CHALMERS | GÖTEBORGS UNIVERSITET

LaTeX-tips

En manual för matematikstudenter

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 behöver lära sig LaTeX inför arbeten i matematik eller andra naturvetenskapliga eller tekniska ämnen på universitetet.

När vi gjorde arbeten inom våra grundutbildningar i matematik upptäckte vi att LaTeX är ett **väldigt kraftfulllt** verktyg att skriva vetenskapliga rapporter med. LaTeX är inte bara ett språk som typsätter matematisk text, utan ett verktyg som, rätt använt, underlättar arbetet med referenser, svårigheter som kan uppstå när man är flera författare, och mycket mer.

LaTeX är väldigt trevligt att använda när man gör rätt, men det kan lätt kännas hopplöst när saker inte blir som man tänkt sig; figurer hamnar på fel plats, ekvationer blir oläsliga och felmeddelandena är ofta allt annat än tydliga. Lägg därtill att det inte finns någon officiell manual på samma sätt som för andra programmeringsspråk och program. Däremot finns det gott om manualer skrivna av personer från olika universitet, som valt att rikta in sig på olika områden inom LaTeX. Vi saknade däremot ett dokument som gick igenom de saker som kändes relevanta för oss. Mycket hittade vi på egen hand, och det visade sig att vissa tips gått i arv mellan årskurser inom matematikprogrammet, men aldrig skrivits ner.

Det vi vill åstadkomma med det här häftet är en LaTeX-manual som innehåller det vi tycker att man har störst nytta av att veta medan man skriver sitt arbete. Vi har delvis valt att nämna de 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 sedan med lätthet kunna nyttja de många resurser som finns online. 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 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, har vi valt att hänvisa till 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 dokument.

Som en avslutning på förordet vill vi 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, som är det verktyg som LaTeX bygger vidare på, är ett programmeringsspråk skapat av Donald Knuth för att typsätta dokument på ett enhetligt och snyggt sätt. TeX utvecklades 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 så kräver det en hel del kunskaper och tid för att kunna användas. LaTeX är ett typsättningsverktyg skapat av Leslie Lamport som lägger ett lager på TeX vilket dels uttökar funktionaliteten men också 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 resultatet blir riktigt bra.

I nästa 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 Mitt första dokument - tex-filer och kompilering

De dokument man gör skapas genom att man skriver in sin källkod i filer med ändelsen `.tex`, vilka man sedan kompilerar med hjälp av en kompilator.

Ett första dokument som skriver ut den klassiska programmeringsfrasen *Hello World* i ett eget dokument ser ut enligt följande:

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

Första raden specificerar vilken typ av dokument vi vill ha. I det här fallet vill vi använda textstorlek 10pt och pappersformat A4. Inom måsvingarna specifcierar vi att det är en artikel som vi skriver. En annan dokumentklass är *beamer*, som används för att skapa presentationer (se sid 46). Det finns även ett antal olika klasser, som alla har gemensamt att de på något sätt styr utseendet på det kompilerade resultatet, och ofta även lägger ytterligare funktionalitet, eller begränsningar, till LaTeX. I detta dokument kommer vi enbart att använda just dokumentklasserna *article* och *beamer*, men även exempelvis dokumentklassen *report* kan vara ett alternativ i studiesammanhang.

Kommandona `\begin{document}` och `\end{document}` markerar vart innehållet i vårt dokument ligger. I ovanstående fall kommer dokumentet enbart att skriva ut meningen *Hello World*. Mellan kommandot `\documentclass[...]{...}` och kommandot `\begin{document}` kommer vi under det här dokumentets gång att fylla på med diverse inställningskommandon, med det är viktigt att `\documentclass[...]{...}` alltid ligger överst i dokumentet.

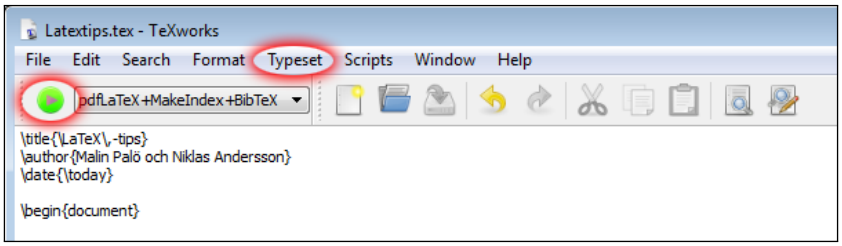
Koden placeras i ett dokument med namnet `HelloWorld.tex`, och denna kompileras sedan för att skapa själva dokumentet. Detta görs lite olika beroende på om du använder Linux, Mac eller Windows. Vi vill dock höja ett varningens finger innan ni kompilerar: det finns vissa tecken som man inte kan använda rakt av då dessa är reserverade tecken som används i kommandon i LaTeX. Vilka dessa är, och hur man gör för att skriva dem visas i tabell 1.1.

Tecken	Betydelse i LaTeX	Kommando
%	Kommentar	`\%`
$	Början/avslut på matematiska uttryck	`\$`
\	Används som första tecken i de flesta inbyggda kommandon	`\backslash`
_	Används för nedsänkning i ekvationer	`\_{}`
{ resp. }	Används för att ge indata till LaTeX-kommandon	`\{` resp. `\}`
#	Används när man definierar egna omgivningar och macron	`\#`
^	Används för upphöjning i ekvationer	`\^{}`
~	Gör ett mellanrum där radbrytning ej får finnas	`\~{}`
&	Kolumnseparator 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.

1.2 Kompilering via ett grafiskt användargränssnitt

I Windows använder man ofta ett grafiska användargränssnitt för att kompilera sin fil, snarare än att kompilera via en kombination av kommandon i terminalen, som man ofta gör i Linux. Ett exempel på en gratis sådan programvara, som finns installerat på Chalmers Windows-datorer, är MikTex[2]. MikTex är 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, och sedan använda exakt samma kommandon via terminalen som i Linux. Versioner av MikTeX finns även till Mac och Linux men då under namnet TeXWorks.

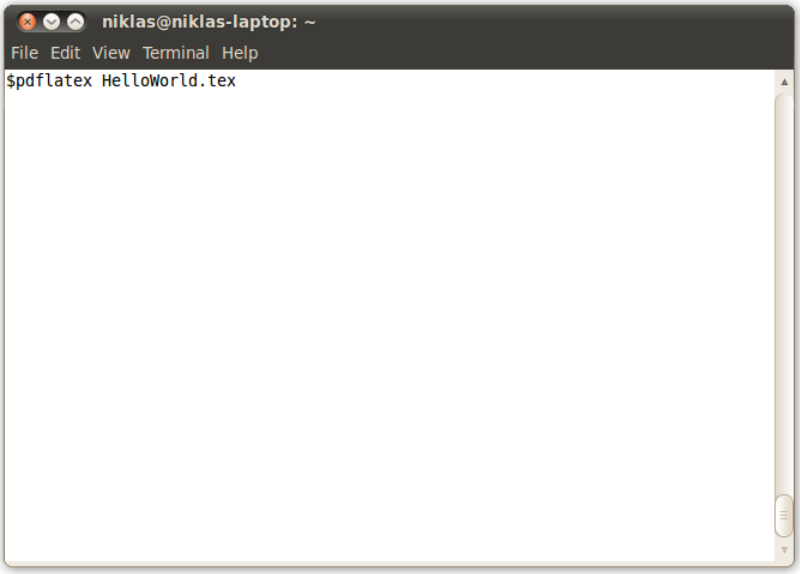
1.3 Kompilering i Linux

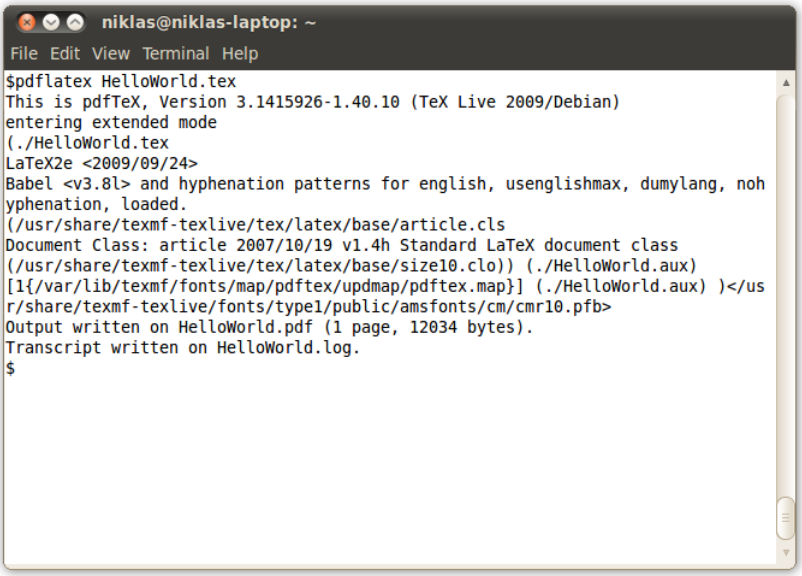
Kompilering av tex-filer i Linux sker genom att man anropar den kompilator man använder från ett terminalfönster. För att kompilera `HelloWorld.tex` i Linux, gör följande:

- 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` eller `latex 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.





Att kompileringen genomfördes med ett lyckat resultat ser vi genom utskriften:
`Output written on HelloWorld.pdf (1 page, 12034 bytes)`.

Kommandot `pdflatex HelloWorld.tex` skapar filen `HelloWorld.pdf`, medan `latex HelloWorld.tex` skapar filen `HelloWorld.dvi`. Filändelsen `dvi` hör till ett filformat som påminner om pdf. Den dvi-fil som skapas kan man konvertera till en pdf-fil genom kommandot `dvipdf HelloWorld.dvi`, vilket då skapar filen `HelloWorld.pdf`.

1.4 Hur vet jag vilken kompilator jag ska använda?

På ett tidigt stadium spelar det i princip ingen roll vilken kompilator man väljer att använda. Man bör dock tänka på att om man jobbar med figurer så kräver latexkompilatorn att bildformatet är eps. Kompilatorn pdflatex kan däremot använda bilder i bland annat png-format. Dessutom kan man inkludera flera eller enstaka sidor från andra pdf-dokument på ett smidigt sätt om man använder pdflatex.

1.5 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 `\usepackage{***}` högst upp i ditt dokument så anges att ett visst paket skall användas. Funktionalitet som läggs till kan exempelvis vara möjligheten att ha färg på texten i dokumentet, möjlighet att göra länkar klickbara eller något särskilt typsnitt. Vissa av paketen kommer att följa med din grundinstallation av LaTeX, medan vissa måste installeras separat för att kunna användas. 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 utf8, 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 så behövs dock importer av ett antal ytterligare paket. Hur detta går till beskrivs i samband med varje avsnitt.

I stället för att lägga alla sina paketimporter direkt i huvuddokumentet kan man välja att istället lägga dem i en separat fil; en så kallad *.sty*-fil. Dessa kan man sedan importera precis som om den vore ett eget paket.

För att underlätta för den som skriver sitt första LaTeX-dokument, och för att ge ett exempel på hur en *.sty*-fil kan se ut för den som redan kan en del LaTeX, har vi gjort fyra stycken sådana *.sty*-filer. Dessa innehåller importer av de paket som vi nämner i det här dokumentet. Det innebär att om ni väljer att importera våra paket, så räcker det att ni använder använder nedanstående dokumentskal, och kan bortse från alla andra fall av `\usepackage{...}`. Varje paket motsvaras av en fil med ändelsen `.sty` som innehåller de saker som man normalt lägger mellan kommandona `\documentclass[...]` och `\begin{document}` i sitt dokument. Om du vill använda paketen så behöver de ligga i samma mapp som den LaTeX-fil som du kompilerar, dvs. lägg en kopia på de filerna till samma mapp som det dokument du kompilerar ligger. Om du senare vill använda ett annat paket än de vi nämnt i det här dokumentet; kan du antingen välja att lägga till dem direkt i ditt dokument eller i någon av *.sty*-filerna.

```
\documentclass[10pt,a4paper,oneside]{article}
\usepackage{svenska}
\usepackage{matematik}
\usepackage{formattering}
\begin{document}
Hello World
\end{document}
```

Notera importen av paketen *svenska*, *formattering* och *matematik*

svenska.sty innehåller de paket som måste importeras för att kunna skriva LaTeX-dokument på svenska, samt de inställningar vi kommer att visa för att kunna använda omgivningar för satser, korrelarium osv. på svenska.

engelska.sty innehåller de paket som måste importeras för att kunna skriva LaTeX-dokument på engelska, samt de inställningar vi kommer att visa för att kunna använda omgivningar för satser, korrelarium osv. på engelska.

matematik.sty innehåller de paket vi brukar använda för att göra matematikrelaterade saker, exempelvis så importerar det här paketet matematiska symboler, teckensnitt och omgivningar för lemman, satser, bevis osv.

formattering.sty innehåller de paket vi använder för att formattera det här dokumentet.

`.sty`-filerna går att öppna i en editor precis som om den vore en vanlig textfil och går då att ändra i om man vill lägga till ytterligare paket, ändra i inställningarna för formatteringen eller ta bort något paket. Det enda som krävs för att en fil skall gå att använda som ett paket är att den har ändelsen `.sty` och att den översta raden innehåller kommandot `\ProvidesPackage{filnamn}`. Filen `matematik.sty` består exempelvis följande text:

```
\ProvidesPackage{matematik}

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

Källkoden till de andra tre paketen finns i appendix B.

Vad vi alltså i praktiken gjort är bara att flytta över importen av paket till en separat fil. Den fil vi faktiskt kompilerar när vi typsätter det här dokumentet ses nedan. Observera även paketet vi döpt till *latextips*; det innehåller de paket och kommandon som är specifika för just vårt dokument.

```
\documentclass[10pt,a4paper, twosided]{report}

% Import av paket
\usepackage{svenska}
\usepackage{matematik}
\usepackage{formattering}
\usepackage{latextips}


\begin{document}

\input{titelsida.tex}
\input{entomsida.tex}
\input{forord.tex}
```

```
\clearpage
\tableofcontents

\input{grundlaggande.tex}
\input{rapporter.tex}
\input{kompileringsfel.tex}
\input{matematik.tex}
\input{referenser.tex}
\input{kod.tex}
\input{layout.tex}
\input{beamer.tex}
\input{kommentarer.tex}


\clearpage
\appendix

\input{editorer.tex}


\input{paket}




\input{mertips}

\end{document}
```

Kapitel 2

Rapporter

2.1 Dokumentstruktur

Det finns en princip som, om den följs, underlättar arbetet med dokument som involverar flera skribenter: **skriv inte all text i samma dokument**. Det finns nämligen många fördelar med att dela upp ditt dokument i flera mindra dokument:

- Eftersom att all text skrivs som källkod, och det hela tiden är källkoden man arbetar med, och inte 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.
- Om man under arbets gång vill byta plats på sektioner 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 stycke 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 orning, vilket är lättare än att klippa och klistra ihop text manuellt. Flera olika författare kan dessutom skriva samtidigt på varsin del.

Bortsett från att man totalt får fler filer så finns det få nackdelar med att dela upp sitt dokument i flera olika filer. Givet att vi lägger en del av koden i filen *filnamn.tex* så kan vi byta ut motsvarande text mot kommandot `\input{filnamn.tex}` i vårt huvuddokument. `\input{filnamn.tex}` läser innehållet från filen `filnamn.tex` och lägger det på motsvarande ställe i den fil på den plats man gett kommandot.

Ett enkelt sätt att hålla koll på all text och okumentets struktur är att skapa ett huvuddokument som ser ut enligt följande:

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

\title{LaTeX-tips för kandidatskrivaren}
\author{Malin och Niklas}
\date{\today}

\begin{document}
\maketitle

\newpage
\section{Introduction}
\input{intro.tex}

\newpage
\tableofcontents

\newpage
\section{Main part}
\input{mainpart.tex}

\newpage
\section{Summary}
\input{summary.tex}

\end{document}
```

Vi ser att några saker tillkommit jämfört med vårt första *'Hello World'*-dokument:

- Kommandona `\title{}`, `\author{}` och `\date{}` används för att specificera rubrik, författare och datum till framsidan. Kommandot `\maketitle` infogar en framsida, och använder där informationen specificerad ovan. Notera vad som ligger före, och vad som ligger efter `\begin{document}`!
- Kommandot `\tableofcontents` skapar en innehållsförteckning som innehåller alla rubriker och underrubriker. Rubrikerna är de som specificeras med kommandot `\section{kapitelnamn}`, och underrubrikerna ges av `\subsection{namn på underrubrik}`. Observera att innehållsförteckningens rubrik automatiskt blir på engelska. Om innehållsförteckningens rubrik skall bli på svenska, och likaså källförteckningnes rubrik och dylikt, så måste det anges via en import av ett paket som anger att dokumentet skrivs på svenska. Du kan läsa mer om hur man gör detta under avsnitt 1.5.
- Raden `\input{filnamn.tex}` tar innehållet i filen `filnamn.tex` och infogar det i dokumentet på just den plats `\input`-kommandot ligger. I ovanstående mall har vi antagit att dokumenten `intro.tex`,`mainpart.tex`, `summary.tex`, `avslutning.tex` innehåller den text som ska ligga under respektive kapitel. Observera alltså att de filerna kan vara helt tomma. Vi har helt enkelt bara flyttat över exakt den text som tidigare låg i huvudfilen till en annan fil. Det är viktigt att alla våra filer ligger i samma mapp.

När vi vill kompilera vårt arbete så behöver vi bara kompilera vårt huvuddokument, dvs. det dokument som innehåller kommandona `\begin{document}` och `\end{dokument}`, så infogas all

annan text automatiskt.

2.2 Mappstruktur 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 organisera den mapp i vilken alla filer som hör till arbetet ligger på ett vettigt sätt. Eftersom att man kan ange relativa sökvägar i LaTeX så 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

Vanliga orsaker till kompileringsfel

Det finns ett flertal vanliga fel som orsakar kompileringsfel, varav de flesta nämns på `http://en.wikibooks.org/wiki/LaTeX/Errors_and_Warnings`. Exempel på sådana fel som är relativt lätta att lösa är felstavade kommandon, paket som importerats med ej finns på datorn, start och slut på omgivningar som inte matchas osv. Ett väldigt vanligt fel som inte nämns lika ofta, är problem med dokumentformat. Det här felet är särskilt vanligt förekommande om man använder ett annat operativsystem än Linux.

3.1 UTF8

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[utf8]{inputenc}
```

Det andra av de här två paketen, anger att formatet på alla filer med LaTeX-kod som dokumentet använder använder teckenkodningen UTF8. Om de inte har det, vilket de ofta inte har som standard i exempelvis Windows, så går dokumentet som helhet oftast inte att kompilera, eller alternativt så 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 UTF8. 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*.

Kapitel 4

Matematik

I detta avsnitt går vi igenom olika sätt på vilka man kan skriva matematiska uttryck i sitt dokument. Observera att vi här inte går igenom specifik syntax, eftersom att det finns mycket dokumentering av detta på internet. Snarare går vi här igenom de olika omgivningar man kan välja att använda för att skriva ekvationer, samt den absoluta grundläggande *grammatik* man behöver 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 av grammatik man behöver kunna för att kunna förstå hur matematiska uttryck byggs upp i LaTeX. Vi fortsätter sedan 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

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

precis före `\begin{document}` eller genom att importera vårt matematikpaket (se avnitt 1.5).

4.1 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, kommandon som anger vart symboler skall placeras och kommandon som formatterar symboler.

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 att använda dessa måste man i stället använda kommandon, 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 nämnt 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 kommandon de ges av i LaTeX.

Nedanstående tabell 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{v}`	$\mathbf{u}$, $\mathbf{v}$
`\mathbb{}`	Det typsnitt som ofta används för vanliga mängder	`\mathbb{R}, \mathbb{Z}`	$\mathbb{R}$, $\mathbb{Z}$
`\cal{}`	Kalligrafisk stil, används exempelvis ofta i algebra	`\cal{F}, \cal{G}`	$\mathcal{F}$, $\mathcal{G}$
`\displaystyle`	Kan skrivas precis före kommandot `\frac{}{}` för att siffrorna skall täljaren och nämnaren skall skrivas i full storlek	`\frac{x^2+1}{x-1}` vs. `\displaystyle \frac{x^2+1}{x-1}`	$\frac{x^2+1}{x-1}$ vs. $\displaystyle\frac{x^2+1}{x-1}$

4.2 Ekvationer i löpande text

Om man vill ha matematiska tecken mitt i en mening så gör man det enklast med enkla dollartecken. 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*.

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

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

```
\( \pi \approx 3.1415 \)
```

4.3 Ekvationer på en egen rad

Om man skall ha med större 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 använda dubbla dollartecken. 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 alla fall nedan så hamnar uttrycket på en egen rad oavsett om man har det på en egen rad i .tex-filen eller ej.

Ett alternativ till dubbla dollartecken som ger exakt samma resultat är att skriva

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

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

$$E[\chi] = \sum_m E[\chi_m] = \sum_m \frac{1}{2} = \frac{n}{2} \tag{4.1}$$

dvs. ett litet nummer inom parantes vid den högra marginalen. Ekvationen har nu blivit tilldelad ett eget nummer. Man skulle nu kunna referera till ekvationen genom att använda den siffran direkt, men eftersom att man ibland lägger till en ny ekvation innan den aktuella ekvationen så kan det hända att ekvationen byter siffra, 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 modifiera vår ekvation i .tex-filen till följande:

```
\begin{equation}
 E[\chi] = \sum_mE[\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}` och kan då exempelvis skriva `ekvation \ref{ekv1}` för att referera till ekvationen. Resultatet blir *ekvation (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 *align*:

```
\begin{align}
\label{ekv2}
\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{align}
```

[1] Av utrymmesskäl är detta inte samma ekvation som ovan, men förhoppningsvis illustrerar detta exempel ändå hur omgivningen *displaymath* används.

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

$$\mathbb{E}[\chi_v] = p + (1-p)(1 - p^{d_v} - (1-p)^{d_v}) \geq \tag{4.2}$$
$$p + (1-p)(1 - p^{n^\epsilon} - (1-p)^{n^\epsilon}) = \tag{4.3}$$
$$1 - (1-p)(p^{n^\epsilon} + (1-p)^{n^\epsilon}) \tag{4.4}$$

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

$$\mu(A_i) = \tag{4.5}$$
$$\mu((A_i \cup B) \cup (A_i \backslash B)) = \tag{4.6}$$
$$\mu(A_i \cup B) + \mu(A_i \backslash B) \geq \tag{4.7}$$
$$\mu(A_i \cap B) = \tag{4.8}$$
$$\infty \tag{4.9}$$

I LaTeX-filen motsvarar det följande kod:

```
\begin{align}
\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{align}
```

Om vi inte gör någonting så blir ekvationerna 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{align}
&\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{align}
```

$$\mu(A_i) = \tag{4.10}$$
$$\mu((A_i \cup B) \cup (A_i \backslash B)) = \tag{4.11}$$
$$\mu(A_i \cup B) + \mu(A_i \backslash B) \geq \tag{4.12}$$
$$\mu(A_i \cap B) = \tag{4.13}$$
$$\infty \tag{4.14}$$

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

```
\begin{align}
\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{align}
```

$$\mu(A_i) = \tag{4.15}$$
$$\mu((A_i \cup B) \cup (A_i \backslash B)) = \tag{4.16}$$
$$\mu(A_i \cup B) + \mu(A_i \backslash B) \geq \tag{4.17}$$
$$\mu(A_i \cap B) = \tag{4.18}$$
$$\infty \tag{4.19}$$

Om man vill använda *align*-omgivningen men inte få radnummer så kan man skriva *align** överallt istället för *align*. Detta fungerar även för *equation*-omgivningen, som då fungerar likadant som *displaymath*-omgivningen. 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 en specifik rad i en *align*-omgivning så placerar man istället `\label{ekvationsnamn}` precis före `\\`. Ett exempel på både rader utan och med nummer, och med referenser till specifika rader i en *align*-omgivning ses nedan:

```
\begin{align}
\mu(A_i) = \nonumber \\
\mu ((A_i \cup B) \cup (A_i \backslash B)) = \label{rad2} \\
\mu (A_i \cup B) + \mu  (A_i \backslash B) \geq \nonumber  \\
\mu (A_i \cap B) = \\
\infty \label{rad5}
\end{align}

Nu kan vi referera till ekvationerna på den andra respektive femte raden ovan,
ekvation~\ref{rad2} samt ekvation~\ref{rad5}.
```

$$\mu(A_i) =$$
$$\mu((A_i \cup B) \cup (A_i \backslash B)) = \tag{4.20}$$
$$\mu(A_i \cup B) + \mu(A_i \backslash B) \geq$$
$$\mu(A_i \cap B) = \tag{4.21}$$
$$\infty \tag{4.22}$$

Nu kan vi referera till ekvationerna på den andra respektive femte raden ovan, ekvation 4.20 samt ekvation 4.22.

4.4 Tecken och symboler

En bra start för att få reda på hur man skriver olika tecken och symboler inom LaTeX är `http://en.wikibooks.org/wiki/LaTeX/Mathematics`. Anledningen till att vi inte går igenom hur man skriver de faktiska kommandon som bygger upp de matematiska fomlerna är att det här en del av LaTeX som är mycket väl dokumenterad på internet.

4.5 Definitioner, satser, korrelarium, lemman och propositioner

Matematiska texter innehåller ofta definitioner, satser, bevis och liknande. Dessa finns det naturligtvis stöd för i LaTeX, och detta avsnitt 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 inom matematisk texter är *theorem*, *definition*, *lemma*, *proposition*, och *corollary*. Dessa deklareras 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 sitt dokument. Vi visar med några exempel. Nedanför varje grå ruta står resultatet av den källkod som finns i rutan.

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

Definition 4.5.1. *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 4.5.2. *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 4.5.3. *$\sqrt{2}$ är irrationellt. Detta följer direkt av satsen ovan med $n = p = 2$*

4.6 Bevis

Ofta vill man inte bara presentera påståenden, utan vill till dessa även ange bevis. Givet att vi, liksom ovan, redan skrivit `\begin{document}` skrivit:

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

så kan vi nu 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 $p$ delar HL, dvs $p | a^n$.
Detta innebär att $p|a$, dvs $a= pm$ för något heltal $m$.
Detta ger att  $p b^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.
\end{proof}
```

Sats 4.6.1. *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}
\label{minsats}
Om $p$ är ett primtal så är $\sqrt[n]{p}=p^{\frac{1}{n}}$ irrationellt för alla
heltal $n \geq 2$.
\vspace{-2mm}
\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 $p$ delar HL, dvs $p | a^n$.
Detta innebär att $p|a$, dvs $a= pm$ för något heltal $m$.
Detta ger att  $p b^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.
\end{proof}
```

Resultatet blir:

Sats 4.6.2. *Om p är ett primtal så är $\sqrt[n]{p} = p^{\frac{1}{n}}$ irrationellt för alla heltal $n \geq 2$.*

Bevis av sats 4.6.2. 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 5

Referenser

De referenser man gör i texter är oftast av något 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 hänvisar 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.

5.1 Referenser till litteratur

För att referera till externa källor skapar man oftast först en fil med ändelsen *.bib* där man placerar information om de externa källor man refererat till i sin text. Filen kan t.ex. 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: [1].

Notera att namnen *referens1* respektive *roginskaya* kan vara vilka namn du vill. Ett bra tips är att döpa referenserna så att namnen säger något om vilken referens det är - det blir lättare att hålla koll på dem 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 bland annat 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. Om du använder latex-kompilatorn är det bara byta ut pdflatex mot latex överallt ovan.

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 använda fler kommandon än `pdflatex` om man inte gjort några nya referenser.

Observera att det här inte kommer att ske automatiskt i de flesta **Windows-program** som sköter kompileringen om vi inte anger detta specifikt. 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
```

Om man är flera författare som skriver varsin del av ett arbete har varje författare antagligen egna referenser för sina delar. 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 bib-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).

```
% En del av en bok som saknar en egen titel, exempelvis ett
% kapitel av en bok
@INBOOK{fractalgeometry,
   title = "The Geometry of Fractal Sets",
   publisher = "Cambridge University Press",
   edition = "First",
   year = 1985,
   pages = "7--9",
   key = "Cambridge"
   }

% En publicerad bok
@BOOK{mattila,
    author = "Pertti Mattila",
    title = "Geometry of Sets and Measures in Euclidean Spaces",
    publisher = "Cambridge University Press",
    series = "Cambridge studies in advanced mathematics",
    number = "44",
    year = "1995",
}

% En ej publicerad bok eller artikel
@UNPUBLISHED{nmlatex,
   author = "Niklas Andersson och Malin Palö",
   title = "{LaTeX-tips}",
   note = "LaTeX för matematikstudenter",
   month = "september",
   year = "2012"
   }

% För artiklar från tidskrifter
@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",
}

}
```

Om vi lagt referenserna ovan i en fil och kompilerat vårt dokument hade resultatet blivit som följer.

Litteraturförteckning

[1] *The geometry of fractal sets*, first ed., pp. 7–9, Cambridge University Press, 1985.

[2] Pertti Mattila, *Geometry of sets and measures in euclidean spaces*, Cambridge studies in advanced mathematics, no. 44, Cambridge University Press, 1995.

[3] Niklas Andersson och Malin Palö, *LaTeX-tips*, LaTeX för matematikstudenter, september 2012.

[4] 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.

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.

5.2 Referenser inom dokumentet

Ofta vill man kunna hänvisa till satser, bevis, avsnitt, figurer, tabeller och liknande i sitt dokument, exempelvis genom att skriva *i figur 3 på sidan 8 såg vi att...*. Detta blir naturligtvis väldigt omständligt 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 ges ett namn, 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.

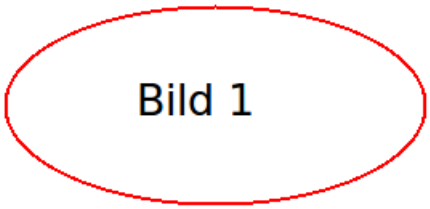
De objekt som är markerade med en `label` kan sedan refereras till, och beroende på vilken information om objektet man vill ha används olika kommandon:

- 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...*.
- Objektets titel refereras till med kommandot `\nameref{namn}`.
  Detta är lämpligt om man vill skriva *... i kapitlet* **Infoga referenser**....

Nedan följer några exempel på hur man kan använda kommandona *label-ref* beroende på vad man refererar till.

5.3 Figurer

```
\begin{figure}[h]
\center
\includegraphics[scale=0.8]{bild1.png}
\caption{Bildtext}
\label{figure_visa_label+ref}
\end{figure}

Ovan ser vi figur \ref{figure_visa_label+ref}.
```



Figur 5.1: Bildtext

Ovan ser vi figur 5.1.

Viktigt att notera är att `\label`-kommandot måste ligga efter `\caption`-kommandot. Om så inte är fallet kommer referensen istället att peka på det avsnitt som figuren ligger i.

5.4 Tabeller

```
\begin{table}[h]
\center
\begin{tabular}{|r|l|}
  \hline
  7C0 & hexadecimal \\
  3700 & octal \\ \cline{2-2}
  11111000000 & binary \\
  \hline \hline
  1984 & decimal \\
  \hline
\end{tabular}
\caption{}
\label{tabell-exempel}
\end{table}

Ovan ser vi tabell \ref{tabell-exempel}.
```

Ovan ser vi tabell 5.1.

Även här måste `\label{}`-komanndot ligga efter `\caption`-kommandot för att referensen ska peka på tabellen och inte på kapitlet. Observera här att även om vi inte lägger någon text i en figurtext så placeras ett kolon efter *tabell 5.1*, dvs. vi får en bildtext med ett överflödigt kolon. Det här löses

7C0	hexadecimal
3700	octal
11111000000	binary
1984	decimal

Tabell 5.1:

automatiskt om vi importerar paketet *caption*, dvs. om vi placerar nedanstående kod precis innan `\begin{document}`:

```
\usepackage{caption}
```

5.5 Definitioner, satser och bevis

I matematisk text refererar man ofta till tidigare definitioner och bevis, vilket görs enligt följande:

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

I definition \ref{def:primtal} definierade vi primtal.
```

Definition 5.5.1. *Ett* **primtal** *är ett heltal större än 1 som enbart har* ± 1 *och* $\pm p$ *som delare.*

I definition 5.5.1 definierade vi primtal.

5.6 Avsnitt

Följande förutsätter att vi precis efter kommandot `\subsection{Referenser inom dokumentet}` har lagt in kommandot `\label{subsec:ref_inom_dokument}`.

```
I kapitlet {\bf \nameref{subsec:ref_inom_dokument}}
 (\ref{subsec:ref_inom_dokument}, som börjar på sidan
 \pageref{subsec:ref_inom_dokument}  har vi pratat om
 referenser inom det egna dokumentet.
```

I avsnittet **Referenser inom dokumentet** (5.2), som börjar på sidan 28 har vi pratat om referenser inom det egna dokumentet.

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, är ganska enkel; det finns en risk att många av de special-symboler 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. Om man använder det tecknet som en del av en bit källkod i 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 kod 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 ofta vill byta typsnitt på den del av en text som är källkod, så att man ser att det är just källkod den är. Det här avsnittet 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 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 kommandot `\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:

```
\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
```

Observera alltså att omgivningen *listings* automatiskt känner igen vissa nyckelord i olika programmeringsspråk och låter dem ha en annan färg.

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 den 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[language=Fortran]{filnamn.f90}
```

Om man vill använda ett annat språk än Fortran så byter man ut just Fortran mot exempelvis MATLAB istället. Vi kommer nu att få se koden med de standardinställningar som paketet har för hur Fortrankod skall visas; med det enda undantag att vi har angett att vi vill ha radnummer på vänster sida. Det går dock ganska enkelt att modifiera standardinställningarna en bit till. Börja med att definiera ett antal färger innan `\begin{document}`:

```
\definecolor{dkgreen}{rgb}{0,0.6,0}
\definecolor{gray}{rgb}{0.5,0.5,0.5}
\definecolor{pink2}{rgb}{1,0.4,0.8}
```

Precis efter `\begin{document}` skriver vi nu:

```
\lstset{language=Fortran,
basicstyle=\ttfamily,
extendedchars=true,
keywordstyle=\color{blue},
commentstyle=\color{dkgreen},
stringstyle=\color{pink2},
numbers=left,
numberstyle=\tiny\color{gray},
stepnumber=1,
numbersep=10pt,
backgroundcolor=\color{white},
tabsize=4,
showspaces=false,
showstringspaces=false}
```

För att infoga källkoden skriver vi nu

```
\lstinputlisting[language=Fortran]{Exempel/mult.f90}
```

av vilket resultatet blir:

```
1  subroutine mult(re_a, im_a, re_b, im_b, re_c, im_c)
2      ! This routine multiplies two complex numbers re_b+i*im_b
3      ! and re_c+i*im_b and saves the result in the variables
4      ! re_a and im_a
5
6     implicit none
7     double precision          :: re_a, im_a, re_b, im_b, re_c, im_c
8
9     re_a = re_b*re_c - im_b*im_c
10    im_a = re_b*im_c + re_c*im_b
11 end
```

givet att filen filnamn.f90 ligger i mappen Exempel som ligger i samma mapp som vår .tex-fil. Vi kan även backa i filsystemet genom att skriva ../Exempel/filnamn.f90 om vi exempelvis haft en mapp som innehållit både mappen Exempel och en annan mapp som innehöll vår .tex-fil.

Motsvarande förfarande, men med en MATLAB-fil ger:

```
% Plots a 3d Mobius strip

if nargin~=2
    u = [0 2*pi];
    v = [-0.4 0.4];
end

u = linspace(u(1),u(2),100);
v = linspace(v(1),v(2),20);

[u,v] = meshgrid(u,v);
x = cos(u) + v.*cos(0.5*u).*cos(u);
y = sin(u) + v.*cos(0.5*u).*sin(u);
z = v.*sin(0.5*u);

h = surf(x,y,z);

hold on;
axis tight;
grid off;
material([0.8,0.6,0.5,30]);
light('Position',[200,-20,-20]);
light('Position',[-20,300,-20]);
colormap summer;
shading interp;
lighting phong;
axis off;
hold off;
```

6.4 En grå box bakom källkoden

I det här dokumentet har vi valt att markera alla sektioner som innehåller LaTeX-källkod med en ljust grå bakgrund så att den skall synas ännu tydligare än vad den gör om man bara byter teckensnitt, så som sker med både *verbatim*- och *listing*-omgivningar.

Om man använder en *listing*-omgivning så kan man åstadkomma detta genom att vid importen av källkoden skriva:

```
\definecolor{gray}{rgb}{0.95,0.95,0.95}
\lstinputlisting[language=Fortran,
backgroundcolor=\color{gray},
xleftmargin=6pt,
xrightmargin=6pt,
framesep=6pt,
frame=single,
rulecolor=\color{gray95}]
{Exempel/mult.f90}
```

Vi kan också välja att vid resten av inställningarna för ett specifikt språk skriva:

Observera att vi genom att ange inställningar vid infogningen av källkod kan skriva över inställningar för ett specifikt språk som gjordes i början av dokumentet.

```
\lstset{language=Fortran,
basicstyle=\ttfamily,
extendedchars=true,
keywordstyle=\color{blue},
commentstyle=\color{dkgreen},
stringstyle=\color{pink2},
numbers=none,
stepnumber=1,
numbersep=10pt,

backgroundcolor=\color{gray},
xleftmargin=6pt,
xrightmargin=6pt,
framesep=6pt,
frame=single,
rulecolor=\color{gray95}

tabsize=4,
showspaces=false,
showstringspaces=false}
```

Då räcker det att när man vill infoga källkod skriva:

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

Resultatet blir enligt nedan:

```
subroutine mult(re_a, im_a, re_b, im_b, re_c, im_c)
    ! This routine multiplies two complex numbers re_b+i*im_b
    ! and re_c+i*im_b and saves the result in the variables
    ! re_a and im_a

   implicit none
   double precision          :: re_a, im_a, re_b, im_b, re_c, im_c

   re_a = re_b*re_c - im_b*im_c
   im_a = re_b*im_c + re_c*im_b
end
```

Att lösa det här problemet för *verbatim*-omgivningar är betydligt knepigare. I det här dokumentet har vi löst det på följande sätt: För att slippa skriva samma sak flera gånger så börjar vi med att definiera två konstanter. Vi definierar även färgen *gray* samt importerar paketet *fancyvrb*, precis innan `\begin{document}`:

```
\usepackage{fancyvrb}
\definecolor{gray}{rgb}{0.5,0.5,0.5}
\newcommand{\verbmargins}{6pt}
\newcommand{\verbalpha}{10}
```

Syftet med konstanten `\verbmargins` är att spara hur mycket marginal vi vill ha i våra gråa boxar. Vi har valt att ha marginaler på 6 punkter, och hade lika gärna kunnat skriva det direkt när vi infogar kod, men genom att spara värdet som en variabel så kan vi lätt ändra marginalerna

Det går givetvis även att skapa andra konstanter på samma sätt.

överallt genom att bara ändra värdet på variabeln. Variabeln `\verbalpha` sparar vilken grad av transparens vi vill använda.

För att få till den gråa boxen kring vår källkod skriver vi nu:

```
\begin{SaveVerbatim}{cverbatimtext}
[Placera din kod har]
\end{SaveVerbatim}
\colorbox{gray!\verbalpha}{\parbox{\textwidth}
{\vskip \verbmargins \leftskip\verbmargins \rightskip \verbmargins
\BUseVerbatim{cverbatimtext} \vskip \verbmargins }}
```

Resultatet blir då följande:

```
[Placera din källkod här]
```

Kapitel 7

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.

7.1 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 arbetar med LaTeX är att de ej har kontroll över vart bilderna hamnar. När man infogar en bild försöker LaTeX att placera bilden på ett så *bra* ställe som möjligt. Anledningen 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, och ibland dessutom i en annan ordning, än de 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...

Observera att bokstäverna här är gemener.

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

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

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

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 den gör den här typen av typsättning åt en, och oftast riktigt snyggt. Ofta blir slutresultatet därför mycket 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 [...]*.

7.2 Radbrytningar

LaTeX bryter rader där den 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 LaTeX markerar 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 ***quattro-decim*** *quindecim septendecim duedeveginti uneviginti viginti*

7.3 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 var figurer hamnar i LaTeX så finns det en risk att en figur hamnar långt efter vart man avsett placera den. Oftast vill man 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.

7.4 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}
```

7.5 Mellanrum i matematiska formler

När man skriver matematiska formler i LaTeXså tolkas blanksteg bara 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:

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

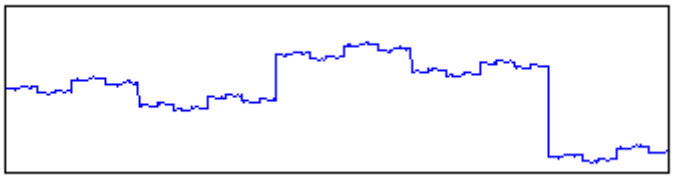
7.6 Horisontella och vertikala mellanrum i dokumentet

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

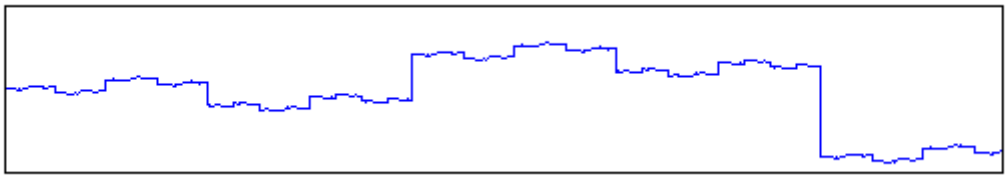
1. Av rent typografiska skäl, dvs. för att man vill åstadkomma en effekt.
2. För att fixa typsättning som inte blev så snygg när källkoden kompilerades.

Horisontella mellanrum görs med kommandot `\hspace{}`, var vi innanför klamrarna anger hur mycket utrymme vi vill ha. Kommandot `\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].

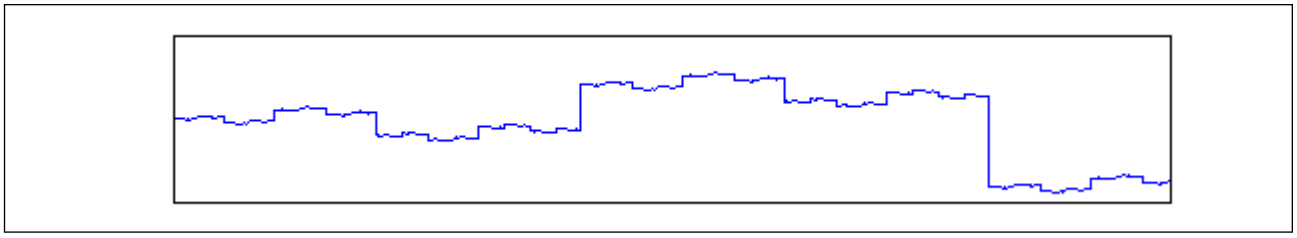
Ett exempel där det kan vara smidigt att använda kommandot `\hspace{}` är när man vill centrera en lite större bild i sitt dokument. Genom att lägga till kommandot `\centering` inuti en *figure*-omgivning så centreras en bild som är *tillräckligt liten*:



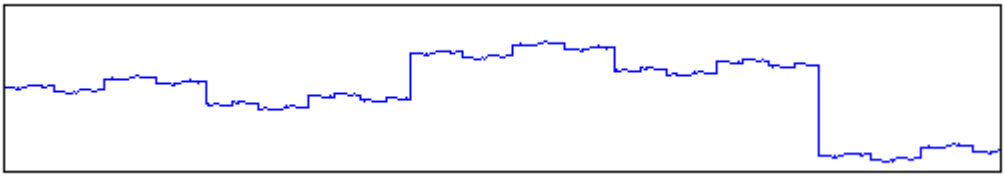
Ofta är dock inte bilden så liten att den får plats helt, vilket gör att kommandot `\centering` inte får någon som helst effekt, och bilden blir osymmetriskt placerad:



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 stor, 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.



Lösningen är att lägga till `\hspace{-3cm}` innanför *figure*-omgivningen för att centrera bilden manuellt:



[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.

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.

Källkod för exemplen:

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

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

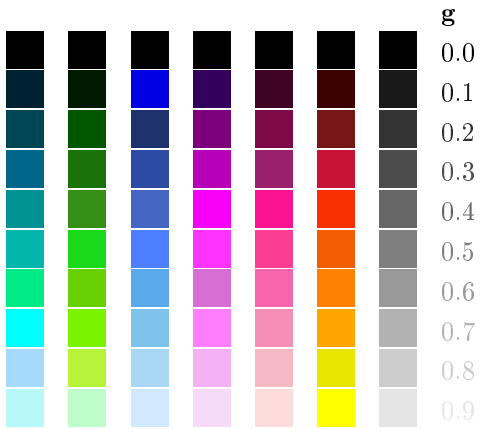
7.7 Färger

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 (g,b och b antas här vara tal mellan 0 och 1):

$$\mathbf{g} = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B \tag{7.1}$$

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



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.

7.8 Färger i tabeller

För att göra tabeller, och då framför allt stora tabeller, mer lättläsliga kan man låta olika rader ha olika bakgrundsfärg. För att åstadkomma detta så behöver vi först ange att vi vill använda paketet `xcolor` innan `\begin{document}`: `\usepackage[table]{xcolor}`

Precis efter `\begin{document}` definierar vi sedan vilka/vilken färg vi vill använda i bakgrunden; vi har valt att definiera färgen `gray`:
`\definecolor{gray}{rgb}{0.5,0.5,0.5}`

På den plats i dokumentet där vi vill ha vår tabell lägger vi nu till enbart en rad precis efter `\begin{table}`, nämligen:

```
\rowcolors{2}{gray!30}{gray!50}
```

Det vi har angett då är just att vi vill ha bakgrundsfärger på raderna med början från rad två; på varannan rad vår gråa färg med 30% transperans och på varannan rad samma grå färg men med 50% transperans. Total så ser hela koden för tabellen ut enligt nedan, och resultatet syns nedanför källkoden.

```
\begin{table}[htp]
  \rowcolors{2}{gray!30}{gray!50}
  \centering
  \begin{tabular}{| l  c  c | }
    \hline
    \bf Operation & \bf Title1 & \bf Title 2 \\ \hline \hline
    Addition & $4.1672918 $ & $5.7479853$ \\
    Addition-muliplication pair & $3.828598 $ & $3.1352778 $ \\
    Division & $1.5676365 $ & $1.5700978 $ \\
    Exponent & $2.5173724$ & $2.6161446 $ \\
    Sine & $2.1928352 $ & $2.2260226 $ \\
    \hline
  \end{tabular}
\rowcolors{1}{}{}
\end{table}
```

Operation	Title1	Title 2
Addition	4.1672918	5.7479853
Addition-muliplication pair	3.828598	3.1352778
Division	1.5676365	1.5700978
Exponent	2.5173724	2.6161446
Sine	2.1928352	2.2260226

7.9 Att tänka på för elektroniska pdf-filer

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 tre kommandon gör att innehållsförteckning och alla referenser blir klickbara, vilket gör det enklare att navigera i dokumentet för läsaren.

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

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.

7.10 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
```

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 8

Presentationer

Beamer [1] är ett dokumentformat i LaTeX som kan användas för att skapa presentationer. Presentationen skapas i en `.tex`-fil och kompileras sedan 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 samt lite längre ner; importen av paketet *beamerthemeshadow*.

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

```
\documentclass{beamer}
\usepackage[T1]{fontenc}
\usepackage[swedish]{babel}
\usepackage[utf8]{inputenc}
\usepackage{amssymb,amsmath}
\usepackage{graphicx}
\usepackage{listings}
\usepackage{courier}
\usepackage{color}

% Val av layout för presentationen
\usepackage{beamerthemeshadow}

\definecolor{dkgreen}{rgb}{0,0.6,0}
\definecolor{gray}{rgb}{0.5,0.5,0.5}
\definecolor{pink2}{rgb}{1,0.4,0.8}

% För att kunna skriva källkod i presentationen
\lstset{language=C,
basicstyle=\footnotesize,
basicstyle=\ttfamily,
keywordstyle=\color{blue},
commentstyle=\color{dkgreen},
stringstyle=\color{pink2},
numbers=none,
numberstyle=\tiny\color{gray},
stepnumber=1,
numbersep=10pt,
backgroundcolor=\color{white},
tabsize=4,
showspaces=false,
showstringspaces=false,
extendedchars=true}

% Definierar titel, författare och datum för presentationen.
% \today kan ersättas med ett fixt datum.
\title{Exempelpresentation}
\author{Niklas Andersson, Malin Palö}
\date{\today}

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

Ovanstående ger en helt tom presentation, till vilken vi nu skall lägga den kod som bygger upp vår faktiska presentation.

Innehållet i presentationen delas upp i *frames*, där en frame utgör en sida i presentationen. Varje frame 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.

8.1 Titelsida

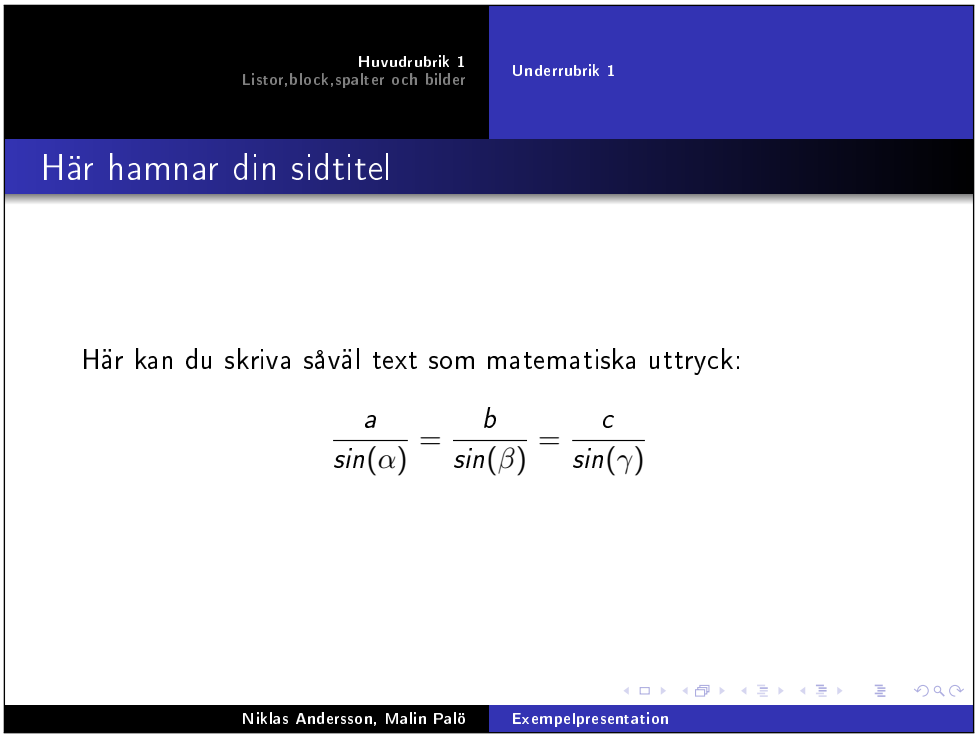
```
\begin{frame}
\titlepage
\end{frame}
```

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

8.2 Matematisk text

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

\begin{frame}
\frametitle{Här hamnar din sidtitel}
Här kan du skriva såväl text som matematiska uttryck:
$$\frac{a}{sin(\alpha)}=\frac{b}{sin(\beta)}=\frac{c}{sin(\gamma)}$$
\end{frame}
```

`\section{}` och `\subsection{}` används för att skapa titlarna som syns längst upp på sidan. Det som skrivs i `\section{}` hamnar till vänster och det som skrivs inom `\subsection{}` hamnar till höger. Notera att dessa kommandon ska ligga utanför `\begin{frame}` och `\end{frame}`. Kommandot `\frametitle{titel}` skapar titeln för den aktuella sidan. För att skriva matematiska uttryck används precis samma kommandon som när man skapar ett dokument i LaTeX, och utskriften blir precis som förväntat.



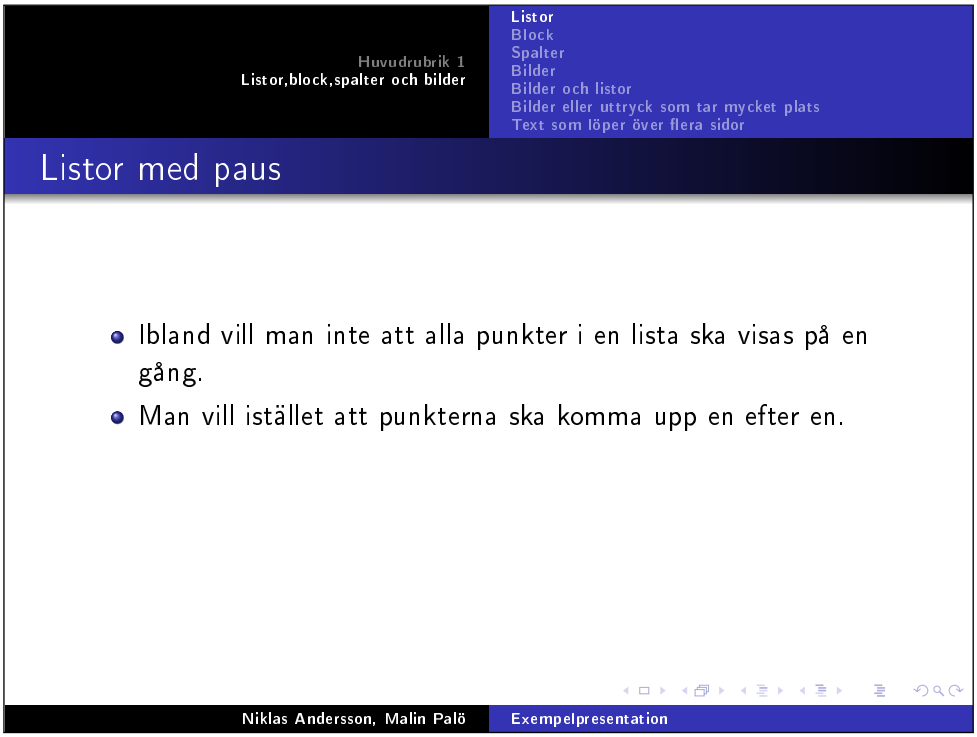
8.3 En punktlista med ”pause”

En vanlig situation är att man har en lista i vilken man vill att punkterna ska komma upp en i taget. Detta åstadkommer man genom att i en helt vanlig lista placera kommandot `\pause` efter varje `\item` i listan man skapat där man vill att det ska vara en paus.

```
\section{Listor,block,spalter och bilder}
\subsection{Listor}

\begin{frame}
\frametitle{Listor med paus}
\begin{itemize}
\item Ibland vill man inte att alla punkter i en lista ska visas
      på en gång. \pause
\item Man vill istället att punkterna ska komma upp en efter
       en. \pause
\item Detta åstadkommer man lätt med kommandot
      $ \setminus pause$ \pause
\item Tänk dock på att inte missbruka pausekommandot - kanske det
      är bättre att visa hela listan direkt?
\end{itemize}
\end{frame}
```

Resultatet av ovanstående kod blir nedanstående fyra sidor. Notera att vi bara har ett par av `\begin{frame}` och `\end{frame}` i koden, men kompilatorn genererar fyra sidor efersom vi använder `\pause`-kommandot.

Figur 8.1: 3a



Figur 8.2: 3b

Figur 8.3: 3c

Figur 8.4: 3d

8.4 Block

Definitioner, satser, och annan viktig information som man vill lyfta fram lite extra kan man lägga i ett eget s.k. block. Det finns 3 olika sorters block som man kan använda: `block`, `exampleblock` och `alertblock`.

```
\subsection{Block}
\begin{frame}\frametitle{Block}
Definitioner, satser och bevis och annan fakta kan
med fördel läggas i egna block för att belysas bättre.

\begin{block}{Titel för block 1}
Text för block 1 - blå bakgrund.
\end{block}

\begin{exampleblock}{Titel för block 2}
Text för block 2 - grön bakgrund.
\end{exampleblock}

\begin{alertblock}{Titel för block 3}
Text för block 3 - röd bakgrund.
\end{alertblock}

\end{frame}
```

8.5 Kolumner

Man kan dela upp sidor i 2 (eller flera) kolumner, t.ex. för att ha text i två spalter eller text i den ena kolumnen och en bild/tabell i den andra kolumnen.

Kommandona `\begin{columns}` respektive `\end{columns}` talar om var vi har spalter. Kommandona `\begin{column}{5cm}` respektive `\end{column}` talar om att innehållet mellan de kommandona hamnar i en kolumn som är 5 cm bred.

```
\subsection{Spalter}

\begin{frame}\frametitle{Spalter}
\begin{columns}
\begin{column}{5cm}
\begin{itemize}
\item En lista till vänster
\item Med en eller flera punkter
\item Och en tabell till höger
\end{itemize}

Eller bara helt vanlig text.
\end{column}

\begin{column}{5cm}

\begin{tabular}{|c|c|c|}
\hline
$\mathbf{A}$ & $\mathbf{B}$ & $\mathbf{A \wedge B} $ \\
\hline
S &  S & S \\
\hline
S &  F & F  \\
\hline
F &  S & F  \\
\hline
F &  F & F  \\
\hline
\end{tabular}

\end{column}

\end{columns}

\end{frame}
```

Spalter

- En lista till vänster
- Med en eller flera punkter
- Och en tabell till höger

Eller bara helt vanlig text.

A	B	$A \wedge B$
S	S	S
S	F	F
F	S	F
F	F	F

8.6 Bilder

Bilder i Beamer infogas precis på samma sätt som i ett vanligt LaTeX-dokument:

```
\subsection{Bilder}
\begin{frame}\frametitle{Bilder i Beamer}
\begin{figure}
\includegraphics[scale=0.5]{bild1.png}
\caption{En tämligen fantasilös bild.}
\end{figure}
Hej hej
\end{frame}
```

8.7 Overprint

Det kan finnas situationer då man vill ha flera sidor som är i princip identiska, men där endast någon bild eller mening ändras på varje. Ett dåligt sätt att lösa detta på är att manuellt skapa flera sidor med `\begin{frame}` och `\end{frame}` där man justerar de ändringar man vill ha från sida till sida.

Ett bättre sätt att lösa det på är att använda inbyggda kommandon i Beamer. `\item<3->` talar t.ex. om att den punkten i listan ska läggas till i den tredje sidan och sedan ligga kvar hela tiden. Kommandot `\item<3-5>` hade istället bara tagit med punkten i listan på sida tre till och med sida fem.

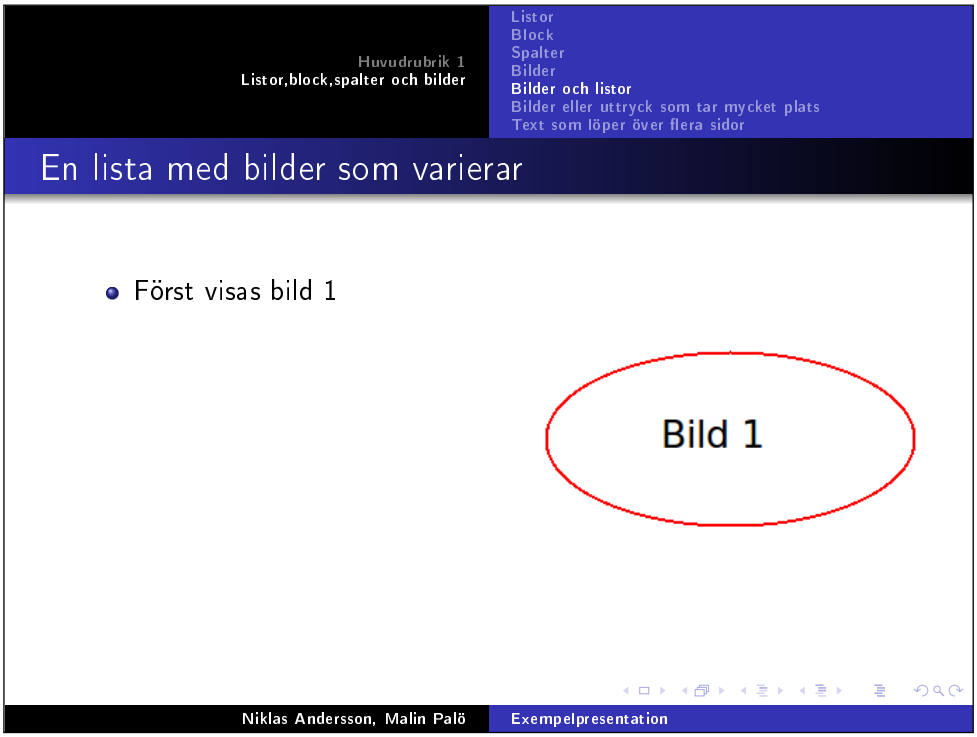
Kommandona `\begin{overprint}` resp. `\end{overprint}` används om man vill att element vara olika på olika sidor. I nedanstående exempel har vi gjort så att `bild1.png` visas på sida två, `bild2.png` på sida fyra och `bild3.png` på sida sex.

Nedanstående kod kommer alltså generera sex sidor i dokumentet, trots att vi bara har en `\begin{frame}` och `\end{frame}`.

```
\subsection{Bilder och listor}

\begin{frame}
\frametitle{En lista med bilder som varierar}
\begin{columns}
\begin{column}{5cm}
\begin{itemize}
\item<1-> Först visas bild 1
\item<3-> Sen visas bild 2...
\item<5-> ...och sist visas bild 3
\end{itemize}
\vspace{3cm}
\end{column}
\begin{column}{5cm}
\begin{overprint}
\includegraphics<2>[scale=0.7]{bild1.png}
\includegraphics<4>[scale=0.7]{bild2.png}
\includegraphics<6>[scale=0.7]{bild3.png}
\end{overprint}
\end{column}
\end{columns}
\end{frame}
```

Resultatet blir följande 6 sidor:

Figur 8.5: 3a

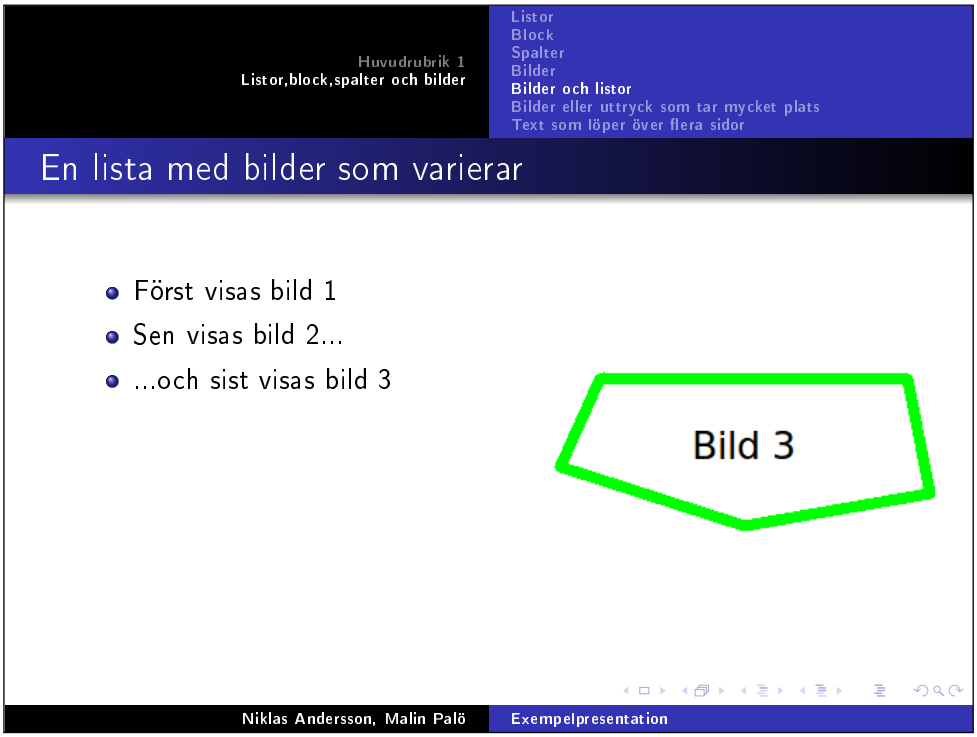


Figur 8.6: 3b

Figur 8.7: 3c

Figur 8.8: 3d

Figur 8.9: 3e

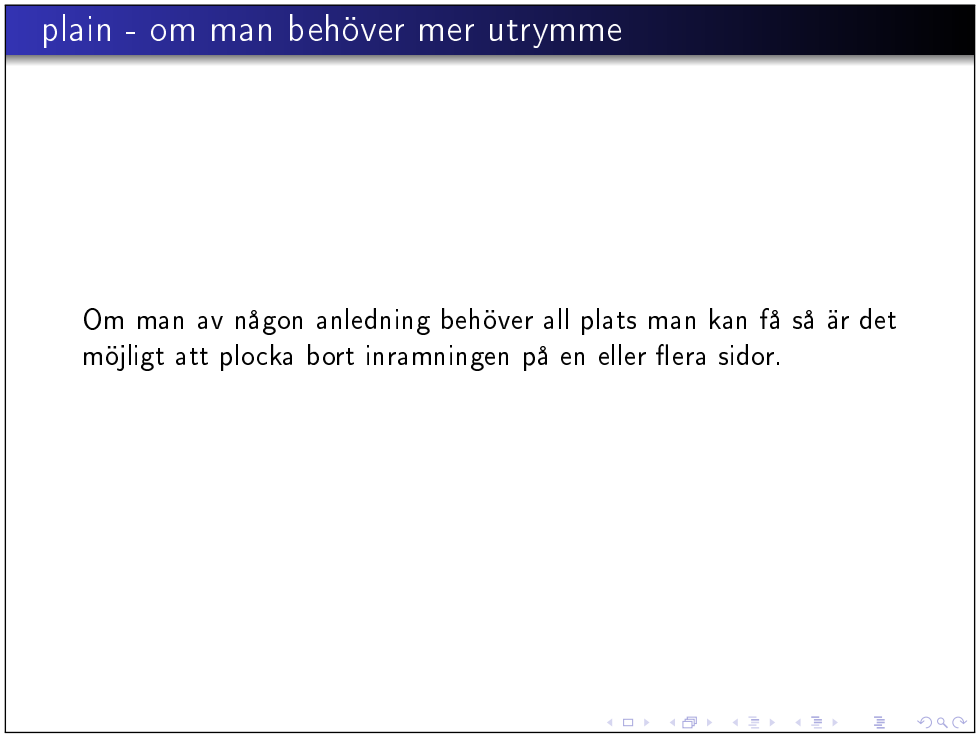


Figur 8.10: 3f

8.8 Plain

Om man har stora bilder eller stora ekvationer kan man vilja plocka bort inramingen för att kunna använda hela sidan. Detta åtstadkoms med kommandot `[plain]`, som skrivs direkt efter `\begin{frame}`.

```
\subsection{Bilder eller uttryck som tar mycket plats}
\begin{frame}[plain]
\frametitle{plain - om man behöver mer utrymme}
Om man av någon anledning behöver all plats man kan få så är det
möjligt att slocka bort inramningen på en eller flera sidor.
\end{frame}
```



8.9 Text som löper över flera sidor

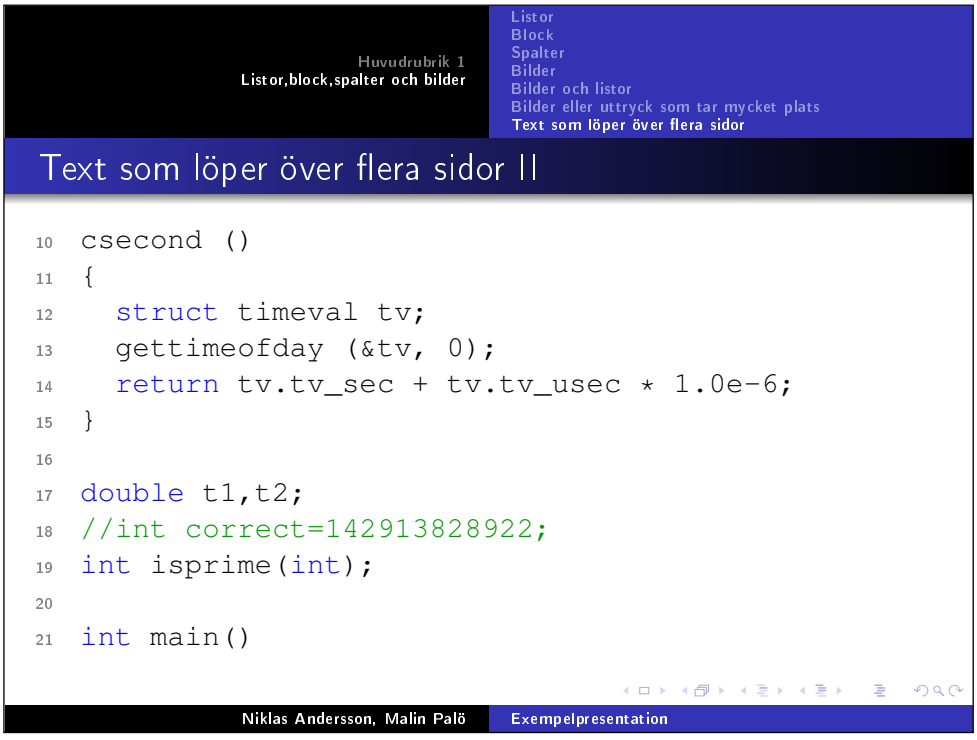
Om man har en lång text (exempelvis källkod) som inte får plats på en sida kan man göra på två sätt:

- själv dela upp texten på flera sidor, då hamnar sidbrytningen precis där man vill
- låta LaTeX och Beamer lägga in en sidbrytning där det behövs, då slipper man hålla koll på var sidbrytningen ska komma.

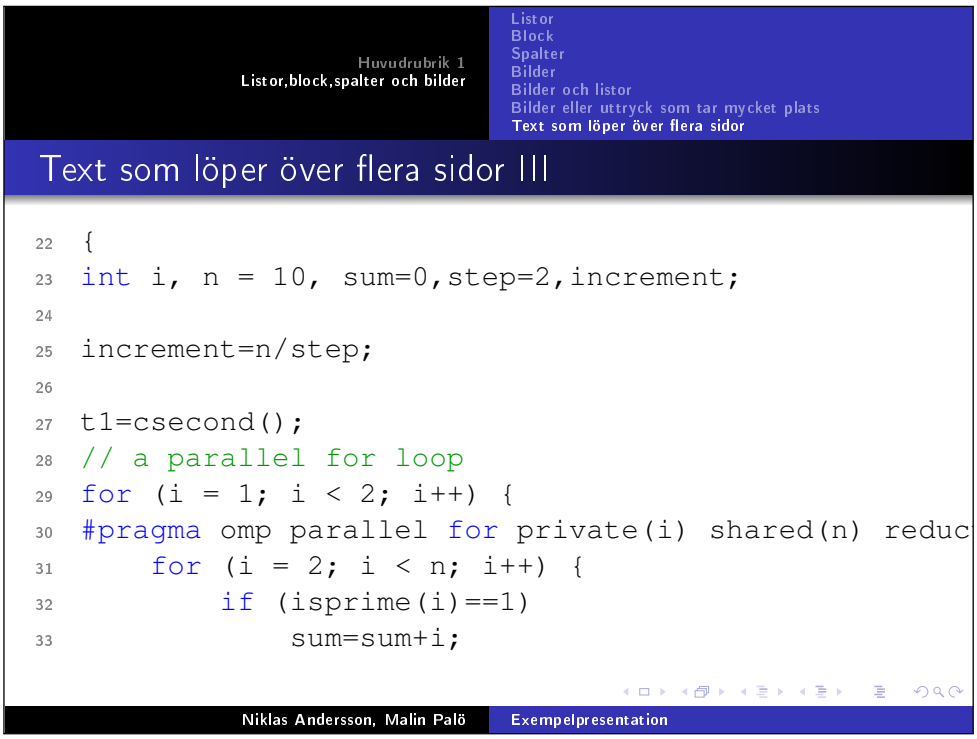
För att låta sidbrytningen ske automatiskt lägger man till `[allowframebreaks,t]` direkt efter `\begin{frame}`. Med kommandot `allowframebreaks` anger vi att vi tillåter texten löpa över flera sidor.

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

\begin{frame}[allowframebreaks,t]
\frametitle{Text som löper över flera sidor}
Här kommer lite kod som sträcker sig över flera sidor,
med automatisk sidbrytning

% Infogning av källkoden som ligger i filen eulerproblem10.c.
\lstinputlisting[numbers=left]{eulerproblem10.c}
\end{frame}
```

Figur 8.11: 3a



Figur 8.12: 3b



Figur 8.13: 3c

Figur 8.14: 3d

Om man istället har en lång text och själv vill lägga in sidbrytning kan man göra enligt följande: mellan `\begin{frame}` och `\end{frame}` läggs kommandot `\framebreak` in där man vill ha en sidbrytning. Notera att man fortfarande måste påbörja sidan i vilken du använder kommandot `\framebreak` med `\begin{frame}[allowframebreaks,t]`.

Om texten du har inte får plats på en sida och du inte har sagt till Beamer att byta sida med kommandot `\framebreak` kommer ett sidbyte ske automatiskt.

8.10 Olika layouter

Med kommandot `\usepackage{beamerthemeshadow}` anges vilken layout vi ville ha på våran presentation. Denna kan naturligtvis ändras, och det finns många olika layouter tillgängliga. Använd sökorden ”beamer themes” på valfri sökmotor och du hittar en uppsjö av layouter att välja mellan.

Kapitel 9

Kommentarer i dokumentet

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

Importera paketet genom att skriva:

```
\usepackage[colorinlistoftodos=true, textwidth=4cm]{todonotes}
```

mellan kommandona `\documentclass[10pt,a4paper]{article}` och `\begin{document}`. `\textwidth=4cm` anger att vi vill att kommentarerna skall vara 4 cm breda.

Använd något av kommandona `\todo{***}` eller `\todo[inline]{***}` för att skriva en kommentar direkt i dokumentet. Det andra alternativet ger en kommentar som ligger på en egen rad:

En till kommentar

Titta här! Det här är en kommentar av typ 1

Genom att skriva:

```
\listoftodos[Korrektur]
```

så får vi en lista med alla kommentarer; det 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:

Korrektur

Om man är flera personer som skriver på arbetet 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. Lägg följande kod precis innan `\begin{document}`:

```
\newcommand{\noteMalin}[2][]
{\todo[backgroundcolor=red!75!green!50!blue!25,
linecolor=red!75!green!50!blue!25!gray,
bordercolor=red!75!green!50!blue, #1]{#2}}
```

Genom att skriva det så har vi definierat ett nytt kommando; `\noteMalin`, som ser ut precis som vårt gamla kommando `\todo` bortsett från valet av färger. För att använda det skriver vi:

```
\noteMalin{Testkommentar 1}
```

eller, liksom tidigare:

```
\noteMalin[inline]{Testkommentar 2}
```

och får då resultaten: Testkommentar 1

Testkommentar 2

Bilaga A

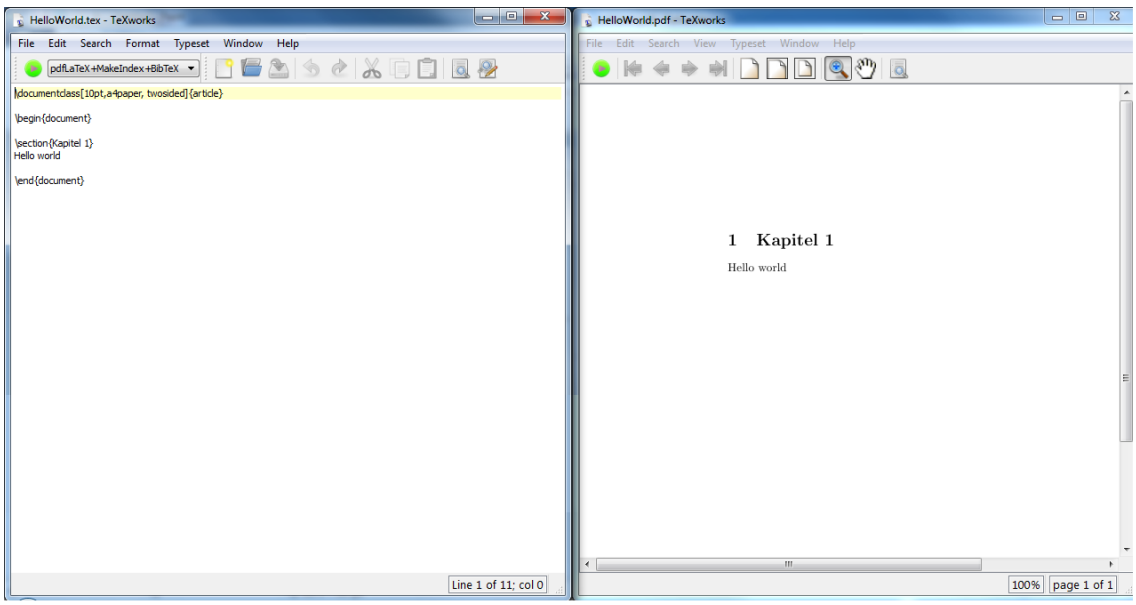
Editorer

Det finns en djungel av texteditorer man kan använda för att skapa källkoden till sina dokument, och vilken man till slut väljer 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

MikTEX(TEXWorks) 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.

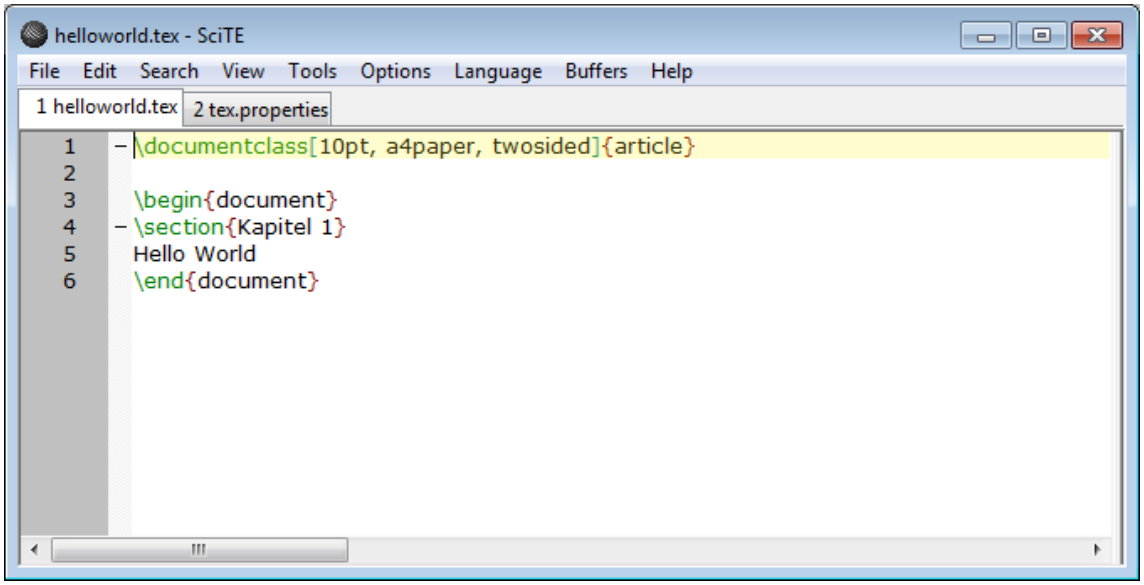


Figur A.1: Ett dokument i TEXworks: källkoden till vänster och det typsatta resultatet till höger.

MikTEX finns tillgängligt både för Linux, Mac och Windows (under namnet TEXworks) och är lätt att komma igång med.

A.2 Scite

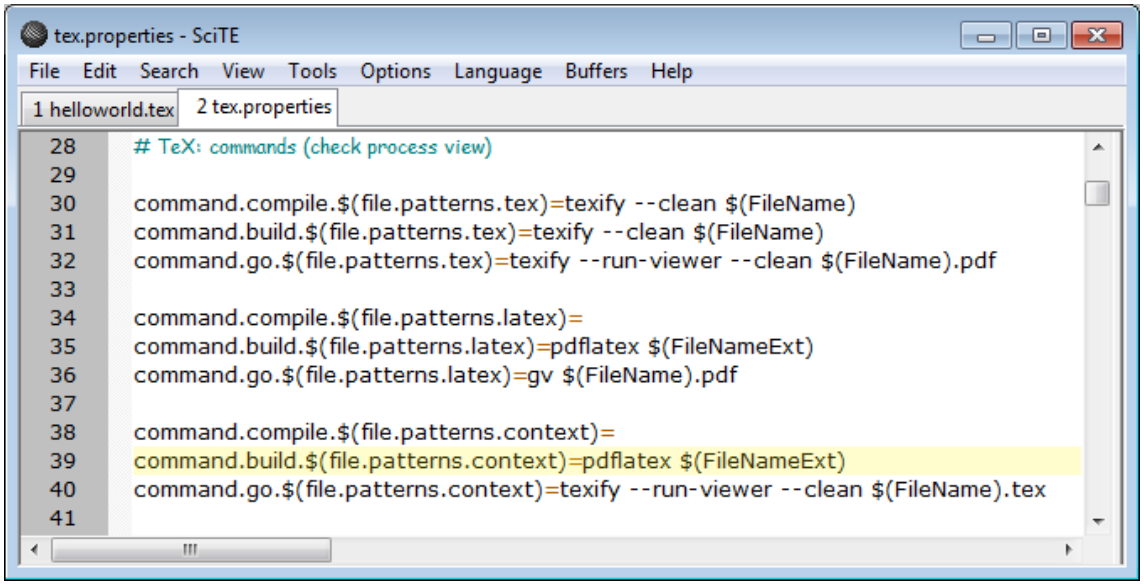
Scite är en editor som är anpassad för programmering, bland annat i LATEX. Scite finns gratis både för Linux och Windows, och finns som betalversion för Mac. När man sparar en fil med ändelsen `.tex` så känner Scite automatiskt igen att det är LATEX man skriver i, och möjliggör därmed kompilering via menyn. Om man inte vill kunna kompilera utan bara vill skriva i sin editor och exempelvis kompilera i terminalen, så krävs inga inställningar utan det är bara att sätta igång att skriva (se nedan).



Figur A.2: Ett dokument i Scite

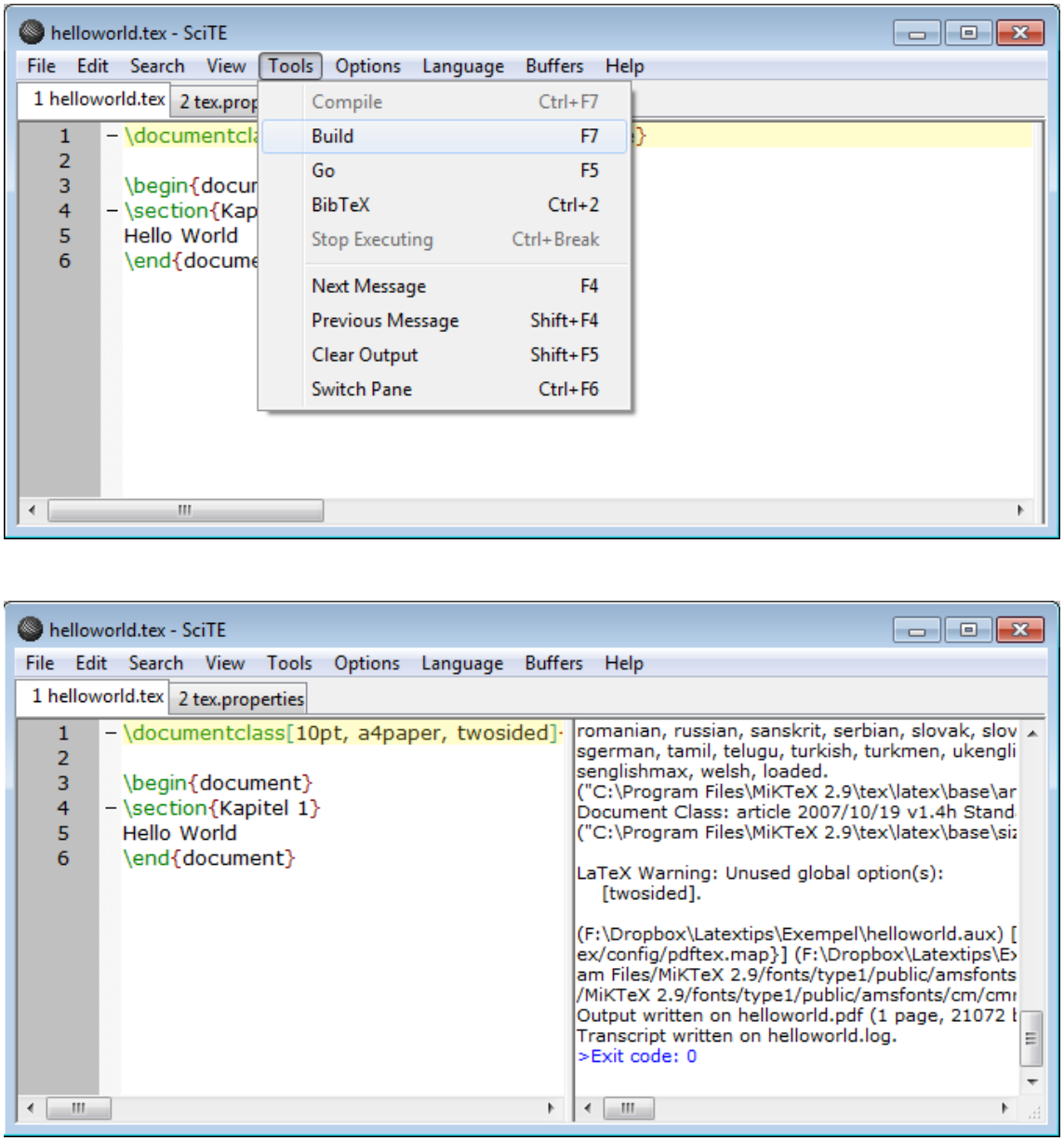
Förutom att det går att kompilera LATEX-dokument direkt från Scite, så har Scite många verktyg som kan vara smidiga att använda när man skriver källkod, så som källkodsfärgning, radnumrering, möjlighet att lägga till egna snabbkommandon osv. Scite har stöd för ett 20-tal olika språk, så programmerar man i fler språk än ett så är Scite trevlig att arbeta med.

Om man vill kunna kompilera direkt i editorn så krävs några inställningar. Öppna fillen med inställningar för TEX genom att klicka på *Open tex.properties* under *Options* i menyn högst upp på sidan. Under `#TeX: commands`, ändra de inställningar som står där till de vi angett nedan, eller din favoritkombination av kompilatorer, och spara sedan filen.



Figur A.3: Kompileringsinställningar i Scite

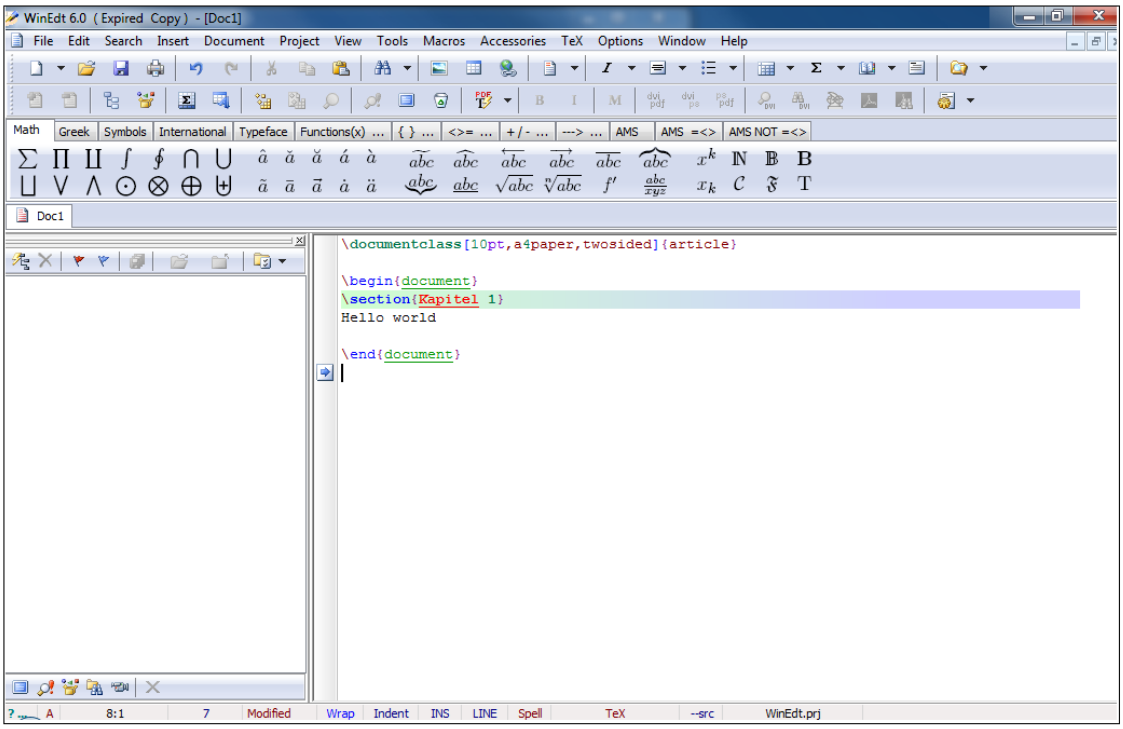
Under *Tools* kan du nu välja *Build* (F7) för att kompilera filen. Till höger dyker då en ruta upp som visar hur det går, vilket är samma information som du fått om du kompilerat direkt i terminalen.



Figur A.4: Kompilering direkt i Scite

A.3 WinEdit

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



Figur A.5: Ett dokument i winedit

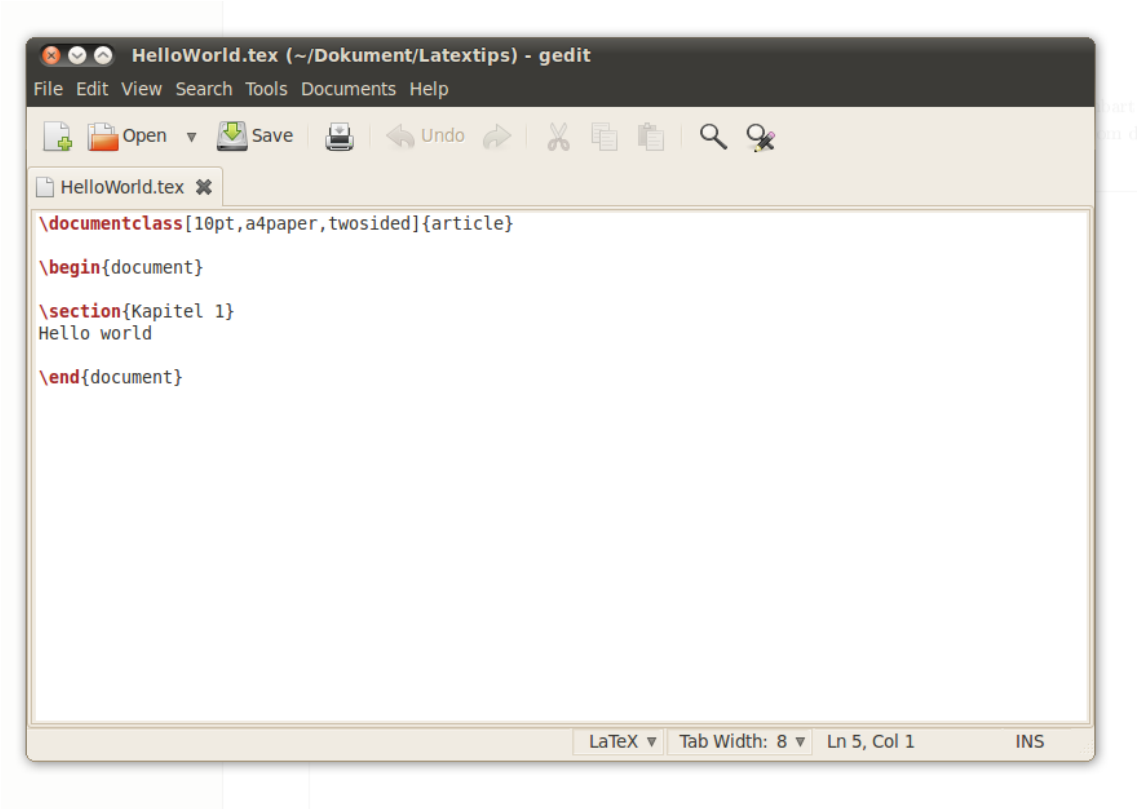
Winedit har ett inbyggt interface som gör det enkelt att infoga symboler man inte kan LaTeX-kommandon för. Precis ovanför textfältet finns nämligen en meny som innehåller diverse möjliga symboler, och för att infoga någon av dessa i sitt dokument räcker det att klicka på motsvarande symbol.

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

WinEdit väldigt lätt att komma igång med, och innehåller även en mycket god dokumentation. WinEdit går att ladda ner från Chalmers/GU:s programvaruservrar.

A.4 Gedit

Gedit är en enkel editor som inte är kopplad till LaTeXpå något sätt, utan är anpassad enbart för att redigera text. Det finns således inte någon inbyggd kompilator i gedit på samma sätt som det finns i TeXworks och WinEdit.



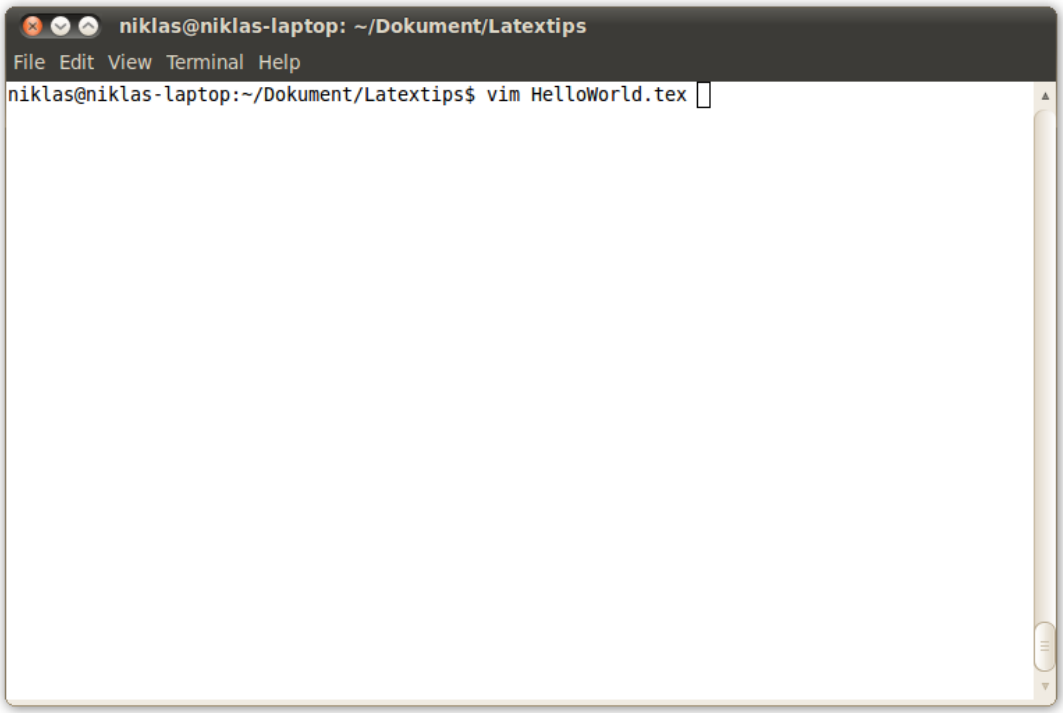
Figur A.6: Ett dokument i gedit

Gedit är en editor som är lätt att komma igång med om man inte har så stor erfarenhet av texteditorer. I gengäld är det dock inte lika flexibel som exempelvis vim eller emacs, men den fungerar mer än väl om man bara ska skapa enkla dokument. Man kan exempelvis inte definiera egna kommandon i gedit, något som man kan i vim och emacs.

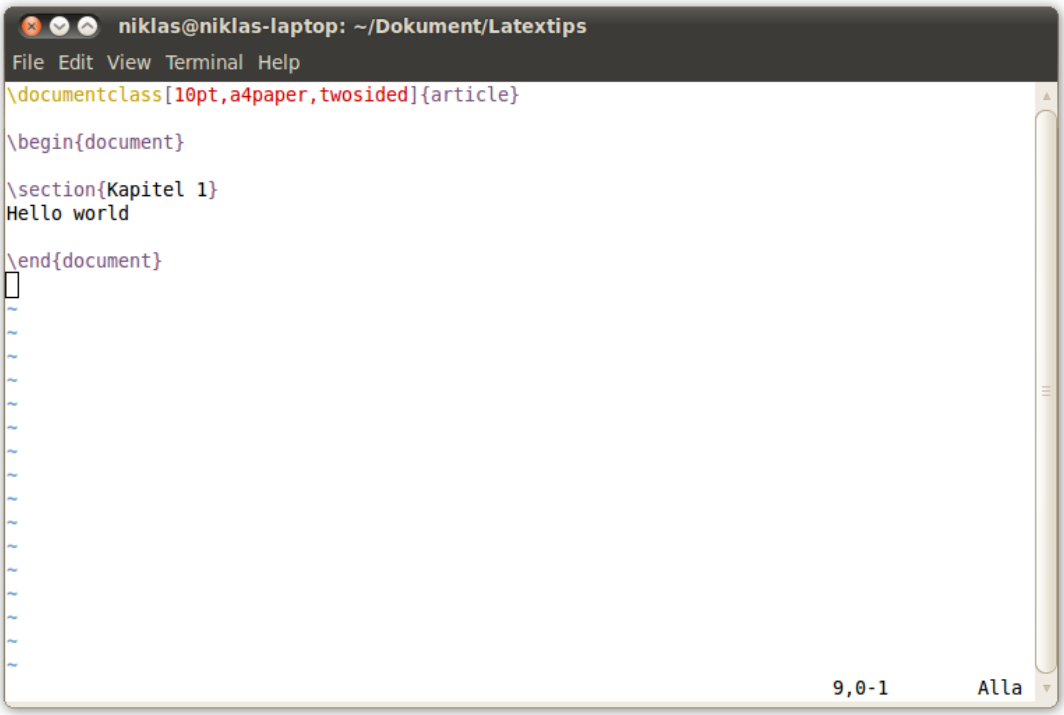
A.5 Vim

Vim är en editor som körs via terminalfönstret i Linux. Den har inget grafiskt användargränssnitt, utan allt man vill göra får man göra via kommandon i terminalfönstret. Vim passar främst bäst för den som är van vid Linuxmiljön och att använda tangentbordet snarare än musen för att åstadkomma saker.

För att öppna Vim, skriv *vim dokuemtnamn.tex* i terminalfönstret. Om filen *dokumentnamn.tex* finns öppnas den, om inte så skapas den.



Figur A.7: Öppna dokumentet *HelloWorld.tex* i Vim.



Figur A.8: Ett dokument i Vim

För att starta inmatningsläget, vilket du behöver göra för att kunna skriva in text i ditt dokument ger du komanndot *:i* eller *:a*. Du kan nu skriva in text som i vilken editor som helst. För att lämna inmatningsläget trycker du på Esc-tangenten.

Notera att du inte kan använda musen för att klicka i dokumentet, utan du får flytta markören med piltangenterna.

När du inte står i inmatningsläget kan du ge kommandon till Vim. En lista över de asolut nödvändigaste kommandona för att komma igång samt deras funktion ges i tabell A.1. En mer detaljerad lista finns t.ex. här: `http://www.tuxfiles.org/linuxhelp/vimcheat.html`.

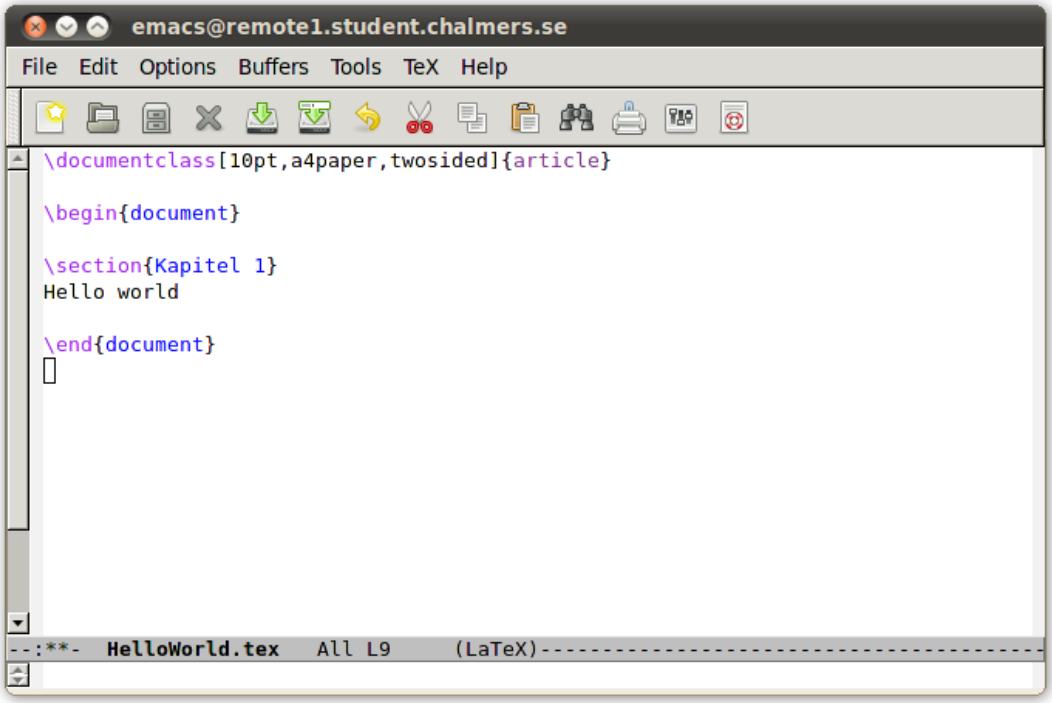
Kommando	**Funktion**
:w	Sparar alla ändringar
:q	Stänger Vim
:wq	Sparar alla ändringar först och stänger sedan Vim
q!	Stänger Vim utan att spara ändringar
i	Startar inmatningsläge
u	Ångra

Tabell A.1: Kommandon i Vim

En annan fördel med Vim är att det är lätt att definiera egna makron, dvs du kan lagra en sekvens av kommandon och textinmatning i ett nytt kommando som du döper själv. Detta kan vara praktiskt om man exempelvis ska göra samma sak eller liknande saker flera gånger. Macron i Vim är något som finns väl dokumenterat på internet.

A.6 Emacs

Emacs påminner till viss del om Vim i den mening att mycket styrs genom kommandon från tangentbordet snarare än val ur menyer med hjälp av musklick. Till skillnad från Vim är det dock möjligt att navigera i texten genom att klicka där man vill att markören ska hamna.



Figur A.9: Ett dokument i emacs.

Emacs finns väl dokumenterat på internet, en start för att komma igång med de mest nödvändigaste kommandona finns här: `http://www.cs.colostate.edu/helpdocs/emacs.html`.

Bilaga B

Källkoden till våra paket

B.1 matematik.sty

```
\ProvidesPackage{matematik}

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

B.2 svenska.sty

```
\ProvidesPackage{svenska}


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

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

B.3 engelska.sty

```
\ProvidesPackage{engelska}

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

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

B.4 formattering.sty

```
\ProvidesPackage{formattering}



\usepackage{url}%
\usepackage{hyperref}%
\hypersetup{colorlinks=false}%

\usepackage{verbatim}
\usepackage{fancyvrb}
\usepackage{listings}

\usepackage{graphicx}
\usepackage[table]{xcolor}

\usepackage{color}
\usepackage{xcolor}

\usepackage[colorinlistoftodos=true, textwidth=2cm]{todonotes}




\usepackage{subfigure} %
\usepackage{chngpage}
\usepackage{lipsum}
\usepackage{booktabs}
\usepackage{pdfpages}
\usepackage{float}
\usepackage{fancyhdr}
\pagestyle{plain}


\setlength\fboxsep{0pt}
```

```
\setlength\fboxrule{0.5pt}

\usepackage[parfill]{parskip}
\setlength{\parfillskip}{10\p@ \@plus 1fil}

\usepackage[hmargin=3cm]{geometry}
\addtolength{\topmargin}{-.875in}
\addtolength{\textheight}{1.75in}

\usepackage{setspace}
\setstretch{1.2}
```

Bilaga C

Länkar och fler tips

C.1 Länktips

`http://en.wikibooks.org/wiki/LaTeX/` -

`http://tex.stackexchange.com/` -

C.2 Paket vi inte hann ta upp i den här versionen

cleveref Möjliggör att genom att skriva `\cref{minsats}` få LaTeX att automatiskt känna av om det är en sats, ett lemma, en figur eller en tabell man refererar till, och därmed automatiskt i det här fallet skriva *sats* 4.6.2. Smidigt för långa dokument där man har många satser och lemman.

semiverbatim Ett bättre paket för källkod i *Beamer*-presentationer då mycket källkod skall infogas.