{Green},
  extendedchars=true, backgroundcolor=\color{white},
```

```
  tabsize=4, showspaces=false, showstringspaces=false]{filnamn.m}
```

Vi kan då definiera stilen *MATLAB* med hjälp av omgivningen `lstlisting`, som då placeras i ditt dokumentets preambel:

```
\lstdefinestyle{MATLAB}{basicstyle=\ttfamily,
  stringstyle=\color[rgb]{.6275,.1255,.9412},
  keywordstyle=\bfseries\color{blue},
  commentstyle=\color[rgb]{.1333,.5451,.1333},
  extendedchars=true, backgroundcolor=\color{white},
  tabsize=4, showspaces=false, showstringspaces=false}
```

Vi kan nu använda vår nya stil *MATLAB* då vi lägger in kod i vårt dokument genom att skriva:

```
\lstinputlisting[language=MATLAB, style=MATLAB]
    {filnamn.m}
```

eller motsvarande

```
\begin{lstlisting}[language=MATLAB, style=MATLAB]
% calculates m mod n
function remainder = modulo(m,n)
    remainder = mod(m,n);
\end{lstlisting}
```

och får då följande resultat:

```
% calculates m mod n
function remainder =  modulo(m,n)
    str = 'malin';
    remainder = mod(m,n);
```

På motsvarande sätt kan vi definiera den formatteringsstil vi använt i det här dokumentet:

```
\lstdefinestyle{LaTeX}{
  basicstyle=\ttfamily,
  backgroundcolor=\color{light_gray},
  keywordstyle=\color{black},
  commentstyle=\color{black},
  numberstyle=\tiny\color{gray},
  xleftmargin=8pt,
  xrightmargin=8pt,
  frame=single,
```

```
  rulecolor=\color{light_gray},
  framexleftmargin=4pt,
  framexrightmargin=4pt,
  framextopmargin=0pt,
  framexbottommargin=0pt,
  extendedchars=true,
  columns=flexible,
  moredelim=[is][\color{code_gray}]{|}{|}}
```

Detta ger bland annat gråa rutor bakom källkoden, och gör all kod som ligger mellan två vertikala streck ljusgrå.

och pgfplot.tex

Kapitel 7

Tikz

I det här avsnittet beskriver vi olika sätt på vilka man kan ange matematiska uttryck i sitt dokument. Observera att vi inte går igenom specifik syntax, eftersom att det finns mycket dokumentering av detta på internet[1], utan snarare går igenom de olika omgivningar man kan välja att använda för att skriva ekvationer, samt den grundläggande *grammatik* man behöver känna till för att förstå hur matematiska formler byggs upp i LaTeX.

LaTeX skiljer mellan två olika typer av matematiska ekvationer/uttryck; de som finns i flytande text och de som är på en egen rad. Vi kommer i det här avsnittet att börja med att gå igenom det minimum man behöver kunna för att kunna förstå hur matematiska uttryck byggs upp i LaTeX, för att sedan fortsätta med att gå igenom de olika sorters omgivningar man kan välja mellan och sedan avsluta med att visa hur man kan använda sådana omgivningar som en del i satser, korrelarium, lemman osv. som en del av sin text.

I samtliga avsnitt av detta dokument kommer vi att anta att paketen *amsmath* och *amsfonts* redan är inporterade till dokumentet, vilket du gör genom att skriva nedanstående kommandon i din preambel eller genom att importera vårt matematikpaket (se avnitt 8.2).

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

7.1 Matematisk grammatik i LaTeX

```
\begin{align*}
   \lim_{n \to \infty} \frac{n!}{\displaystyle \sqrt{2\pi n}
     \left( \frac{n}{e}\right)^n} = 1 \Leftrightarrow
```

[1] Se exempelvis `http://en.wikibooks.org/wiki/LaTeX/Mathematics`.

```
    \\[2mm]
    \lim_{n \to \infty} \frac{n!}{\displaystyle \sqrt{2\pi n}
      \left( \frac{n}{e}\right)^n} = 1
\end{align*}
```

och pgfplot.tex

Kapitel 8

Kommentarer i dokumentet

När man är flera personer som skriver ett arbete tillsammans så kan det vara bra att på ett smidigt sätt kunna ge kommentarer på varandras texter i dokumentet. Ett smidigt paket för just det är paketet *todonotes*.

8.1 Kommentarer med *todonotes*

Importera paketet *todonotes* genom att skriva följande text i din preambel:

```
\usepackage[colorinlistoftodos=true]{todonotes}
```

Vi får nu tillgång till två olika slags kommentarer. Den första typen av kommentar är en kommentar i marginalen av en text. Nedan följer ett kort exempel:

```
Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis
suscipit \todo{laboriosam betyder mödosam}laboriosam, nisi ut aliquid ex ea
commodi consequatur?
```

Om vi skriver detta så får vi följande resultat:
Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur?

laboriosam betyder mödosam

Notera särskilt att vi får en linje från det ställe i texten där kommentaren är placerad, till rutan med kommentaren i.

Den andra typen av kommentarer man får tillgång till med paketet *todonotes* är kommentarer på en egen rad, vilket kan vara bra för långa kommentarer:

```
\todo[inline]
{
  Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
  doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore
  veritatis et quasi architecto beatae vitae dicta sunt explicabo.
}
```

Detta ger följande resultat:

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

8.2 Anpassning av kommentarer

När man är flera personer som skriver i samma dokument så kan det vara smidigt att på ett enkelt sätt kunna se vem som har skrivit vilken kommentar. Ett sätt att lösa det på är att ge varje skribent varsin färg. Vi gör det genom att definiera ett nytt kommando[1]. Lägg följande kod i ditt dokuments preambel.

```
\newcommand{\noteMalin}[2][]
{
  \todo[backgroundcolor=light_blue!40,
        linecolor=dark_blue,
        bordercolor=dark_blue, #1]{#2}
}

\newcommand{\noteNiklas}[2][]
{
  \todo[backgroundcolor=light_blue!80,
        linecolor=dark_blue,
        bordercolor=dark_blue, #1]{#2}
}
```

Detta definierar två nya kommandon; `\noteNiklas` samt `\noteMalin`, som fungerar precis som vårt gamla kommando `\todo` bortsett från valet av färger. För att använda det skriver vi:

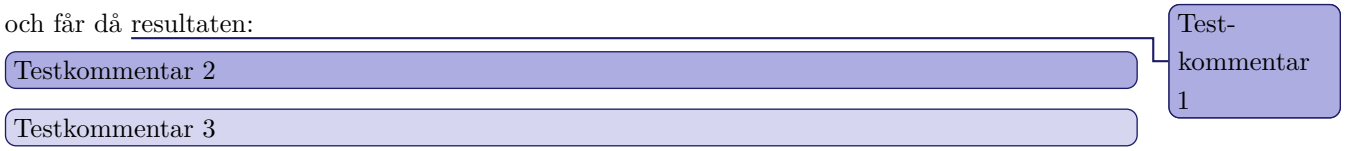
```
\noteNiklas{Test-kommentar 1}
```

[1] Se även avsnitt 8

respektive:

```
\noteNiklas[inline]{Testkommentar 2}
\noteMalin[inline]{Test-kommentar 3}
```

och får då resultaten:



8.3 En lista med alla kommentarer

Genom att skriva:

```
\listoftodos[Kommentarer]
```

så får vi en lista med alla kommentarer, vilket kan vara praktiskt för att se vilka kommentarer på texten som är kvar att åtgärda. Texten innanför klamrarna anger namnet på vår lista. Resultatet blir följande:

Kommentarer

Kapitel 9

Anpassning

9.1 Att göra egna kommandon

När man skriver samma kommando om och om igen i sin text, som exempelvis kommandot `\mathbb{R}` i de flesta matematiska texter, så kan det vara smidigt att definiera ett nytt kommando som gör samma sak, men med färre antal tecken. I det här avsnittet kommer vi att visa hur man gör just det genom att visa ett antal olika exempel. I samtliga fall skall definitionerna placeras i dokumentets preambel.

9.1.1 Nya kommandon utan parametrar

Den enklaste typen av nya kommandon är de som inte har några parametrar. Antag att vi vill ha ett kommando som ersätter en bit LaTeX-kod. Vi skriver då följande text i vår preambel:

```
\newcommand{\<kommandonamn>}{<kommandokod>}
```

där `<kommandonamn>` respektive `<kommandokod>` ersätts med vad vi vill att kommandot skall heta respektive vad det skall göra. Nedan visar vi ett antal exempel:

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

\newcommand{\dx}{\,\mathrm{d}x}
\newcommand{\ddx}{\frac{\mathrm{d}}{\mathrm{d}x}}

\newcommand{\verticalspace}{6pt}
```

Observera att koden mellan måsvingarna som definierar vad ett kommando gör kan vara i princip vilken typ av LaTeX-kod som helst; både vanlig text och kommandon för matematik som för vad som helst annat.

9.1.2 Nya kommandon med parametrar

Vi visade ovan hur man gör egna kommandon utan parametrar, men ibland vill man kunna ange en parameter till sitt kommando. Syntaxen för att göra det skiljer sig mycket lite från syntaxen ovan:

```
\newcommand[<antal parametrar>]{\<kommandonamn>} {
  <kommandokod>
}
```

I koden som hör till sin definition kan man sedan skriva `#1`, `#2`, `#3` osv. för att komma åt de parametrar som matats in av den som använt kommandot.

Nedan visar vi ett antal exempel på sådana definitioner av nya kommandon.

Vårt första exempel är det kommando vi använt för att göra de små färgade rutorna i tabell 5.1. För att använda kommandot i vårt dokument kan vi, efter att vi lagt definitionen nedan i vår preambel, skriva exempelvis `\coloredbox{29,29,102}` för att i vår text få följande blåa ruta:

```
% Ritar en färgad fyrkant med sidan 5mm givet en RGB-trippel
\newcommand{\coloredbox}[3] {
  \definecolor{box_color}{RGB}{#1,#2,#3}
  \textcolor{box_color}{\rule[-1mm]{5mm}{5mm}}
}
```

Vårt nästa exempel har bara en parameter, och skriver lägger en uppochnedvänd hatt över sin inparameter. Kommandot för att lägga en sådan hatt över en symbol i LaTeX är egentligen `\check`, heter på svenska helt enkelt *hake* och på engelska *caron*, men eftersom att Hasse Carlsson brukar glömma av det och istället kalla den för *plonk*, så definierar vi härmed ett kommando med det namnet som exempel.

```
% Gör precis samma sak som kommandot \check
\newcommand{\plonk}[1] {
  \check{#1}
}
```

Om vi efter att ha lagt definitionen ovan i vår preambel så kan vi alltså skriva `\plonk{u}` och få resultatet $\plonk{u}$.

Eller för ett roligare exempel; om vi skriver

```
\[
  \langle E, \hat{\hat{\hat{\hat{\varphi}}}} \rangle =
    \langle \hat E, \hat{\plonk \varphi}\rangle
\]
```

så blir resultatet:

$$\langle E, \hat{\hat{\hat{\varphi}}} \rangle = \langle \hat{E}, \hat{\plonk{\varphi}} \rangle$$

9.1.3 Anpassning av befintliga kommandon

Ibland vill man ändra lite på ett befintligt kommando, eftersom att man exempelvis alltid gör gör samma anpassning av det medan man använder det. Ett exempel på när detta kan vara intressant är kommandot `\marginpar{}`, som lägger en kommentar i marginalen. Nedanstående kod ger exempelvis marginalkommentaren till höger.

```
\marginpar{... sunt in culpa qui officia deserunt mollit anim id est laborum.}
```

... sunt in culpa qui officia deserunt mollit anim id est laborum.

Eftersom att det ibland blir lite väl stora mellanrum mellan orden i sådana kommentarer, och typsnittet är lite för stort jämfört med den vanliga texten, så är följande modifikation av kommandot vanlig.

```
\marginpar{\raggedright \footnotesize ... sunt in culpa qui officia deserunt
  mollit anim id est laborum.}
```

... sunt in culpa qui officia deserunt mollit anim id est laborum.

Vi skulle nu kunna definiera ett helt nytt kommando med ett helt nytt namn, men om vi faktiskt aldrig kommer att vilja använda kommandot `\marginpar{}` med dess standardutseende så kan vi lika gärna ändra beteendet av själva kommandot, dvs. definiera om kommandot `\marginpar{}`. Syntaxen för att göra det är ganska lik syntaxen för att göra ett helt nytt kommando, med den enda skillnad att vi tillfälligt behöver spara vårt orginalkommando under ett annat namn för att inte hamna i en oändlig loop ...

```
\let\oldmarginpar\marginpar

\renewcommand{\marginpar}[1]
{
  \oldmarginpar{\raggedright \footnotesize #1}
}
```

Observera alltså särskilt att vi skrivit `\renewcommand` istället för `\newcommand` som vi skrev när vi definierade ett helt nytt kommando. Observera även den första raden i exemplet ovan, och hur vi använder just `\oldmarginpar` istället för kommandot `\marginpar` i vår omdefiniering av just det kommandot.

9.1.4 Definitioner av egna matematikoperatorer

I LaTeX finns många olika operatorer man kan använda i sina formler, så som `\sin`, `\cos`, `\lim` osv. Det finns två synliga skillnader resultatet av att använda sådana kommandon och att direkt skriva `sin`, `cos`, `lim` osv. Dels så blir operatorn inte kursiverad i det första fallet, och dels så blir mellanrummen rätt kring operatorn. I tabellen nedan syns en jämförelse:

Med operator	*Utan operator*
$\cos(x)$	$cos(x)$
$\sin(x)$	$sin(x)$
$\det(A)$	$det(A)$
$\sup f(x)$	$supf(x)$
$\lim(a_n)$	$lim(a_n)$
$\liminf(a_n)$	$liminf(a_n)$

Det finns dock en del funktioner som inte finns fördefinierade i LaTeX, varav ett exempel är funktionen $\operatorname{sgn}(x)$. Med hjälp av kommandot `\DeclareMathOperator` kan vi definiera funktionen $\operatorname{sgn}(x)$ så att vi kommer åt den på samma sätt som vi kommer åt andra funtioner. Definitionen av sådana funktioner, som funktionen $\operatorname{sgn}(x)$ nedan, läggs i din huvudfils preambel eller i en separat sty-fil.

```
\DeclareMathOperator{\sgn}{sgn}
```

Vårt nya kommando `\sgn` kommer nu att fungera precis likadant som kommandona `\sin` och `\cos`, och vi kan exempelvis skriva:

```
\[
  \bar x \bar = \sgn(x) \cdot x
\]
```

De kommandon för funktioner som finns i LaTeX kan lite grovt delas i i två kategorier; de som ibland har subindex under sig, och de som aldrig har det. Exempel på operatorer som ibland harsina subindex under sig är `\lim` och `\sup`, medan funktioner så som `\cos` och `sin` aldrig har det. Vi skulle ju exempelvis kunna vilja skriva

$$\lim_{n\to\infty} \qquad \sup_{x\in[0,1]} \qquad \liminf_{x\to\infty}$$

medan vi i princip aldrig vill skriva

$$\log_2 \qquad \log_{10}$$

istället för

$$\log_2 \qquad \log_{10}$$

och liknande. När vi vill definiera en funktion där subindexen skall kunna ligga under funktionsnamnet så lägger man till en stjärna precis efter `\DeclareMathOperator`. Nedan visar vi några sådana exempel:

Lägg till fler exempel här

```
\DeclareMathOperator*{\esssup}{ess\,sup}
```

9.2 Att göra egna paket

I stället för att lägga alla sina paketimporter och egna kommandon direkt i huvuddokumentet kan man välja att istället lägga hela eller delar av sin preambels innehåll i en eller flera separata filer; så kallade *.sty*-fil. Dessa kan man sedan importera precis som om de vore egna paket. Att göra egna .sty-filer är ofta praktiskt när man har dokument som använder många olika paket eller egna definitioner. Vi börjar med att visa hur man gör egna sådana paket som inte har några parametrar, och visar sedan hur man kan lägga till parametrar för att göra dem med flexibla.

9.2.1 Egna paket utan parametrar

Antag exempelvis att vi har följande LaTeX-kod i en fil som heter `huvudfil.tex`.

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

% Paket
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amsthm}

% Egna kommandon
\newcommand{\R}{\mathbb{R}}
\newcommand{\Z}{\mathbb{Z}}
\newcommand{\N}{\mathbb{N}}

\begin{document}
```

```
\section{Inledning}
Sed ut perspiciatis, unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam eaque ipsa, quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt, explicabo.

\end{document}
```

Vi kommer att göra två nya filer; en som heter `minapaket.sty` och en som heter `minakommandon.sty`.

`minapaket.sty`

```
\ProvidesPackage{minapaket}

% Paket
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amsthm}
```

`minakommandon.sty`

```
\ProvidesPackage{minakommandon}

% Egna kommandon
\newcommand{\R}{\mathbb{R}}
\newcommand{\Z}{\mathbb{Z}}
\newcommand{\N}{\mathbb{N}}
```

Givet att vi sparar dessa filer i samma mapp som huvudfilen behöver den nu bara innhålla texten nedan.

Antag exempelvis att vi har följande LaTeX-kod i en fil som heter `huvudfil.tex`.

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

% Paket
\usepackage{minapaket}

% Egna kommandon
\usepackage{minakommandon}

\begin{document}

\section{Inledning}
Sed ut perspiciatis, unde omnis iste natus error sit voluptatem accusantium
doloremque laudantium, totam rem aperiam eaque ipsa, quae ab illo inventore
veritatis et quasi architecto beatae vitae dicta sunt, explicabo.

\end{document}
```

De .sty-filer vi just gjort kan vi nu återanvända till flera olika dokument, och slipper då lägga till samma preambel varje gång.

9.2.2 Egna paket med parametrar

Det är inte mycket svårare att göra ett paket med parametrar än ett paket utan parametrar, men det kräver lite mer jobb för att få det att fungera. Vi kommer att visa hur man gör genom ett exempel vilket är en liten del av paketet *minapaketochkommandon* som följer med som bilaga till det här dokumentet. Liksom tidigare kommer vi anta att filen nedan heter `minapaket.sty` och ligger i samma mapp som din huvudfil. Vi kommer som exempel att visa hur man kan skriva ett paket som, beroende på vilket språk man matar in som parameter ser till att det dokument man skriver blir på svenska eller engelska. När paketet är färdigt så kommer det att vara möjligt att i sitt dokuments preambel skriva:

```
\usepackage{minapaket}
```

eller

```
\usepackage[swedish]{minapaket}
```

för att få rubriker och dylikt på svenska, och

```
\usepackage[english]{minapaket}
```

för att få samma saker på engelska.

Vi kommer att börja med att förklara vad de olika raderna i vårt exempel gör, och visar sedan koden för vårt exempelpaket i sin helhet. De radnummer som nämns i samband med genomgången av de olika kommandona refererar därmed till motsvarande radnummer i koden för exempelpaketet.

Rad 3:
Kommandot `\newif\if@minapaket@swedish` definierar en variabel med namnet `@minapaket@swedish` som kan ha värdet antingen *true* or *false*. Variabeln kan då användas bland kommandona i paketet för att ange att vissa saker skall göras om den har värdet *true*, och andra om den har värdet *false*.

```
\newif\if@minapaket@swedish
```

Rad 5:
Kommandot `\DeclareOption` används för att ange att en parameter med ett visst namn skall finnas, och avgör delvis vad den gör. Dess första argument är namnet på den parameter vi vill att vårt paket skall ha, och det andra argumentet skall innehålla den kod som skall köras givet att användaren angett det här alternativet. Observera dock att vi inte kan skriva vilkt kod som helst här. Vi kommer bara att ta upp två typer av kommandon som går att skriva i kommandots andra argument, vilka vi beskriver nedan. I koden nedan anger vi alltså att vi vill ha en parameter som heter `swedish`.

```
\DeclareOption{swedish}{...}
```

Rad 6:

Kommandot nedan sätter värdet på variabeln `\@minapaket@swedish` till *true*. På motsvarande ställe i kodexemplet var vi definierar parametern `english` sätter vi värdet på samma variabel till *false*.

```
\@minapaket@swedishtrue
```

Rad 7:

Raden nedan är den första raden som gör någonting helt nytt. Raden nedan ligger inuti det andra argumentet till kommandot `\DeclareOption{swedish}{...}` och anger att `\CurrentOption`, som har värdet *swedish*, skall skickas med som parameter till paketet `babel` då det importeras längre ner i filen. Ett helt ekvivalent kommando hade varit att skriva `\PassOptionsToPackage{swedish}{babel}` och på samma sätt kan vi skicka godtyckliga parametrar till godtyckliga paket vi importerar i filen givet att användaren angett alternativet vi just nu arbetar med.

```
\PassOptionsToPackage{\CurrentOption}{babel}
```

Rad 15:

Kommandot nedan anger vilket av de alternativ som finns för användaren att skicka till vårt program som parameter som skall vara standard om inget alternativ anges. I vårt fall anger vi att alternativet `swedish` skall anges automatiskt om användaren inte skickar med någon parameter. Om vi har flera olika alternativ som kan anges samtidigt, vilket inte är vårt fall eftersom att bara ett språk kan användas per dokument, så kan vi skicka med flera olika alternativ till kommandot `ExecuteOptions` genom att separera dem med ett kommatecken.

```
\ExecuteOptions{swedish}
```

Rad 16:

Nästa kommando är det sista kommandot som är en del i själva *förarbetet* till vårt nya paket, och som har med de alternativ som användaren kan ange att göra. Kommandot markerar helt enkelt att den del av vårt paket som har med alternativ att göra nu är slut, och vi kan därmed inte längre komma åt dem. För att trots allt kunna anpassa resten av vårt paket efter vilka alternativ användaren angett får vi istället använda vår variabel som vi definierade tidigare; `@minapaket@swedish`.

```
\ProcessOptions\relax
```

Rad 18:

Kommandot nedan importerar paketet `babel`, men observera att vi inte skickar med några inparametrar. Inparametrarna kommer istället från koden på rad 7 respektive rad 11, dvs. `\PassOptionToPackage{\CurrentOption}{babel}`. Om vi vill kan vi ge paketet ytterligare inparametrar här, men även då kommer parametrarna från dessa rader att skickas med.

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

Rad 26–38:

De nedanstående raderna kod visar hur vi kan använda vår variabel `@minapaket@swedish` genom att ha en if/else-sats i vårt paket, vilket gör att vissa paketimporter och kommandon bara körs givet att användaren angett ett specifikt alternativ till paketet. I vårt fall använder vi en if/else-sats för att se till att namnen på satser, korrelarium, definitioner osv. står med på rätt språk, men det går att skriva i princip vilken LaTeX-kod som helst här givet att den även går att skriva i ett vanligt dokuments preambel.

```
\if@minapaket@swedish
  ...
\else
  ...
\fi
```

Rad 40:

Det sista kommandot i vår fil markerar helt enkelt att vårt paket tar slut, och det måste alltid ligga sist i våra paketfiler som har alternativ.

```
\endinput
```

På nästa sida visar vi vårt exempelpakets kod i sin helhet.

Om du är intresserad av att läsa mer om hur man gör egna paket med parametrar rekommenderar vi följande två hemsidor:

1. LaTeX 2_ε for class and package writers: `http://latex-project.org/guides/clsguide.pdf`
2. `http://en.wikibooks.org/wiki/LaTeX/Creating_Packages`

```
\ProvidesPackage{minapaket}

\newif\if@minapaket@swedish

\DeclareOption{swedish}{
  \@minapaket@swedishtrue
  \PassOptionsToPackage{\CurrentOption}{babel}
}

\DeclareOption{english}{
  \PassOptionsToPackage{\CurrentOption}{babel}
  \@minapaketn@swedishfalse
}

\ExecuteOptions{swedish}
\ProcessOptions\relax

\usepackage[]{babel}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amsthm}

\if@minapaket@swedish
  \newtheorem{theorem}{Sats}[section]
    \newtheorem{definition}[theorem]{Definition}
    \newtheorem{lemma}[theorem]{Lemma}
    \newtheorem{proposition}[theorem]{Proposition}
    \newtheorem{corollary}[theorem]{Korollarium}
\else
    \newtheorem{theorem}{Theorem}[section]
    \newtheorem{definition}[theorem]{Definition}
    \newtheorem{lemma}[theorem]{Lemma}
    \newtheorem{proposition}[theorem]{Proposition}
    \newtheorem{corollary}[theorem]{Corollary}
\fi

\endinput
```

9.3 Snabbare kompilering av *tikz*-bilder

Tikz är ett omfattande paket för att skapa bilder med hjälp av vektorgrafik i LaTeX[1]. Om man använder många *tikz*-bilder i sitt dokument så tar kompileringen av filen ibland lång tid. I det här avsnittet kommer vi inte att visa hur man använder paketet *tikz*, men föreslå en lösning till hur du får kompileringen av din fil att gå fortare om det är så att du redan använder *tikz*.

1. Lägg till följande kod i din preambel:

```
\usetikzlibrary{external}
\tikzexternalize[prefix=tikz/]
```

2. Om du använder programmet *todonotes*, vilket du exempelvis gör om du använder det paket som finns som bilaga till det här dokumentet, lägg även till följande kod i din preambel:

```
\makeatletter
\renewcommand{\todo}[2][]%
  {\tikzexternaldisable\@todo[#1]{#2}\tikzexternalenable}
\makeatother
```

3. Använd flaggan `-enable-write18` när du kompilerar din kod med *pdflatex*, dvs. för att kompilera dokumentet skriv följande:

```
pdflatex -enable-write18 huvudfil.tex
```

   Tänk på att om du använder en editor och komiplerar via den så måste du lägga till den här flaggan även i din editor. I MikTeX gör man det enligt följande:

   (a) Gå in inställningarna i menyn, dvs. *Edit* → *Preferences ...*

   (b) Tryck på *Typesetting*

   (c) Välj *pdfLaTeX* under *Processing tools* och tryck på *Edit ...*

   (d) Lägg till `enable-write18` före raden `$fullname`

[1] För information om hur *tikz* fungerar rekommenterar vi följande sidor:

- `http://cremeronline.com/LaTeX/minimaltikz.pdf`
- `http://ctan.uib.no/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf`
- `http://tex.stackexchange.com/questions/84753/summary-of-tikz-commands`
- `http://tex.stackexchange.com/questions/42611/list-of-available-tikz-libraries-with-a-short-introduction`
- `http://en.wikibooks.org/wiki/LaTeX/PGF/TikZ`

Kapitel 10

Presentationer

Beamer[1] är ett dokumentformat i LaTeX som kan användas för att skapa presentationer. Presentationen skapas i en `.tex`-filer som kompileras med pdflatex/latex-kompilatorn, och påminner i och med det väldigt mycket om hur man skapar en rapport i LaTeX.

Ett enkelt sätt att lära sig Beamer är att använda en färdig mall och sedan anpassa den efter sina behov. Tids nog lär man sig de kommandon som krävs för att kunna skapa sina egna presentationer från grunden.

Nedan följer en mall som innehåller och redogör för de vanligaste sakerna man kan vilja göra i Beamer.

Vi börjar med att visa hur ett skelett för presentationsdokumentet kan se ut, vilket påminner väldigt mycket om filen för en vanlig rapport skapad i LaTeX. Nedanstående skelett innehåller följande:

- Val av dokumentklass: *beamer*, istället för *document*.
- Inkludering av de paket som ska användas.
- Definition av färger, kommandon samt det som behövs för att infoga källkod.
- Definition av författare, titel och datum

Nedan följer dokumentskelettet. Notera särskilt valet av dokumentklass på den första raden.

```
\documentclass{beamer}
\usepackage{mina_paket}

\usetheme{copenhagen}
\usecolortheme{whale}
```

[1] `https://bitbucket.org/rivanvx/beamer/wiki/Home`

```
\title{Exempelpresentation}
\author{Malin och Niklas}
\date{\today}

\begin{document}

\end{document}
```

Ovanstående ger ett helt tomt dokument, till vilket vi nu skall lägga den kod som bygger upp vår faktiska presentation.

De två raderna

```
\usetheme{copenhagen}
\usecolortheme{whale}
```

anger vilket tema vi vill använda för vår presentation. Temat styr hur presentationen ser ut, både vad käller sidhuvud och färgval. För mer om teman se section 10.2.

10.1 Sidinnehåll

Innehållet i presentationen delas upp i sidor, där varje sida (*frame* i beamer) utgör en sida i presentationen. Varje sida börjar med kommandot `\begin{frame}` och avslutas med `\end{frame}`, och dessa läggs mellan `\begin{document}` och `\end{document}`. Nedan följer en lista med exempel på olika sidor som tillsammans illustrerar en del av funktionerna i beamer.

10.1.1 Titelsida

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

Kommandot `\titlepage` skapar en titelsida som sammanställer informationen du angivit i `\title`, `\author` och `date`.

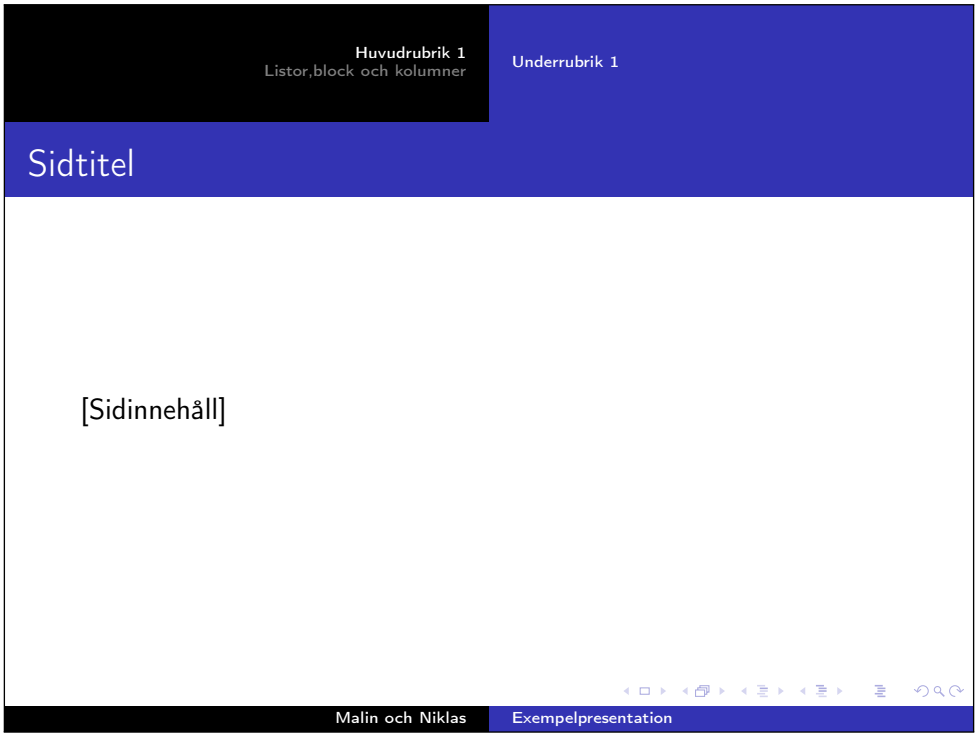
10.1.2 Sidtitlar, avsnitt och innehåll

```
\section{Huvudrubrik 1}
\subsection{Underrubrik 1}

\begin{frame}
  \frametitle{Sidtitel}
  [Sidinnehåll]
\end{frame}
```

`\section{}` och `\subsection{}` används för att skapa de listor som syns längst upp på sidan. Det som skrivs i `\section{}` hamnar i listan till vänster och det som skrivs inom `\subsection{}` hamnar i listan till höger. Dessa parametrar används också för att markera vilken sida som är den aktuella. Notera att dessa kommandon ska ligga utanför `\begin{frame}` och `\end{frame}`. Observera särskilt att kommandona `\section{}` och `subsection{}` är placerade ovanför kommandot `\begin{frame}`. Du kan därmed ha flera olika sidor (frames) under samma avsnitt och behöver alltså inte skriva `\section{}` respektive `\subsection{}` mellan varje sida i din presentation.

Kommandot `\frametitle{}` skapar titeln för den aktuella sidan. `[Sidinnehåll]` ersätts med den LaTeX-kod som motsvarar det du vill visa på respektive sida i din presentation. De flesta LaTeX-kommandon, så som vanlig text, formatteringskommandon och matematiska utryck går att lägga här, dvs. vanlig LaTeX-kod.



10.1.3 Listor

Ett vanligt element i presentationer är listor i vilka man visar punkterna i listan en i taget. Detta åstadkommer man genom att i en helt vanlig lista på en sida i sin presentation placera kommandot `\pause` mellan de punkter i listan man skapat där man vill man vill behöva ange manuellt att ytterligare en punkt skall visas.

```
\section{Presentationsspecifika kommandon}
\subsection{Listor}

\begin{frame}
  \frametitle{Listor}

  \begin{itemize}
    \item Punkt 1
      \pause
    \item Punkt 2
  \end{itemize}
\end{frame}
```

Ett alternativ till koden ovan, som ger exakt samma resultat, är följande:

```
\section{Presentationsspecifika kommandon}
\subsection{Listor}

\begin{frame}
    \frametitle{Listor}

    \begin{itemize}
      \item<1-> Punkt 1
      \item<2-> Punkt 2
  \end{itemize}
\end{frame}
```

Kommandot `\item<1->` anger att den punkt i listan den står vid ska läggas till på den första sidan och sedan ligga kvar på samtliga följande sidor. På samma sätt kan vi skriva `\item<2-3>` om vi velat ha med en punkt enbart på sida två till och med 3 och `\item<2>` om vi vill att en punkt enbart skall vara med på den andra sidan.

Resultatet av båda alternativen kod ovan blir följande två sidor. Notera att vi i båda fallen bara har använt `\begin{frame}` respektive `\end{frame}` vid ett tillfälle vardera, men att kompilatorn trots det genererar två sidor i presentationen.

Listor

- Punkt 1

Huvudrubrik 1
Listor,block och kolumner

Listor
Block
Kolumner
Byte av bild på en sida
Innehåll som tar mycket plats
Innehåll som löper över flera sidor

Listor

- Punkt 1
- Punkt 2

10.1.4 Block

Definitioner, satser, och annan viktig information som man vill lyfta fram lite extra kan man lägga i ett eget så kallat block. Det finns flera olika sorters block som man kan använda, exempelvis `block`, `exampleblock` och `alertblock`, vilket i princip bara ändrar färgerna på innehållet som visas. Utöver det så finns i Beamer även omgivningarna *theorem*, *corollary*, *definition*, *example* och *proof*, som också blir boxar givet vissa teman (du kan läsa mer om teman längre ner). Nedan har vi valt att visa tre stycken vanliga block.


```
\definecolor{dark_blue}{RGB}{29,29,102}
\definecolor{mid_blue}{RGB}{51,51,178}
\definecolor{light_blue}{RGB}{153,153,217}


\subsection{Block}
\begin{frame}
  \frametitle{Block}

  \begingroup
    \setbeamercolor{block title}{bg=dark_blue,fg=white}
    \setbeamercolor{block body}{bg=dark_blue!10,fg=black}
```

```
    \begin{block}{Titel för block 1}
      Text för block 1
    \end{block}
  \endgroup

  \begingroup
    \setbeamercolor{block title}{bg=mid_blue,fg=white}
    \setbeamercolor{block body}{bg=mid_blue!10,fg=black}

    \begin{block}{Titel för block 2}
      Text för block 2
    \end{block}
  \endgroup

  \begingroup
    \setbeamercolor{block title}{bg=light_blue,fg=white}
    \setbeamercolor{block body}{bg=light_blue!10,fg=black}

    \begin{block}{Titel för block 3}
      Text för block 3
    \end{block}
  \endgroup

\end{frame}
```

10.1.5 Kolumner

Varje sida i presentationen kan man välja att dela upp i en eller flera kolumner, exempelvis för att ha text i två kolumner eller text i den ena kolumnen och en bild i den andra.

Kommandona `\begin{columns}` respektive `\end{columns}` talar om var våra kolumner börjar respektive slutar. Kommandona `\begin{column}{5cm}` respektive `\end{column}` talar om att kolumnen skall vara 5 cm bred.

```
\subsection{Kolumner}

\begin{frame}\frametitle{Kolumner}
  \begin{columns}

    \begin{column}{5cm}
      Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam
      nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas
      nulla pariatur?
    \end{column}

    \begin{column}{5cm}
      \begin{figure}
```

```
        \includegraphics[scale=0.25]{../../Bilder/bild1.jpg}
        \caption{En liten blomma}
      \end{figure}
    \end{column}

  \end{columns}
\end{frame}
```

10.1.6 Overprint

Samma kommando som användes för att specificera när varje del i en lista skall visas kan även användas för att byta ut andra detaljer på sina sidor, exempelvis bilder. Nedanstående kod genererar två sidor där den enda skillnaden mellan dem är vilken bild som visas. Denna metod för bilder kan givetvis kombineras med motsvarande teknik för listor.

```
\subsection{Byte av bild på en sida}

  \begin{frame}
  \frametitle{Byte av bild på en sida}

  \begin{columns}

    \begin{column}{5cm}
      Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam
      nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas
      nulla pariatur?
    \end{column}

    \begin{column}{5cm}
      \begin{overprint}
        \includegraphics<1>[scale=0.25]{../../Bilder/bild1.jpg}
        \includegraphics<2>[scale=0.25]{../../Bilder/bild2.jpg}
      \end{overprint}
    \end{column}

  \end{columns}

\end{frame}
```

Resultatet blir följande två sidor:

Byte av bild på en sida

Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

Huvudrubrik 1
Listor,block och kolumner

Listor
Block
Kolumner
Byte av bild på en sida
Innehåll som tar mycket plats
Innehåll som löper över flera sidor

Byte av bild på en sida

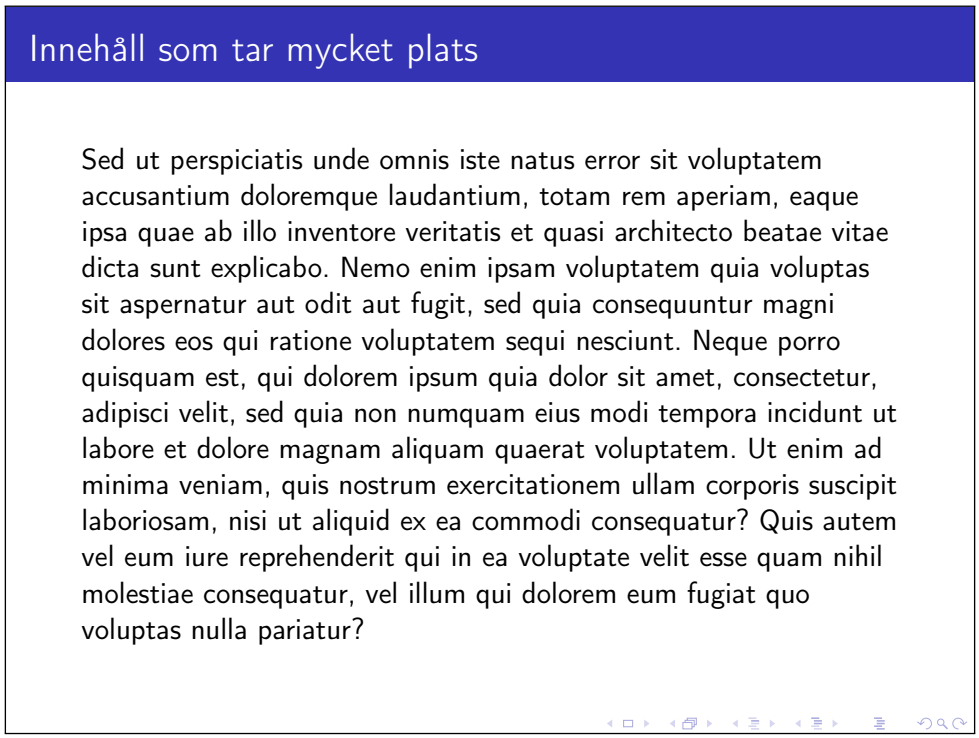
Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

10.1.7 Plain

Om man har stora bilder eller stora ekvationer i sin presentation kan man vilja plocka bort delar av inramningen kring varje sida för att kunna använda hela sidan. Detta görs med kommandot `[plain]`, som skrivs direkt efter `\begin{frame}`.

```
\subsection{Innehåll som tar mycket plats}

\begin{frame}[plain]
  \frametitle{Innehåll som tar mycket plats}
  Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
  doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore
  veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim
  ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia
  consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque
  porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur,
  adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et
  dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis
  [...]
\end{frame}
```



10.1.8 Text som löper över flera sidor

Om man har en lång text (exempelvis källkod) som inte får plats på en sida kan man göra på två sätt

- Själv dela upp texten på flera sidor vilket har fördelen att man får sidbrytningen precis där man vill
- Låta LaTeX och Beamer lägga in en sidbrytning där det behövs så att man slipper hålla koll på var sidbrytningen ska komma.

För att låta sidbrytningen ske automatiskt lägger man till `[allowframebreaks,plain]` direkt efter `\begin{frame}`. Kommandot `allowframebreaks` anger vi att tillåter texten att löpa över flera sidor.

```
\subsection{Innehåll som löper över flera sidor}

\begin{frame}[allowframebreaks,plain]
  \frametitle{Innehåll som löper över flera sidor}
  \lstinputlisting[numbers=left,language=MATLAB]{problem115.m}
\end{frame}
```

Om du har en lång text och själv vill ange var en sidbrytning skall vara finns kan du mellan `\begin{frame}[allowframebreaks,plain]` och `\end{frame}` lägga kommandot `\framebreak` in där du vill att sidbrytningen skall ske.

Om texten du har inte får plats på en frame och du inte har använt kommandot `\framebreak` kommer ett sidbyte ske automatiskt givet att du skrivit `allowframebreaks`.

Innehåll som löper över flera sidor I

```
1  function walks = problem115(m,n)
2  % This file is a solution to Euler problem number
3  % 115.
4
5  % A row measuring n units in length has red
6  % blocks with a minimum length of m units placed
7  % on it, such that any two red blocks (which are
8  % allowed to be different lengths) are separated
9  % by at least one black square.
10
11 % Let the fill-count function, F(m, n), represent
12 % the number of ways that a row can be filled.
13 % For m = 50, find the least value of n for which
14 % the fill-count function first exceeds one
15 % million.
```

Innehåll som löper över flera sidor II

```
16
17 % Sets initial values
18 br = zeros(n,2);
19 br(1:m,1) = 1;
20 br(1:m-1,2) = 0;
21 br(m,2) = 1;
22
23 % Calculates the number of such combinations
24 % which ends with a black or a red tile
25 for k = m+1:n
26     br(k,1) = 1 + sum(br(1:k-1,2));
27     br(k,2) = 1 + sum(br(1:k-m,1));
28 end
29
30 % Calculates F(m,n)
31 walks = sum(br,2);
```

10.2 Teman

Precis som vi i vanlig LaTeX-kod kan styra ganska mycket kring hur dokumentet skall se ut, så kan vi även styra utseendet på våra presentationer. I Beamer kallas huvudverktyget för att styra utseednet av presentationer för teman. Enklast gör man det genom att använda de två kommandona `\usetheme{}` and `\usecolortheme{}`. Det första temat styr strukturer på vår presentation, så som var titlar, rubriker osv. skall visas. Det andra temat anger vilka färger som skall användas. I vårt exempel ovan använde vi struktur-temat *copenhagen* och färgtemat *whale*. Det finns flera olika strukturteman och färgteman att välja mellan vilka vi listar nedan. Utöver dessa teman går nästan allt att anpassa, vilket du kan läsa mer om på de länkar vi listar nedan.

Strukturteman

- default
- boxed
- AnnArbor
- Antibes
- Bergen
- Berkeley
- Berlin
- Boadilla
- CambridgeUS
- Copenhagen
- Darmstadt
- Dresden
- Frankfurt
- Goettingen
- Hannover
- Ilmenau
- JuanLesPins
- Luebeck
- Madrid
- Malmoe
- Marburg
- Montpellier
- PaloAlto
- Pittsburgh
- Rochester
- Singapore
- Szeged
- Warsaw

Färgteman

- default
- structure
- albatross
- beaver
- beetle
- crane
- dolphin
- dove
- fly
- lily
- orchid
- rose
- seagull
- seahorse
- whale
- wolverine

Samtliga färgteman ovan kan du använda så som vi visat tidigare, förutom temat *structure* som är ett specialtema. Temat *structure* tillåter nämligen att vi väljer valfri färg, varpå alla färger i vår presentation automatiskt anpassas efter den färgen. För att exempelvis använd temat *structure* tillsammans med den i LaTeX inbyggda färgen *yellow* skriver vi följande i vår preambel.

```
\usecolortheme[named=yellow]{structure}
```

Vidare läsning

För mer information om hur man kan anpassa utseendet på sin Beamer-presentation så rekommenderar vi följande sidor:

`http://www.math.umbc.edu/~rouben/beamer/quickstart.html`

`http://www.hartwork.org/beamer-theme-matrix/`

10.3 Åhörarkopior

Beamer har inget inbyggt stöd för att generera den typ av åhörarkopior, men i likhet med många andra funktioner i LaTeX så får man den funktionaliteten om man inporterar ett specifikt paket. Det specifika paketet heter *handoutWithNotes*, och ingår inte i standardinstallationer av LaTeX. Det är trots det ganska lätt att använda paketet, och mer om hur man gör går att läsa på följande sida: `http://www.guidodiepen.nl/2009/07/creating-latex-beamer-handouts-with-notes/`.

Bilaga A

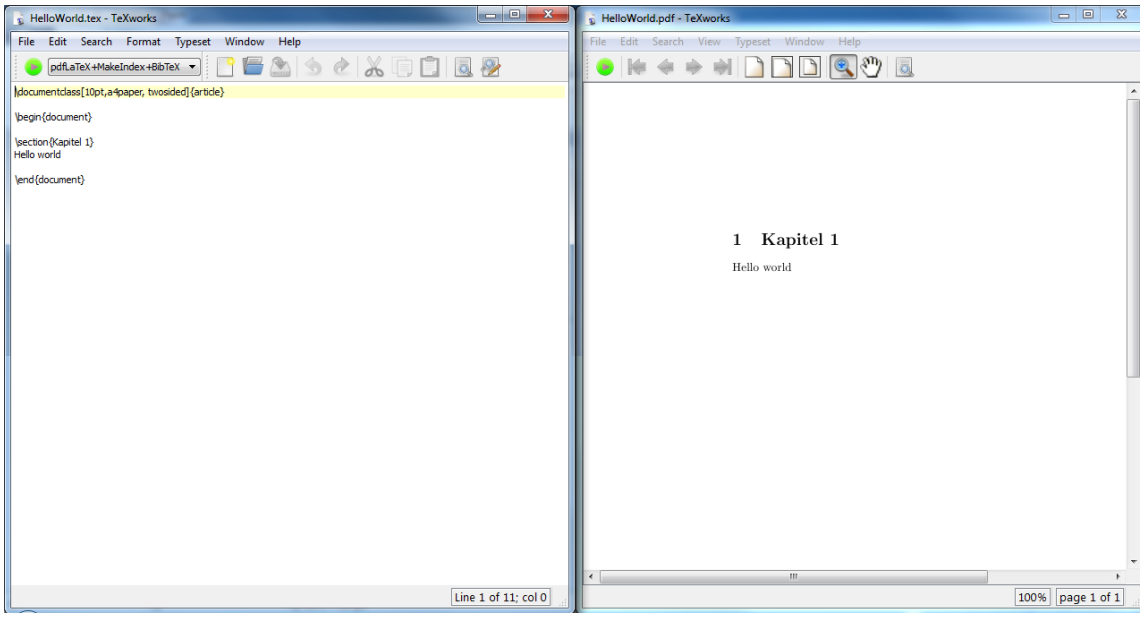
Editorer

Det finns många olika texteditorer man kan använda för att skapa källkoden till sidan dokument, och vilken man till slut vänder att använda beror på vilken man själv trivs bäst med. I följande appendix visar vi ett urval av editorer som finns tillgängliga, och beskriver ytterst kortfattat hur man gör de mest grundläggande sakerna i dessa. När ni har hittat en texteditor som ni känner er nöjda med så finns det dokumentation i överflöd att ta del av, både i texteditorns egna hjälp, men även i manualer på internet.

A.1 MikTEX/TEXworks

MikTEX är en editor som innehåller kompilatorer för LATEX-dokument.

Programmet MikTEX består av två fönster: ett i vilket du ser och redigerar din källkod, ett i vilket du ser hur dokumentet ser ut efter att det typsatts med LATEX-kompilatorn. Fördelen med MikTEX är att man lätt kan kompilera sitt dokument genom att välja kompilator i en lista och klicka på Typeset.

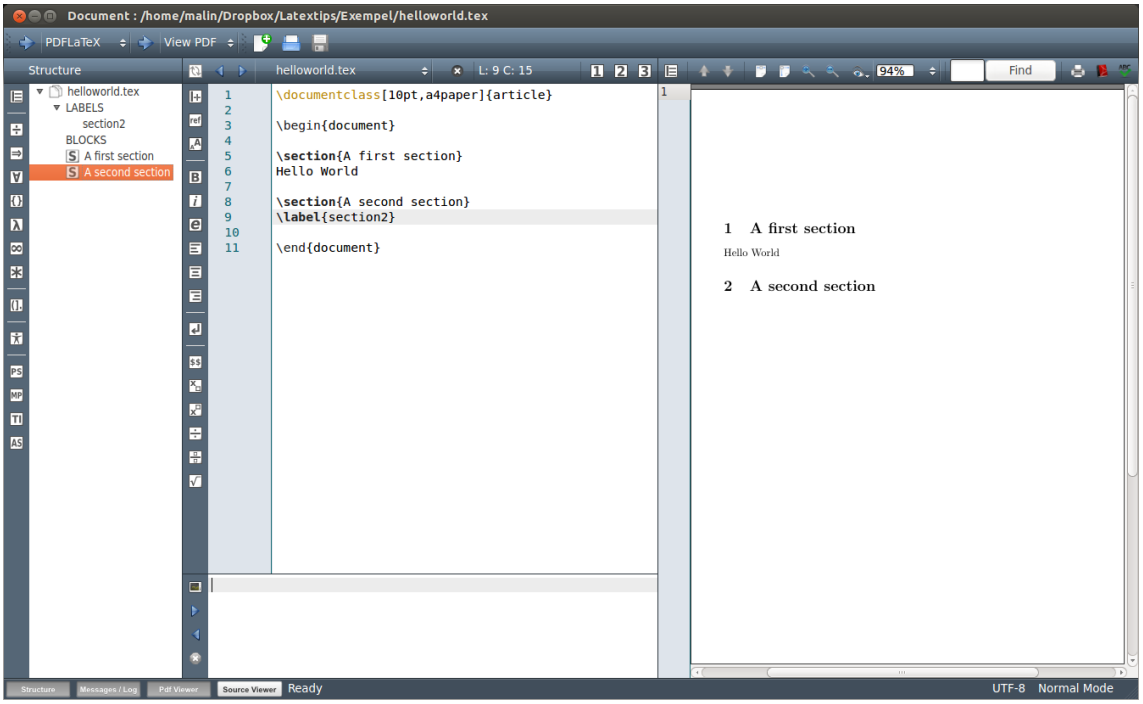


Figur A.1: Ett dokument i TEXworks: källkoden till vänster och det typsatta resultatet till höger.

MikTEX finns tillgängligt både för Linux och Windows och är lätt att komma igång med.

A.2 TEXmaker

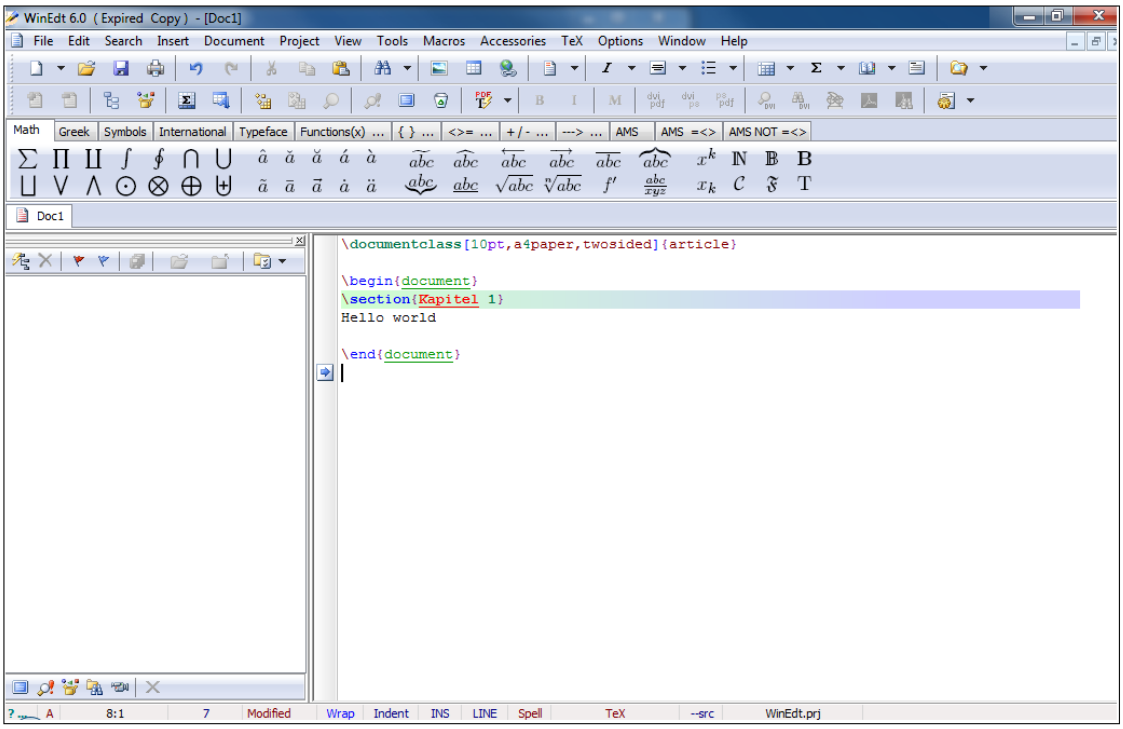
TEXmaker är en LATEX-editor som finns för både Windows och Linux. TEXmaker är relativt lik MikTEX men har mer inbyggt stöd för saker som *auto completion*, kod-färgning, minimering av delar av kod osv, och är dessutom mer anpassningsbar. En annan fördel med TEXmaker är att den har bra funktioner för saker som att hålla koll på vilka namn an gett olika omgivningar i sin kod och för att arbete med flera dokument samtidigt, vilket kan vara bra då man exempelvis har delat upp sitt dokument i flera olika del-dokument.



Figur A.2: Ett dokument i TEXmaker: till vänster ses källkoden och till höger det typsatta resultatet. Observera även det vita fältet längst till vänster; här kan du se alla omgivningar du gett ett namn med kommandot *label*, men också välja att se exempelvis någon viss typ av symboler, så som pilar, grekiska bokstäver eller liknande

A.3 WinEdit

Winedit påminner lite om Texworks i den meningen att man kan kompilera sitt dokument från editorn.



Figur A.3: Ett dokument i winedit

Winedit har ett grafiskt gränssnitt som gör det enkelt att infoga symboler man inte kan LaTeX-kommandon för. Precis ovanför textfältet finns nämligen en meny som innehåller diverse möjliga symboler, och för att infoga någon av dessa i sitt dokument räcker det att klicka på motsvarande symbol.

För att kompilera sitt dokument väljer man *PDFLatex* under *TeX/PDF* i menyraden.

WinEdit är väldigt lätt att komma igång med, och innehåller även en mycket god dokumentation.

A.4 Emacs + AucTeX

Skriv någonting här. Jag tog bort övriga sektioner här för att det känns dumt om vi bara skriver om editorer i allmänhet, när vi kanske snarare borde skriva om editorer som har särskilt stöd för LaTeX, eftersom att vi qändå antar att de som läser det här har lite datorvana...