

INTRODUKTION TILL MAXIMA

JOHAN WILD

©Johan Wild 2010

johan.wild@europaskolan.se

Får gärna användas i undervisning, kontakta i så fall författaren.

20 september 2010

Innehåll

1	Inledning	4
2	Start	4
3	Matematik A	5
3.1	Aritmetik och enklare uttryck	5
3.2	Talteori	5
3.3	Uttryck och ekvationer	6
3.4	Funktioner och grafer	8
4	Funktioner i Maxima	9
5	Matematik B	9
5.1	Polynom och algebra	9
5.2	Ekvationer med flera lösningar	10
5.3	Flera lösningar - En fallstudie	11
5.4	Ekvationssystem	12
6	Loopar, villkor och filer	13
6.1	Loopar	13
6.2	Loop med villkor	14
6.3	Spara resultatet till en fil	14
7	Matematik C	15
7.1	Rationella uttryck	15
7.2	Logaritmer	16
7.3	Derivata	16
7.4	Talföljder, serier och summor	17
8	Matematik D	17
8.1	Trigonometriska funktioner och ekvationer	17
8.2	Integraler	18
9	Matematik E	19
9.1	Komplexa tal	19
9.2	Polynom och ekvationer	20

1 Inledning

Denna text syftar till att introducera programmet Maxima.

Maxima är ett program som klarar av att hantera matematiska uttryck symboliskt. Det betyder att det till exempel kan lösa ut x i uttrycket $ax + b = c$, där a , b och c alltså inte behöver vara bestämda tal (eller ens tal). Detta skall jämföras med andra program som kan använda sig av någon numerisk metod för att *räkna* ut vad lösningen till ekvationen $2x - 3 = 4$ är. Vidare svarar Maxima med en exakt lösning om så är möjligt. Till exempel avrundar Maxima inte $1/3$ till $0,3333$ eller andra tal.

Maxima klarar även att faktorisera, rita grafer, lösa ekvationer, derivera, integrera och mycket mera. Den här introduktionen är skriven så att den tar upp funktioner i ungefär samma ordning som motsvarande begrepp tas upp i gymnasiets kurser.

Det finns ingen möjlighet att introducera alla Maximas funktioner i en text som denna, men om du försöker se ett mönster i hur man använder funktionerna och kan nog snart pröva dig fram och förstå programmets hjälpfunktioner så att du lyckas med vad du vill göra.

I detta häfte är text skriven i **detta typsnitt** instruktioner till Maxima i en eller annan form. Det är vanligt i texter om programmering.

2 Start

Denna text gäller version 5.19 av Maxima. Under årens lopp har framför allt sättet att ge kommandon ändrats något. Den grundläggande strukturen har dock varit den samma sedan många versioner tillbaka. Om du använder en tidigare eller senare version än 5.19 kan det tänkas att det som följer härnäst inte riktigt är sig likt, men du kommer nog igång i alla fall.

När du startar Maxima ser du ett fönster, som till största delen är tomt. Skriver du något dyker det upp en röd markering till vänster. Det är meningen att du skall skriva olika kommandon, till exempel kan du skriva `a:3` (vilket betyder att du bestämmer att symbolen `a` skall ha värdet 3) och trycka på *både* SHIFT *och* RETURN samtidigt. Då ändras raden du just skrev till `(%i1) a:3;` (observera semikolonet!) och Maxima återger resultatet på formen `(%o1) 3`.

Meningen med detta är att Maxima ger dina instruktion och dess resultat ett löpnummer som sedan kan användas. Du kan nu som nästa instruktion skriva `%o1+2` varefter resultatet blir 5, eftersom `%o1` var 3. Detta är dock sällan man i praktiken använder Maxima på detta sätt.

Viktigare är däremot att du kan skriva flera instruktioner i ett moment. Om du bara trycker på RETURN efter att ha givit en instruktion, får du en ny rad att skriva på. Det som skiljer olika instruktioner åt är dock egentligen inte att de står på separata rader, utan att de skiljs åt med ett semikolon. Det var därför Maxima själv satte dit ett om du bara ger en instruktion.

Man kan ge en samling instruktioner i en textfil också, men det blir det mer om senare.

3 Matematik A

3.1 Aritmetik och enklare uttryck

Naturligvis kan Maxima räkna, konstigt vore väl annars. Det som du behöver lära dig är hur man skriver uttryck i Maxima. Hur man skriver uttryck brukar ibland benämnas *syntax*.

Du måste ange alla multiplikationstecken som man normalt utelämnar. Exponenter anges med symbolen \wedge . Om exponenten är mer än ett tecken långt måste du använda parenteser för att ange vad som är exponent.

Vissa symboler, till exempel π har egna namn i Maxima. Symboler som dyker upp i högre kurser i matematik presenteras då de behövs.

Tabellen nedan visar en rad uttryck och hur de skriv i Maxima.

Uttryck	Syntax	Kommentar
$(2 + x)(x + y)$	<code>(2+x)*(x+y)</code>	Observera att <code>*</code> är nödvändigt.
$\sqrt{2}$	<code>sqrt(2)</code>	Jämför med engelskans square root.
x^2	<code>x^2</code>	Exponenter med \wedge .
$2,5 \cdot 10^3$	<code>2.5*10^3</code>	Decimalkomma med <code>.</code> istället för <code>,</code>
$\frac{1}{2}$	<code>1/2</code>	Maxima räknar alltid med rationella tal.
2π	<code>2*%pi</code>	

3.2 Talteori

Det finns flera funktioner som har att göra med talteori. De mest grundläggande är att avgöra om ett tal är ett primtal eller inte, samt att faktorisera tal. Den funktionen heter `primep(n)` respektive `factor(expr)` och används där `n` är ett tal och `expr` är ett uttryck (till exempel ett tal). Exempel:

```
primep(9);
factor(9);
```

Funktionen `primep` svarar (vilket brukar kallas returnerar) `true` eller `false` om `n` är ett primtal eller inte. Funktionen `factor` visar vilka faktorer som `expr` innehåller.

Anledningen till att `factor` skall ges ett uttryck som argument och inte ett tal är att Maxima kan faktorisera fler saker än tal. Mer om det senare.

Övning 3.1. Pröva funktionerna `factor` och `primep` i några fall där du själv kan räkna ut svaren. Jämför med hur Maxima svarar så att du förstår hur du skall tolka resultaten.

Övning 3.2. Pröva funktionerna för allt större tal. Du kommer att finna att det går fortare att kolla om ett tal är ett primtal eller inte än det tar att faktorisera det. När blir detta märkbart? Varför är det så?

Övning 3.3. Man kan få hjälp om hur man använder funktioner på flera sätt. Ett sätt är att använda menyalternativet "Help". Ett snabbare sätt är att skriva ? och den funktion man vill ha hjälp om. Prova detta genom att skriva ? `primep` och ? `factor`. Det som då visas är kanske inte så lättläst. Informationen bygger på att man förstår en hel del matematik och är van att använda Maxima.

Maxima klarar även att ta fram den största gemensamma faktorn till två tal (eller uttryck). På engelska heter det `greatest common divisor` och funktionen heter `gcd` och används som i följande exempel.

```
gcd(12,18);
```

Funktionen som ger den minsta gemensamma multipeln av två tal heter `lcm` (least common multiple) och används på samma sätt som `gcd`.

Funktionerna som rör rest och kvot heter `divide`, `remainder` och `quotient`. Funktionen `divide` returnerar en lista som innehåller kvoten och resten. Listor betecknas med hakparenteser i Maxima. Om man är intresserad av bara endera kvoten eller resten måste man referera till listans första eller andra element.

Ofta är det smidigt att ge listan ett namn så att man har något att arbeta med. Om listan heter `L` så får man första respektive andra elementet i listan med `L[1]` och `L[2]`.

Övning 3.4. Testa och övertyga dig om att du förstår:

```
L:divide(15,6);
L[1];
L[2];
quotient(15,6);
remainder(15,6);
```

3.3 Uttryck och ekvationer

Funktionen för att lösa ekvationer heter `solve` och används såhär:

```
solve(2*x-2=4,x);
solve(a*x+b=c,x);
```

Som argument tar `solve` det ett uttryck, dels den variabel som skall lösas ut. Om uttrycket inte är en likhet utgår Maxima från att uttrycket skall sättas $=0$.

Övning 3.5. Observera att Maxima inte räknar ut lösningen till ekvationen, utan löser ut variabeln algebraiskt. Det är därför man måste ange vilken variabel som skall lösas ut. Prova följande.

```
solve(a*x+b=c,x);
solve(a*x+b=c,a);
solve(a*x+b=c,b);
solve(a*x+b=c,c);
```

Övning 3.6. Då Maxima ändå räknar så räknar Maxima exakt i den meningen att bråk inte hanteras som decimaltal med "många" decimaler (de flesta miniräknare räknar med 10 till 13 decimaler).

Det är till och med så att Maxima helst räknar med bråk. Prova följande.

```
solve(0.12*x=6,x);
```

Raden 'rat' replaced 0.12 by $\frac{3}{25} = 0.12$ ger information om att Maxima (eller snarare funktionen rat som används av solve) har skrivit om talet 0,12 till bråket $\frac{3}{25}$ varefter detta används istället.

Nu har du skrivit in flera kommandon. Om du skulle råka skriva fel behöver du inte skriva om hela raden igen. Du kan bläddra mellan senast inskrivna rader med piltangenterna.

Ofta har vill man göra mer med ett uttryck än att bara lösa en ekvation. Man kan därför ge uttryck namn enligt följande exempel.

```
u1:2*x-18;  
u2:4*x+b;  
solve(u1=u2,x);
```

Här ges alltså två uttryck namnen u1 och u2. När man sedan skall berätta för solve vilken likhet som x skall lösas ut ur, kan man använda dessa namn.

Om man jobbar med ett stort problem kanske man vill spara alla definierade uttryck så att man slipper börja om från början varje gång. Det går att göra, det finns ett alternativ som heter *Save* och *Save as* under *File* i menyn. När en gammal session öppnas körs alla kommandon man givit i den sessionen. Detta är ganska opraktiskt eftersom även alla mindre lyckade försök också sparas (man sitter ju ofta och testar än det ena än det andra då man jobbar med ett problem).

Ett betydligt bättre sätt att arbeta är att spara alla sina kommandon i en textfil, ett så kallat script. Du kan använda vilket program du vill för textredigeringen, men tänk på att spara filen i textformat med filnamnstillägget *.mac. (dokument i Word-format *.doc är helt oläsliga för Maxima).

I Maxima kan du sedan välja *Batch File* under *File* i menyn och välja vilken fil som skall köras.

En mycket viktig sak angående scriptfilerna är att varje instruktion måste sluta med ett semikolon (;) (en instruktion får alltså sträcka sig över flera rader).

Övning 3.7. Skapa en fil som heter `text1.mac` med följande innehåll.

```
u1:2*x-18;  
u2:4*x+b;  
solve(u1=u2,x);
```

Kör detta script i Maxima.

Fortsättningsvis kommer exempel i denna text visas som om de vore script (alltså med semikolon i slutet på varje instruktion).

För att script skall bli lite mer lättlästa (även efter en längre tid!) är det mycket lämpligt att man förklarar vem som har skrivit scriptet och varför. Det är också lämpligt att man anger vad olika variabler står för, precis som man skall då man löser problem på vanligt sätt. Om Maxima stöter

på teckenkombinationen `/*` någonstans i ett script så hoppar Maxima över allt som står till dess den stöter på kombinationen `*/`.

Vill man beräkna värdet av ett uttryck kan man skriva som i exemplet nedan.

```
u1:a+b;
u1,a=2,b=5;
```

Man anger alltså att Maxima skall återge ett uttryck, med ”tilläggs-informationen” att vissa variabler skall ha de värden man anger. Ett annat sätt är att definiera konstanterna som uttryck, Som i följande exempel.

```
/* Exempel konstruerat av Johan Wild 2005-11-14. */
/* En man kastar sig ut för ett torn som är 120 meter högt.
   Vilken hastighet har han efter tre sekunder om man bortser
   från luftmotståndet? */
/* v  Mannens hastighet i meter per sekund.
   t  Tiden i sekunder.
   g  Jordens tyngdacceleration.  */
v: g*t;    /* Hur v beror av t och g. */
g:9.82;    /* Jordens tyngdacceleration. */
v,t=3;    /* Värdet av v då t=3. */
```

Som framgår av exemplet är det bra att ha en genomtänkt strategi för hur och var man skriver sina kommentarer. Script skall vara lättlästa och vackra!

I exemplet ovan bildas alltså ett uttryck för mannens hastighet. Den beror på två variabler, g och t . Exemplet visar två sätt att beräkna värdet av hastigheten för olika val av g och t . Skillnaden är att g kommer att ha värdet 9,82 till dess man sätter något annat, medan variabeln t inte får något värde.

Efter att ha kört scriptet ovan skulle man alltså kunna fortsätta med att ge följande kommandon.

```
v,t=4;
g:9.82/6; /* På månen */
v,t=4;
```

De olika sätten att ge variabler värden är olika praktiska i olika sammanhang.

3.4 Funktioner och grafer

Man kan deklarera funktioner i Maxima, men det är betydligt enklare att hantera dem som uttryck. Nedan visas ett script som definierar ett uttryck och som ritar grafen till det.

```
f:2*x-3;
plot2d(f,[x,-3,6]);
```

Funktionen `plot2d` tar som argument det uttryck man vill rita, samt en lista som innehåller den variabel man avser och det intervall som man vill titta på.

Vill man rita grafen till flera funktioner kan man ge en lista på dem som första argument enligt följande exempel.

```
f:2*x-3;  
g:-3*x+2;  
plot2d([f,g],[x,-3,5]);
```

Den graf som visas är inte så fint formaterad. Det finns tekniker för att skala om grafen, ge grafen och axlarna rubriker och mycket mer. Dessa funktioner är inte så lätt att använda från Maxima. Vill man göra en informativ graf som man använda i något annat sammanhang är det bättre att använda ett program som har som huvudsyfte att göra grafer.

Ett enkelt sätt att få en x -axel är att konstanten noll skall ritas som en funktion:

```
plot2d([0,f,g],[x,-3,5]);
```

4 Funktioner i Maxima

Som sagt så är det oftast mest praktiskt att ge uttryck namn och tänka på dem som funktioner. Det finns dock fall där man kan ha nytta av Maximas skillnad mellan funktioner och uttryck. En funktion i Maxima definieras med konstruktionen $f(x) := x^2$. Här skall alltså parenteser med x i finnas med och man använder både ett `:` och ett `=`. Har man gjort en sådan konstruktion kan man skriva $f(2)$ till exempel.

Jag har inte hittat någon anledning att använda denna konstruktion i själva matematiken, det är oftast enklast att definiera uttryck. Däremot kan man göra egna funktioner i Maxima med denna konstruktion.

Fortsättningsvis kommer matematiska funktioner att definieras som uttryck, men då det behövs visas hur man kan definiera egna Maxima-funktioner.

5 Matematik B

5.1 Polynom och algebra

Ett uttryck på faktorform kan omvandlas till polynomform med funktionen `expand`. Man kan också ange att `expand` skall användas på ett sätt som liknar fallet när man skall beräkna värdet av uttryck.

```
p1:(x-1)*(x-2);  
p1,expand;  
expand(p1);
```

Maxima kan faktorisera polynom likväl som tal. Det är samma funktion som används för polynom, man kan som sagt ange ett uttryck som argument till funktionen `factor`.

Det är viktigt att notera att `factor` arbetar med polynom över de rationella talen. Med konjugatregeln kan du faktorisera polynomet $x^2 - 4$ till $(x - 2)(x + 2)$.

Du skulle även klara att faktorisera $x^2 - \frac{1}{4}$ till $(x - \frac{1}{2})(x + \frac{1}{2})$. Detta skulle Maxima hålla med om.

Vidare gäller $x^2 - 2 = (x - \sqrt{2})(x + \sqrt{2})$, men här skulle Maxima inte hålla med. Talet $\sqrt{2}$ är inget rationellt tal, därför tycker Maxima att polynomet $x^2 - 2$ är irreducibelt.

Övning 5.1. Övertyga dig om att du förstår ovanstående problematik genom att använda `factor` för att faktorisera $x^2 - 4$, $x^2 - \frac{1}{4}$ och $x^2 - 2$

Övning 5.2. Bilda ett polynom genom att välja några faktorer. Använd `expand` för att multiplicera ihop faktorerna till polynomform. Använd `factor` för att kontrollera att du får tillbaka de faktorer du utgick från. Exempel:

```
u1:(x-1)*(x+2);
u2:u1,expand;
factor(u2);
```

Du kan välja att skapa polynom av mycket högre grad än två, trots att det inte tas upp i Matematik B.

Funktionerna `gcd`, `divide`, `quotient` och `remainder` fungerar precis som för de hela talen. I själva verket är det samma funktion som används. Tal är ju på sätt och vis ett specialfall av polynom, ett tal är ett polynom av grad noll.

En liten detalj är att uttryck som

$$\frac{x^2 - 1}{x + 1}$$

inte automatiskt förkortas, till skillnad från uttrycket $\frac{6}{3}$ som automatiskt blir 2. Vill man förkorta kvoten mellan två polynom måste man använda en funktion som heter `ratsimp`. Det blir lite mer om det i avsnittet om funktioner som har med Matematik C att göra.

5.2 Ekvationer med flera lösningar

Andragradsekvationer (och ekvationer av högre grad) löser man också med `solve`. Det är bara att definiera ett uttryck av grad två (eller högre).

Till skillnad från `factor` så begränsar sig inte `solve` till rationella tal.

Övning 5.3. Lös ekvationerna $x^2 - 2 = 0$ och $x^2 + x - 2 = 0$ med `solve`.

En finess som kanske kan ställa till missförstånd i Matematik B är att `solve` inte heller begränsar sig till reella tal. Du kanske betraktar ekvationen $x^2 + 1 = 0$ som olöslig. Det är den också, om man söker lösningar bland de reella talen. Det finns däremot fler sorters tal där lösningarna ligger. De kallas komplexa tal, och tas upp i Matematik E. Där införs en symbol i för $\sqrt{-1}$. Den symbolen har namnet `%i` i Maxima.

Övning 5.4. Maxima vet ju inte vilken kurs i matematik du går, så om du försöker lösa ekvationen $x^2 + 1 = 0$ så kommer du att få se vad lösningarna är i Matematik E. Testa det.

Övning 5.5. Maxima klarar ju som bekant att lösa ut variabler ur uttryck med flera obekanta. Testa att lösa ekvationen $x^2 + px + q = 0$ med avseende på x så får du se.

Om man bara vill ha reella lösningar till ekvationer så finns det en funktion för det också, `realroots`. Nackdelen är att den bara klarar av att lösa ekvationer med en obekant. Exempel:

```
realroots(x^2-1);
realroots(x^2+1);
```

5.3 Flera lösningar - En fallstudie

Funktionen `solve` returnerar inte bara lösningen till en ekvation, utan en lista av lösningar (om det bara finns en lösning innehåller listan bara ett element). För att vara mer precis innehåller listan likheter. Minns att en lista betecknas med hakparenteser. Vill man använda sig av lösningarna i något sammanhang måste man trixa lite.

Ett standardproblem i Matematik B är att beräkna den maximala höjden som en partikel når om dess höjd över marken är given som funktion av tiden. Ofta kastas partikeln från en given höjd.

Antag att det rör sig om en taikonaut (kinesiska astronauter kallas så) som kastar en sten på månen (vi håller oss på månen för att slippa luftmotståndet). Taikonauten kastar stenen upp från månens yta med en utgångshastighet på 2,0 m/s och från höjden 1,2 m. Stenens höjd y m över marken efter t sekunder kan då beskrivas med

$$y(t) = -0,82t^2 + 2,0t + 1,2.$$

Detta problem löser man som bekant (i Matematik B) genom att beräkna två tidpunkter som ger samma höjd (till exempel höjden $y = 0$) varefter man då kan ta fram symmetrilinjen för parabeln som medelvärde av dessa tidpunkter. Stenens höjd kan då beräknas för denna tidpunkt.

Med Maxima skulle man göra såhär:

Först definierar vi uttrycket y som bara beror av t .

```
y:-0.82*t^2+2.0*t+1.2;
```

Sedan löser vi ekvationen $y(t) = 0$ med `solve`. Som sagt returnerar `solve` en lista med likheter. Denna lista skall sedan användas. Därför måste den ges ett namn, här L som i lösning.

```
L:solve(y=0,t);
```

Listan L innehåller nu två element. De kan man komma åt genom att referera till dem som $L[1]$ och $L[2]$. Men $L[1]$ och $L[2]$ är inte lösningarna, utan likheter som säger för vilka t som $y(t) = 0$.

Det är högerledet av dessa likheter som är själva lösningarna. De kan man komma åt med funktionen `rhs` (som i right hand side). Om lösningarna ges namnen `t1` och `t2` blir fortsättningen på scriptet såhär.

```
t1:rhs(L[1]);
t2:rhs(L[2]);
```

Uttrycken `t1` och `t2` är alltså högerledet av listan `L`'s första respektive andra position.

Nu kan man beräkna symmetrilinjen. Vi ger den tidpunkten namnet `ts`. Därefter beräknas värdet av `y` för detta val av `t`.

```
ts:(t1+t2)/2;
y,t=ts;
```

Maxima envisas med att svara så exakt det går. Vi har förmodligen ingen nytta av det uttryck som nu visas. För att omvandla det till ett tal på decimalform (men som alltså inte är exakt!) kan man lägga till numer på sista raden.

```
y,t=ts, numer;
```

I sin helhet blir scriptet såhär.

```
/* Beräkning av högsta höjd en sten når om
   den kastas på månen. */

y:-0.82*t^2+2.0*t+1.2;
L:solve(y=0,t);
t1:rhs(L[1]);
t2:rhs(L[2]);
ts:(t1+t2)/2;
y,t=ts, numer;
```

Man kan tycka att det vore bättre om `solve` bara returnerade lösningarna, men vi får leva med att Maxima gör som det gör. Då man löser ekvationssystem måste hur som helst Maxima tala om vilken variabel som får vilket värde, så det kanske inte är så dumt i alla fall.

Övning 5.6. Skriv in scriptet ovan rad för rad och begrunda vad resultatet blir så att du förstår vad som händer. Konstruktioner av typen `rhs(L[1])` är mycket vanlig då man vill *använda* värdet av en lösning senare i scriptet. Vill man bara *veta* vad lösningen är räcker det ju med att titta på vad listan innehåller.

5.4 Ekvationssystem

Ekvationssystem löser man också med `solve`. Som argument ger man då en lista med ekvationer och en lista med variabler man vill lösa dessa med avseende på. För att göra scriptet lite mer tydligt är det bra att ge de olika likheterna namn.

```
u1:2*x+3*y=4;
u2:3*x-5*y=7;
L:solve([u1,u2],[x,y]);
```

Övning 5.7. Gör ett script som löser ett ekvationssystem av den sort som finns i Matematik B (som det ovan alltså).

Notera att lösningen (det finns ju bara en skärningspunkt mellan två linjer) återigen presenteras som en lista (innehållet mellan de två yttre hakparenteserna). Denna lista gavs namnet `L` och har ett element (den enda lösningen). Men denna enda lösning är i sin tur en lista (innehållet mellan de inre hakparenteserna). Denna inre lista har två element: en likhet som säger vad x är och en som säger vad y är.

Ett lite större exempel som visar flera lösningar till ett ekvationssystem är att hitta skärningspunkterna för två parablar.

```
u1:y=x^2;
u2:y=(x/4)^2+3;
L:solve([u1,u2],[x,y]);
```

Övning 5.8. Skriv in scriptet ovan och studera resultatet. Här finns det alltså två lösningar. Den yttre listan `L` har nu två element, som båda har två element. Elementen i en lista av listor kan man komma åt genom följande konstruktioner. Skriv in dem och försök förstå vad som händer.

```
L[1];
L[1][1];
rhs(L[1][1]);
L[1][2];
rhs(L[1][1]);
rhs(L[1][2]);
```

Vill du rita parablarna kan du göra det med följande konstruktion. Notera att det är högerledet av `u1` och `u2` du vill rita (`u1` och `u2` är ju *likheter!*)

```
plot2d([rhs(u1),rhs(u2)],[x,-3,3]);
```

6 Loopar, villkor och filer

En av anledningarna till att människan uppfann datorn, var att man ville få datorn att lösa en uppgift om och om igen. Under årens lopp har det växt fram en syntax som ser snarlik ut i nästan alla programmeringsspråk. Som modellproblem väljer vi att lösa en andragradsekvation för olika värden på en av koefficienterna.

Övning 6.1. Lös ekvationen $x^2 - x - C = 0$ för alla värden på heltalet C mellan 2 och 102.

Om du verkligen har löst övningen ovan har du för mycket fritid!

6.1 Loopar

Låt oss se hur man kan använda Maxima istället. Konstruktionen som utför flera instruktioner heter `for ... do ...` och används som i följande exempel.

```

for C:2 thru 102 do (
  ekv:x^2-x-C=0,
  L:solve(ekv,x),
  print("Då C=",C," blir lösningarna ",L[1]," och ",L[2])
);

```

Detta uttryck skall du tolka såhär (det är nästan som att läsa på engelska): "För alla värden på C från och med 2 till och med 102 gör:". Sedan följer ett block med instruktioner, de mellan parentesen efter `do` och den avslutande parentesen.

Notera att det är `,` mellan instruktionerna som skall utföras inom blocket, och notera även det avslutande `;` som alltså gäller hela `for`-konstruktionen.

För att Maxima skall skriva ut lösningarna används funktionen `print`. Notera hur text kan kombineras med värdet av uttryck i Maxima.

Övning 6.2. Skriv in av scriptet ovan och testa det. Gör små förändringar så att du förstår hur allt fungerar.

6.2 Loop med villkor

Konstruktionen `for` används oftast då man vet att man vill göra något ett visst antal gånger.

Ibland har man nytta av att utföra något till dess ett visst villkor har uppfyllts. För det finns konstruktionen `while`. Följande lilla script beräknar den i :te roten ur 2 och slutar då $\sqrt[i]{2} - 1 > 0,1$.

```

i:1; /* Loop-variabel. Måste sättas till ett startvärde. */
d:0.1; /* Avstånd från 1. */
a:2; /* Det tal som den i:te roten skall beräknas från. */

p:100; /* För att få while-loopopen att komma igång. */

while p-1>d do(
  i:i+1, /* Stega upp i. */
  p:a^(1/i), /* Beräkna en ny rot av a. */
  print("Iteration nr ", i, " ger ",p, "=", ev(p,numer)) /* Sriv ut ett delresultat. */
);

```

Övning 6.3. Skriv in av scriptet ovan och testa det. Gör små förändringar så att du förstår hur allt fungerar.

Övning 6.4. Titta i Maximas hjälp-avsnitt om funktionen `do`.

6.3 Spara resultatet till en fil

Ibland har man nytta av att spara resultatet av en beräkning i en fil. Det är till exempel ganska bökigt att göra en snygg graf i Maxima, så det kan vara mer praktiskt att låta Maxima göra vissa beräkningar som sparas i en fil, och sedan använda Gnuplot för att visualisera resultatet av beräkningarna.

En fil kallas i sådana här sammanhang för en *stream*, som öppnas med funktionen `openw`. Det kommer att skapa en fil som skrivs över varje gång instruktionen körs. Ibland har man nytta av att lägga till data till en existerande fil. Då kan man använda funktionen `opena` istället (a som i *append*).

Funktionen `openw` öppnar i alla fall en ström, som i exemplet nedan ges namnet `stream`. Den kan sedan användas i funktionen `printf` (print som i skriv och f som i formaterad). Argumenten till `printf` är en ström och sedan en text inom situationstecken. Denna text kan innehålla tecknet `~` följt av ett eller två tal (åtskiljda med tecknet `,`) och sedan en bokstav. Bokstaven står för vilken sorts tal du vill skriva ut. I exemplet nedan används heltal (d som i *decimal*) och reella tal med decimaler (f som i *float*, vilket syftar på att kommatecknet inte har en fast position).

Det första (och därmed ibland det enda) talet efter `~` anger den ”bredd” som talet minst skall få (mätt i antal siffror) och det andra talet anger hur många tecken av dessa som skall reserveras åt decimaler.

Efter att man har använt sin stream/fil klart måste man stänga den med `close`.

```
stream:openw("c:\\Katalog\\Testfil.dat");

for i:1 thru 10 do (
  printf(stream,"~7i ~8,5f ~%",i,1/i^2)
);

close(stream);
```

Övning 6.5. Skriv in av scriptet ovan och testa det. Gör små förändringar så att du förstår hur allt fungerar. Observera att du måste ange hela sökvägen till den plats du vill att filen skall sparas på, annars sparar Maxima filen på en plats som kan vara svår att finna (beroende på operativsystem).

Övning 6.6. Titta i Maximas hjälp-avsnitt om funktionen `printf`.

7 Matematik C

7.1 Rationella uttryck

Som tidigare nämnt förenklar inte Maxima rationella uttryck automatiskt. Om täljare och nämnare inte är på faktorform så anropar Maxima inte `factor` på täljare och nämnare var för sig. Det finns en specialfunktion för att förenkla rationella uttryck, `ratsimp`, som ger täljare och nämnare på polynomform.

Övning 7.1. Definiera uttrycken

```
p1:(x+1)*(x+2)*(x+3);
p2:x^2-1;
```

och begrunda skillnaden mellan

```
p1/p2;
factor(p1/p2);
ratsimp(p1/p2);
```

7.2 Logaritmer

Maxima använder e som bas för funktionen $\log(x)$. Det finns funktioner för att förenkla uttryck med logaritmer, men det är inget som vi har nytta av i Matematik C, så vi fördjupar oss inte i det här.

Vill man beräkna \log_{10} av något kan man definiera en egen funktion i Maxima såhär: $\log_{10}(x) := \log(x)/\log(10)$.

Även exponentialfunktionen $\exp(x)$ har e som bas. För övrigt finns talet e definierat i Maxima som $\%e$. Att skriva $\%e^x$ är alltså samma sak som att skriva $\exp(x)$.

7.3 Derivata

Funktionen för att derivera i Maxima heter `diff`. Den används enligt följande mönster.

```
p:x^3+2*x+4*x-2;
Dp:diff(p,x);
D2p:diff(p,x,2);
```

Första argumentet till `diff` skall vara det uttryck man vill derivera, det andra den variabel man vill derivera med avseende på och det sista argumentet är derivatans ordning. Om man utelämnar det sista utgård Maxima från att det bara skall deriveras en gång.

Resultatet av `diff` på ett uttryck är ett nytt uttryck. För att det skall kunna användas måste det ges ett namn. För att hålla ordning på uttrycken och dess derivator är det praktiskt att ha en smidig konvention för namnen på uttrycken. Om `p` är ett uttryck brukar jag namnge dess derivata `Dp` och dess andraderivata `D2p`.

Scriptet nedan illustrerar hur man kan lösa ett standardproblem i Matematik C. Det handlar om ett papper som man klipper bort kvadrater i hörnen på. Resten viks ihop till en låda. Det som söks är lådans största volym.

```
/* Script som beräknar maximal och minimal volym för en låda som
viks av ett papper enligt standardproblem i Matematik C
Johan Wild 2008-09-24

V      Lådans volym.
a,b    Papprets mått.
x      Kvadratens sida = lådans höjd. */

/* Ett uttryck för volymen. */

V: x*(a-2*x)*(b-2*x);

/* Derivera V och sök nollställena till derivatan. */

DV: diff(V,x);      /* Volymens derivata med avseende på x. */
L: solve(DV=0,x);   /* L blir lösningarna till evkationen. */
L1: rhs(L[1]);      /* De egentliga lösningarna måste plockas fram. */
L2: rhs(L[2]);      /* L är ju bara en lista med likheter. */

/* Vilken av L1 och L2 som ger störst volym syns inte förrän tal sätts in.
Maxima fixar tyvärr inte att sätta in a och b i först L1 och
L2, och sedan detta i V.

Därför definieras två nya uttryck där x byts ut mot L1
```

```

    respektive L2 i V. */
V1:V,x=L1;
V2:V,x=L2;

/* Nu kan man få värden på volymerna.
   Papprets mått i dm => Volymer i dm^3. */

V1n:V1,a=2.1,b=2.97,numer; /* Tilldela variabler numeriska värden */
V2n:V2,a=2.1,b=2.97,numer;

/* Skriv ut den största volymen. */
if V1n > V2n then print("Den största volymen är ", V1n, "dm^3")
    else print("Den största volymen är ", V2n, "dm^3");

```

För att avgöra vilken volym som är störst används en konstruktion med `if`, `then` och `else`. Själva utskriften görs med funktionen `print`. Jag går inte in närmare på hur dessa funktioner används, men experimentera gärna med dem.

Övning 7.2. Konstruera några polynom och några andra funktioner som du kan derivera själv för att bekanta dig med funktionen `diff`.

Övning 7.3. Skriv ett script som tar fram största värdet till en funktion (som du själv bestämmer hur den skall vara) i ett intervall (som du också bestämmer). Jämför med funktionens graf.

7.4 Talföljder, serier och summor

Om en talföljd definieras av ett uttryck, till exempel $a:n^2$, kan man beräkna summan av med konstruktionen `sum(a,n,1,10)` där det första argumentet är uttrycket som definierar summan, det andra dummyvariabeln och de två sista gränserna för dummyvariabeln.

Om `L` är en lista kan man skriva `sum(L[i],i,1,5)` om man vill summerar de 5 första talen i listan.

8 Matematik D

8.1 Trigonometriska funktioner och ekvationer

De trigonometriska funktionerna heter naturligtvis `sin(x)`, `cos(x)` och `tan(x)`. Deras inverser heter `asin(x)`, `acos(x)` och `atan2(x)`. Alla tar och ger vinklar i radiander.

Vill man ha en funktion i vinklar i stället skulle man kunna definiera en egen funktion i stil med `sinV(v):=sin(v/360*2*%pi)`.

Maxima räknar exakt i vinklarna 0 , $\pi/6$, $\pi/4$, $\pi/3$, and $\pi/2$ och motsvarande vinklar i de tre andra kvadranterna. Vill man ha fler exakta vinklar och fler funktioner som har att göra med trigonometriska funktioner kan man ladda in ett paket för detta med `load(atrig1)`.

Det finns funktioner för att förenkla uttryck med trigonometriska funktioner, till exempel `trigsimp` som använder likheter som $\sin^2(x) + \cos^2(x) = 1$. De är dock av mindre vikt för gymnasiet kursen.

När man använder `solve` för att lösa trigonometriska ekvationer måste man vara observant på att Maxima inte lägger till hela multiplar av 2π till lösningarna. Man får också bara lösningar i en kvadrant. Det betyder att man måste jobba mycket själv för att lösa även enkla fall, som i exemplet nedan.

```
/* Script som visar hur man löser en trigonometrisk ekvation.
   Johan Wild 20080930 */

L:solve(sin(3*x)=1/2,x);

x1n:rhs(L[1])+2*pi/3*n;
x2n:%pi-rhs(L[1])+2*pi/3*n;

print("Några lösningar är:");

for i: -2 thru 5 do
  print(ev(x1n,n=i),ev(x2n,n=i));
```

I scriptet ovan exemplifieras också användningen av en konstruktion med `for` för att generera några av lösningarna. Konstruktionen `ev(x1n,n=i)` motsvarar att skriva `x1n,n=i` på en rad (vilket inte går att göra i argumentet till `print`).

Många funktioner kan förresten användas på två sätt, till exempel kan man skriva både `factor(18)` och `18,factor`. Funktionen `ev` motsvarar att beräkna värdet av ett uttryck. Jämför `x^2-x,x=3` med `ev(x^2-x,x=3)`. Oftast är den första varianten mer smidig om man skriver det på en enskild rad, men om det skall vara en del av ett annat uttryck måste man ibland använda den andra varianten.

8.2 Integraler

Funktionen för att ta fram den primitiva funktionen till ett uttryck heter `integrate` och används som i exemplet nedan.

```
f:x^2+x;
integrate(f,x);
integrate(f,x)+C;
integrate(f,x,1,5);
integrate(f,x,a,b);
```

Om man bara vill ha den primitiva funktionen till en funktion använder man `integrate` som i det första exemplet. Observera att man då inte får med någon obekant konstant. Vill man ha en sådan måste man lägga till en själv som i det andra exemplet.

Vill man ta fram en definit integral (en integral med integrationsgränser) gör man som i det tredje exemplet (med tal som integrationsgränser, vilket i detta fall resulterar i ett tal, integralens värde) eller som i det sista exemplet (med variabler som integrationsgränser, vilket resulterar i ett uttryck).

Nedan visas hur man löser följande standardproblem i Matematik D.

Två funktioner f och g ges av $f(x) = x^2 - x - 1$ och $g(x) = -x^2 + 9$. Beräkna arean mellan kurvorna $y = f(x)$ och $y = g(x)$ på intervallet $-3 < x < 4$.

```

/* Script som löser ett standardproblem i Matematik D.
   Johan Wild 20080930 */

f:x^2-x-1;
g:-x^2+9;

a:-3;
b:4;

/* Rita kurvorna för att få känsla för problemet. */

plot2d([f,g],[x,a,b]);

/* Beräkna skärningspunkterna. För att få rötterna i
   storleksordning så används funktionen sort. */

L:sort(solve(f=g,x));
x1:rhs(L[1]);
x2:rhs(L[2]);

/* En titt på grafen ger vilken funktion som är störst i olika intervall. */

A:integrate(f-g,x,a,x1)+
  integrate(g-f,x,x1,x2)+
  integrate(f-g,x,x2,b);

```

9 Matematik E

9.1 Komplexa tal

Imaginära enheten i heter `%i` i Maxima. Annars räknar Maxima med komplexa tal som vilka som helst. Det finns funktioner för realdel och imaginärdel som heter `realpart` och `imagpart`. Absolutbelopp heter (som för alla tal) `abs` och argumentet fås med funktionen `carg`.

Maxima kan konvertara mellan rektangulär form och Eulerform med funktionerna `rectform` och `polarform`.

Övning 9.1. Definiera några komplexa tal vars absolutbelopp och argument du är säker på. Testa sedan alla funktionerna.

Om en polynomekvation har komplexa rötter måste man se upp. Maxima klarar nämligen inte alltid av att skriva lösningarna som man kanske vill ha dem.

Som exempel kan ges ekvationen $x^3 - 5x^2 - 6x + 7 = 0$. Om man använder `solve` för att lösa denna kommer en lösning vara

$$x_1 = \left(-\frac{\sqrt{3}i}{2} - \frac{1}{2} \right) \left(\frac{\sqrt{7721}i}{2 \cdot 3^{\frac{3}{2}}} + \frac{331}{54} \right)^{\frac{1}{3}} + \frac{43 \left(\frac{\sqrt{3}i}{2} - \frac{1}{2} \right)}{9 \left(\frac{\sqrt{7721}i}{2 \cdot 3^{\frac{3}{2}}} + \frac{331}{54} \right)^{\frac{1}{3}}} + \frac{5}{3}$$

.

Denna lösning fås med följande kommandon i Maxima.

```

f:x^3-5*x^2-6*x+7;
L:solve(f,x);
x1:rhs(L[1]);

```

Notera att den andra faktorn i den första termen och nämnaren i den andra termen innehåller $i^{1/3}$. Det är alltså inte så lätt att se vad som är realdel och imaginärdel.

Vill man omvandla `x1` till rektangulär form eller plocka ut realdel eller imaginärdel kan man naturligtvis använda funktionerna `rectform`, `realpart` och `imagpart`. Dessa kan, som de flesta funktioner, användas på två sätt:

```
x1,rectform;  
rectform(x1);
```

I olika versioner av Maxima har det visat sig att de två varianterna inte ger samma resultat. I just version 5.19 verkar det stämma i detta exemplet men man bör se upp!

Operationerna

```
x1,realpart;  
realpart(x1);
```

ger *inte* samma resultat trots att de borde göra det. Vi får hoppas att felet rättas till i kommande versioner av Maxima.

Övning 9.2. Lös ekvationen ovan och studera resultaten.

9.2 Polynom och ekvationer

Maxima har inga speciella funktioner för polynom som rör Matematik E. Det är bara att konstatera att `factor` använder polynom över \mathbb{Z} medan `solve` löser ekvationer i \mathbb{C} .