Minimal Maxima

BY ROBERT DODIER

10 Eylül 2005

Çeviri: Tufan Şirin - Mustafa Şimşek

1 MAXİMA NEDİR?

Maxima $x+y, \sin(a+b\pi)$ ve $u \cdot v - v \cdot u$ gibi ifadeler ile çalışmak için kullanılan bir sistemdir.

Maxima bir ifadenin anlamını çok merak etmez.İfadenin anlamlı olması kullanıcının kararına bağlıdır.

Bazen, bilinmeyenlere değer atayarak ifadeleri değerlendirmek istersiniz.Bunları yapmak Maxima'yı mutlu eder.Ancak Maxima, bilinmeyenlere (özel) değerler atamayı ertelemekle de mutlu olur;bir ifadeyle bir kaç defa işlem yapabilirsiniz, ya da sadece sonra (ya da hiçbir zaman) bilinmeyenlere değer atayabilirsiniz.

Bazı örneklere bakalım.

1. Bir kürenin hacmini hesaplamak istiyorum.

```
(%i1) V:4/3 * %pi * r^3;
```

(%o1) $\frac{4\,\pi\,r^3}{3}$

```
(%i2) r:10;
```

(%o2) 10

```
(%i3) V;
```

(%o3) $\frac{4\,\pi\,r^3}{3}$

```
(%i4) ''V;
```

(%o4) $\frac{4000\,\pi}{3}$

```
(%i5) ''V,numer;
```

(%o5) 4188.790204786391

```
(%i6)
```

2 İFADELER

Maxima'da herşey bir ifadedir; matematiksel ifadeler, nesneler ve programlama yapıları. Bir ifade ya bir atomdur yada argümanlarıyla beraber bir işlemcidir.

Bir atom, bir sembol (bir isim), tırnak işaretleri içindeki bir dizi ya da bir sayı (tamsayı ya da kayan noktalı sayılar) dır. (p.c. :Kayan noktalı sayılar gerçel sayıların bilgisayar ortamındaki gösterim şekillerinden birisidir.)

Atomik olmayan bütün ifadeler $''\mathrm{op}\,(a_1, a_2, ...a_n)''$ şeklinde gösterilir. " *op* " işlemcinin -operatörün- adı $(a_1, a_2, ...a_n)$ ise argümanlarıdır. İfade farklı olarak da gösterilebilir ancak iç gösterim aynıdır. Bir ifadenin argümanları atomik ya da atomik olmayan ifadeler olabilir.

Matematiksel ifadeler "'+ , - , * , / , = ' gibi işlemciler yada sin(x), bessel_j(n,x) gibi fonksiyonlara sahiptir. Bu durumlarda işlemci bir fonksiyondur.

Maxima'da nesneler (birer) ifadedir. Bir $[a_1, \ldots, a]$ serisi "$\mathrm{list}(a_1, ...a_n)$" şeklinde ifade edilir. Bir matris

$$\mathrm{matrix}(\mathrm{list}\,(a_{1,1}, ...a_{1,n}), ...\mathrm{list}(a_{m-1}, ...a_{m,n}))$$

şeklinde bir ifadedir.

İfadeler *Programlama* ile de oluşturulabilir. Bir **block** kodu "**block**$(a_1, ...a_n)$" ifadesi **block** işlemcisi ve $(a_1, ...a_n)$ argümanları ile bir ifadedir. Bir koşul cümlesi "**if** a **then** b **elseif** c **then** d" bir **f(a,b,c,d)** ifadesidir. Bir "**for** a **in** L **do** S" döngüsü **do (a,L,S)** ifadesine benzer.

Bir Maxima fonksiyonu olan "**op**" atomic olmayan ifadelerin işlemcisidir. "'**args**' fonksiyonu atomik olmayan ifadelerin argümanlarını verir."**atom**" fonksiyonu bir ifadenin atom olup olmadığını söyler.

Daha fazla örnekler görelim.

1. Atomik ifadeler semboller, diziler ve sayılardır. Bir kaç örneği bir listenin içinde grupladım bu şekilde hepsini bir arada görebiliriz.

```
(%i39) [a, foo, foo_bar, "Hello World !", 42, 17,29];
```

(%o39) $[42, \mathrm{foo}, \mathrm{foo\_bar}, \text{Hello World !}\ , 42, 17, 29]$

```
(%i40) [a+b+c, a*b*c, foo=bar, a*b< c*d];
```

(%o40) $[c+59, 714\,c, \mathrm{foo} = \mathrm{bar}, 714 < c\,d]$

```
(%i41) L: [a, b, c, %pi, %e, 1729, 1/(a*d - b*c)];
```

(%o41) $\left[42, 17, c, \pi, \mathrm{e}, 1729, \frac{1}{42\,d - 17\,c}\right]$

```
(%i42) L2: [a, b, [c, %pi, [%e, 1729], 1/(a*d - b*c)]];
```

(%o42) $\left[42, 17, \left[c, \pi, [\mathrm{e}, 1729], \frac{1}{42\,d - 17\,c}\right]\right]$

```
(%i43) L [7];
```

(%o43) $\frac{1}{42\,d - 17\,c}$

```
(%i44) L2 [3];
```

(%o44) $\left[c, \pi, [\mathrm{e}, 1729], \frac{1}{42\,d - 17\,c}\right]$

```
(%i45) M: matrix ([%pi, 17], [29, %e]);
```

(%o45) $\begin{pmatrix} \pi & 17 \\ 29 & \mathrm{e} \end{pmatrix}$

```
(%i46) M2: matrix ([[%pi, 17], a*d - b*c], [matrix ([1, a], [b, 7]), %e]);
```

(%o46) $\begin{pmatrix} [\pi, 17] & 42\,d - 17\,c \\ \begin{pmatrix} 1 & 42 \\ 17 & 7 \end{pmatrix} & \mathrm{e} \end{pmatrix}$

```
(%i47) M [2][1];
```

(%o47) 29

```
(%i48) M2 [2][1];
```

(%o48) $\begin{pmatrix} 1 & 42 \\ 17 & 7 \end{pmatrix}$

```
(%i49) a:42;
```

(%o49) 42

```
(%i50) b:17;
```

(%o50) 17

```
(%i51) a-b; /*Her_satıra_bir_tane*/
```

(%o51) 25

```
(%i51) a:42;b:17;a-b;
```

(%o52) 42

(%o53) 17

(%o54) 25

```
(%i55) (a:42)-(b:17);
```

(%o55) 25

Yukarıda a ya ve b ye değerler atanıp a-b işlemi farklı şekillerde yapılmıştır

```
(%i56) [a,b];
```

(%o56) $[42, 17]$

```
(%i57) block([a],a:42,a^2-1600)+block([b],b:5,%pi^b);
```

(%o57) $\pi^5 + 164$

```
(%i58) (if a>1 then %pi else %e) + (if b<0 then 1/2 else 1/7);
```

(%o58) $\pi + \frac{1}{7}$

5. "**op**" işlemciyi verir, "**args**" argümanları verir, "**atom**" bir ifadenin atom olup olmadığını söyler.

```
(%i59) op (p + q);
```

(%o59) +

```
(%i60) op(p+q>p*q);
```

(%o60) >

```
(%i61) op(sin(p+q));
```

(%o61) sin

```
(%i62) op (foo (p, q));
```

(%o62) +

```
(%i63) op (foo (p, q) := p - q);
```

(%o63) :=

```
(%i64) args (p + q);
```

(%o64) $[q, p]$

```
(%i65) args (p+q>p*q);
```

(%o65) $[q + p, p\,q]$

```
(%i66) args (sin (p + q));
```

(%o66) $[q + p]$

```
(%i67) args (foo (p, q));
```

(%o67) $[p, -q]$

```
(%i68) args (foo (p, q) := p - q);
```

(%o68) $[\mathrm{foo}(p, q), p - q]$

```
(%i69) atom (p);
```

(%o69) **true**

```
(%i70) atom (p + q);
```

(%o70) **false**

```
(%i71) atom (sin (p + q));
```

(%o71) **false**

```
(%i72)
```

3 DEĞERLENDİRME

Bir sembolün değeri, o sembolle ilişkilendirilmiş bir ifadedir. Her sembol bir değere sahiptir, başka bir değer atanmamışsa bir sembol kendini (kendi değerini) değerlerlendirir.(Örn: Başka bir değer atanmadığı durumlarda x kenidisini x olarak değerlendirir.)

Sayılar ve dizinler kendilerini değerlendirir.

Atomik olmayan bir ifade yaklaşık olarak aşağıdaki gibi değerlendirilir.

1. İfadenin işlemcisinin her argümanı değerlendirilir.

2. Eğer bir işlemci çağırılabilen bir fonksiyonu ile ilişkilendirilmişse, fonksiyon çağırılır, ve verilen fonksiyonun değeri ifadenin değeri olur.

Değerlendirme bir kaç yolla (yeniden) düzenlenebilir. Bazı (yeniden) düzenlemeler değerlendirme (süreci)nin kısaltılmasını sağlar:

1. Bazı fonksiyonlar argümanlarının bazılarını ya da hepsini değerlendirmez ya da diğer durumda argümanların değerlendirmesini (yeniden) düzenler.

2. Tek tırnak işareti “ ’ “ değerlendirmeyi önler.
(a) **’a**, ’a’ olarak değerlendirilir. ’a’ nın başka herhangi bir değeri yoksayılır.

a) ’a, a yı değerlendirir. a’nın herhangi bir başka değeri yoksayılır.

b) $'f(a_1, ..., a_n)$, $f(\boldsymbol{ev}(a_1, ..., a_n))$ yı değerlendirir. Yani, argümanlar değerlendirilir ama f çağırılmaz.

c) ’(.....) , (.....) içindeki ifadenin değerlendirilmesini önler.

Bazı (yeniden) düzenlemer daha fazla değerlendirme (işlemlerine) neden olur.

1. Çift tırnak (“a) (a) ifadesi (program tarafından) incelenirken ekstra değerlendirmeler uygular.
2. **ev**(a); a değişkeninin ev(a) ile her defasında tekrar değerlendirilmeye tabi tutulmasına neden olur.
3. “$\boldsymbol{apply(f, [a_1, ..., a_n])}$’ deyimi $a_1, \ldots, a_n$, f normalde tınak içine alsa bile argümanlarının değerlendirilmesine neden olur.
4. **define** , ’:= ’, gibi fonksiyon tanım(lama) işlevi görür, ancak, ’:=’ fonksiyonun bir kısmını tırnak içine alırken **define** fonksiyonu değerlendirir.

Bazı ifadelerin nasıl değerlendirildiğine bakalım.

1. Herhangi bir başka değer atanmadığı durumlarda semboller kendi değerlerini verirler.

```
(%i72) block(a:1,b:2,e:5);
```

(%o72) 5

```
(%i73) [a,b,c,d,e];
```

(%o73) $[1, 2, c, d, 5]$

```
(%i74)
```

2. İşlemcilerin argümanları sırasıyla değerlendirilir. (Değerlendirilme, tek yönlü ya da başka bir şekilde önlenmemişse.)

```
(%i74) block (x: %pi, y: %e);
```

(%o74) e

```
(%i75) sin (x + y);
```

(%o75) $-\sin(\mathrm{e})$

```
(%i76) x>y;
```

(%o76) $\pi > \mathrm{e}$

```
(%i77) x!;
```

(%o77) $\pi!$

```
(%i78)
```

3. Eğer bir işlemci çağırılabilen bir fonksiyona karşılık geliyorsa (engellenmediği durumlarda) fonksiyon çağırılır. Aksi halde değerlendirme , aynı işlemci ile başka bir ifadeyi sonuç verir.

```
(%i78) foo (p, q) := p - q;
```

(%o78) $\mathrm{foo}(p, q) := p - q$

```
(%i79) p: %phi;
```

(%o79) %phi

```
(%i80) foo (p, q);
```

(%o80) %phi $- q$

```
(%i81) bar (p, q);
```

(%o81) $\mathrm{bar}(\%\mathrm{phi}, q)$

```
(%i82)
```

4. Bazı fonksiyonlar kendi argümanlarını parantez içine alırlar. Örnek: **save**, **:=**, **kill**.

```
(%i82) kill(all)$
(%i1) block (a: 1, b: %pi, c: x + y);
```

(%o1) $y + x$

```
(%i2) [a, b, c];
```

(%o2) $[1, \pi, y + x]$

```
(%i3) save ("tmp.save", a, b, c);
```

(%o3) /home/tufan/tmp.save

```
(%i4) f (a) := a^b;
```

(%o4) $f(a) := a^b$

```
(%i5) f(a) ;
```

(%o5) 1

```
(%i6) f (7);
```

(%o6) 7^{π}

```
(%i7) kill (a, b, c);
```

(%o7) **done**

```
(%i8) [a, b, c];
```

(%o8) $[a, b, c]$

```
(%i9)
```

5. Tek tırnak (işareti) değerlendirmeyi önler; sırasıyla gerçekleşse bile.

```
(%i9) kill(all)$
(%i1) foo (x, y) := y - x;
```

(%o1) $\mathrm{foo}(x, y) := y - x$

```
(%i2) block (a: %e, b: 17);
```

(%o2) 17

```
(%i3) foo (a, b);
```

(%o3) $17 - \mathrm{e}$

```
(%i4) foo ('a, 'b);
```

(%o4) $b - a$

```
(%i5) 'foo (a, b);
```

(%o5) $\mathrm{foo}(\mathrm{e}, 17)$

```
(%i6) '(foo (a, b));
```

(%o6) $\mathrm{foo}(a, b)$

6. İki tek tırnak (") (tırnak-tırnak) ifade (maxima tarafından okunurken) ekstra değerlendirmelere neden olur.

```
(%i7) kill(all)$
(%i1) diff (sin (x), x);
```

(%o1) $\cos(x)$

```
(%i2) 'foo (x) := diff (sin (x), x);
```

(%o2) $\mathrm{foo}(x) := \mathrm{diff}(\sin(x), x)$

```
(%i3) foo (x) := ''(diff (sin (x), x));
```

(%o3) $\mathrm{foo}(x) := \cos(x)$

```
(%i4) foo ('a, 'b);
```

Too many `arguments supplied to` $\mathrm{foo}(x)$`; found:` $[a, 'b]$
`-- an error. To debug this try: debugmode(true);`

```
(%i5) 'foo (a, b);
```

(%o5) $\mathrm{foo}(a, b)$

7. **ev** işlemcisi değerlendirme yaptığı her zaman ekstra değerlendirme (işlemlerine neden olur. Bu iki tek tırnakın etkilerine terstir.)

```
(%i6) block (xx: yy, yy: zz);
```

(%o6) zz

```
(%i7) [xx, yy];
```

(%o7) $[\mathrm{yy}, \mathrm{zz}]$

```
(%i8) foo (x) := ''x;
```

(%o8) $\mathrm{foo}(x) := x$

```
(%i9) foo (xx);
```

(%o9) yy

```
(%i10) bar (x) := ev (x);
```

(%o10) $\mathrm{bar}(x) := \mathrm{ev}(x)$

```
(%i11) bar (xx);
```

(%o11) zz

8. **apply** normalde tırnak işareti ile işsaretlenmemiş olsalar bile argümanların değerlendirilmesine neden olur.

```
(%i12) kill(all);
```

(%o0) **done**

```
(%i1) block (a: aa, b: bb, c: cc);
```

(%o1) cc

```
(%i2) block (aa: 11, bb: 22, cc: 33);
```

(%o2) 33

```
(%i3) [a, b, c, aa, bb, cc];
```

(%o3) $[\mathrm{aa}, \mathrm{bb}, \mathrm{cc}, 11, 22, 33]$

```
(%i4) apply (kill, [a, b, c]);
```

(%o4) **done**

```
(%i5) [a, b, c, aa, bb, cc];
```

(%o5) $[\mathrm{aa}, \mathrm{bb}, \mathrm{cc}, \mathrm{aa}, \mathrm{bb}, \mathrm{cc}]$

```
(%i6) kill (a, b, c);
```

(%o6) **done**

```
(%i7) [a, b, c, aa, bb, cc];
```

(%o7) $[a, b, c, \mathrm{aa}, \mathrm{bb}, \mathrm{cc}]$

9. **define** tanımlamış fonksiyonu değerlendirir.

```
(%i8) integrate (sin (a*x), x, 0, %pi);
```

(%o8) $\dfrac{1}{a} - \dfrac{\cos(\pi\, a)}{a}$

```
(%i9) foo (x) := integrate (sin (a*x), x, 0, %pi);
```

(%o9) $\mathrm{foo}(x) := \mathrm{integrate}(\sin(a\, x), x, 0, \pi)$

```
(%i10) define (foo (x), integrate (sin (a*x), x, 0, %pi));
```

(%o10) $\mathrm{foo}(x) := \dfrac{1}{a} - \dfrac{\cos(\pi\, a)}{a}$

4 SADELEŞTİRME

Değerlendirmeyi yaptıktan sonra Maxima daha 'basit (sade)' bir ifade bulmaya çalışır. Maxima sadeleştirmenin geleneksel kavramlarını simgeleyen birkaç farklı kuralı uygular. Örneğin, '1+1', '2' olarak , 'x+x' '2x' olarak ve sin(%pi) '0' olarak sadeleştirilir.
Bununla birlikte, birçok çok bilinen özdeşlikler otomatik olarak uygulanmaz. Örneğin, trigonometrik fonksiyonların 'iki-açı' (double-angle) formülleri ya da
$a/b + c/b \longrightarrow (a + c)/b$ gibi ifadelerin tekrar düzenlemeleri. Özdeşlikleri uygulayan birkaç fonksiyon vardır. Açık bir şekilde önlenmediyse, sadeleştirme her zaman yapılır. Bir ifade değerlendirilmese bile sadeleştirme yapılır. **tellsimpafter** kullanıcı tarafından belirlenen sadeleştirme kurallarını yerleştirir.
Sadeleştirme ile ilgili bazı örnekleri görelim.

1. Tırnak işareti değerlendirmeyi önlediği halde sadeleştirmeyi önlemez. **simp** 'false' olarak seçilmişse değerlendirme yapıldığı halde sadeleştirme önlenir.

```
(%i11) '[1 + 1, x + x, x * x, sin (%pi)];
```

(%o11) $[2, 2\,x, x^2, 0]$

```
(%i12) simp: false$
```

```
(%i13) block ([x: 1], x + x);
```

(%o13) $1 + 1$

2. Bazı özdeşlikler otomatik olarak uygulanmaz. **expand**, **ratsimp**, **trigexpand**,**demoivre** gibi fonksiyonlar özdeşlikleri uygularlar.

```
(%i14) simp: true$
```

```
(%i15) (a + b)^2;
```

(%o15) $(b + a)^2$

```
(%i16) expand (%);
```

(%o16) $b^2 + 2\,a\,b + a^2$

```
(%i17) a/b + c/b;
```

(%o17) $\frac{c}{b} + \frac{a}{b}$

```
(%i18) ratsimp (%);
```

(%o18) $\frac{c + a}{b}$

```
(%i19) sin (2*x);
```

(%o19) $\sin(2\,x)$

```
(%i20) trigexpand (%);
```

(%o20) $2\cos(x)\sin(x)$

```
(%i21) a * exp (b * %i);
```

(%o21) $a\,\mathrm{e}^{\mathrm{i}\,b}$

```
(%i22) demoivre (%);
```

(%o22) $a\,(\mathrm{i}\sin(b) + \cos(b))$

5 apply, map ve lambda

1. **apply** bir ifadeyi oluşturur ve değerlendirir. İfadenin argümanları her zaman değerlendirilir

```
(%i23) apply (sin, [x * %pi]);
```

(%o23) $\sin(\pi\, x)$

```
(%i24) L: [a, b, c, x, y, z];
```

(%o24) $[a, b, c, x, y, z]$

```
(%i25) apply ("+", L);
```

(%o25) $z + y + x + c + b + a$

2. **map** Bir argümanlar dizininde bulunan herbir parça için bir ifade oluşturur ve değerlendirir. İfadenin argümanları herzaman değerlendirilir. Sonuç bir listedir.

```
(%i26) kill (all)$
(%i1) map (foo, [x, y, z]);
```

(%o1) $[\mathrm{foo}(x), \mathrm{foo}(y), \mathrm{foo}(z)]$

```
(%i2) map ("+", [1, 2, 3], [a, b, c]);
```

(%o2) $[a + 1, b + 2, c + 3]$

```
(%i3) map (atom, [a, b, c, a + b, a + b + c]);
```

(%o3) [**true**, **true**, **true**, **false**, **false**]

3. **lambda** bir lambda ifadesi oluşturur. (Bir başka deyişle bir isimsiz fonksiyon). Bazı durumlarda lambda ifadesi normal isimsiz bir fonksiyon gibi kullanılabilir. **lambda**, fonksiyonu değerlendirmez.

```
(%i32) kill(all)$
(%i1) diff (sin (x), x);
```

(%o1) $\cos(x)$

```
(%i2) 'foo (x) := diff (sin (x), x);
```

(%o2) $\mathrm{foo}(x) := \mathrm{diff}(\sin(x), x)$

```
(%i3) foo (x) := ''(diff (sin (x), x));
```

(%o3) $\mathrm{foo}(x) := \cos(x)$

```
(%i4) foo ('a, 'b);
```

Too many `arguments supplied to` $\mathrm{foo}(x)$`; found:` $[a, 'b]$
`-- an error. To debug this try: debugmode(true);`

```
(%i5) 'foo (a, b);
```

(%o5) $\mathrm{foo}(a, b)$

```
(%i11) '[1 + 1, x + x, x * x, sin (%pi)];
```

(%o11) $[2, 2\,x, x^2, 0]$

```
(%i12) simp: false$
(%i13) block ([x: 1], x + x);
```

(%o13) $1+1$

6 Yerleşik (Built-in) Nesne Türleri

Bir nesne bir ifade gibi gösterilir. Diğer ifadeler gibi, bir nesne bir işlemci ve argümanlarından oluşur.
En önemli yerleşik nesne türleri listeler, matrisler ve kümelerdir.

6.1 Listeler

1. Bir liste [a,b,c] şeklinde belirtilir.
2. Eğer L bir liste ise L[i], L listesinin i. elemanıdır. L[1] birinci elemandır.
3. **map** *(f,L) f*' i L 'nin her elemanına uygular.
4. **apply** *("+", L)* ***L*** elemanları toplamıdır.
5. **for** x **in** L **do** *ifade* .... L'nin her elemanı için *ifade*yi değerlendirir.
6. **length** (L) L deki eleman sayısıdır.

6.2 Matrisler

1. Bir matris şu şekilde tanımlanır. **matrix**$(L_1, ...L_n)$. Burada $L_1, ..., L_n$ matrisin satırlarını gösteren listelerdir.
2. Eğer M bir matris ise L[i,j] ya da M[i][j] onun [i,j] nci elemanıdır. M[1,1] üst sol köşedeki elemanıdır.
3. " . " işlemcisi sırabağımlı olmayan (noncomutative) çarpmayı ifade eder. L nin bir liste M ve N' nin birer matris olduğu durumlarda M.L , L.M, ve M.N çarpımları sırabağımlı olmayan çarpımlardır.
4. **transpose** (M) M'nin transpozesidir.
5. **eigenvalues** (M) M'nin özdeğerlerini verir.
6. **eigenvectors** (M) M'nin özvektörlerini verir.
7. **length** (M) M deki satır sayısıdır.
8. **length** (**transpose** (M)) M'nin sütun sayısını verir.

6.3 Kümeler

1. Maxima açık bir şekilde (belirtik-açık) tanımlanmış (explicitly-defined) sonlu kümeleri anlar. Kümeler listeler gibi değildir. Birisini bir diğerine dönüştürmek için '*belirtik bir dönüşüm*' gerekir.
2. Bir küme şu şekilde gösterilir: **set** *(a,b,c,....)* a,b,c kümenin elemanlarıdır.
3. **union** *(A,B)* A ve B kümelerinin bileşimidir.
4. **intersection** *(A,B)* A ve B kümelerinin kesişimidir.
5. **cardinality** (A) A kümesinin eleman sayısıdır.
6. **transpose** (M) M'nin transpozesidir.

7. **eigenvalues** (M) M'nin özdeğerlerini verir.
8. **eigenvectors** (M) M'nin özvektörlerini verir.
9. **length** (M) M deki satır sayısıdır.
10. **length** (**transpose** (M)) M'nin sütun sayısını verir.

7 NASIL YAPILIR

7.1 Bir Fonksiyon Tanımlama

1. ":=" işlemcisi bir fonksiyon tanımlar. Bu örnekte, **diff** (komutu) fonksiyon her çağırıldığında tekrar değerlendirilir. Argüman x için yerine koyulur ve sonucu veren ifade değerlendirilir. Argüman bir sembolden başka bir şey ise bu hataya neden olur: **foo**(1) için Maxima ***diff***$(\sin(1)^2, 1)$ i hesaplamayı dener.

```
(%i5) foo (x) := diff (sin(x)^2, x);
```

(%o5) $\mathrm{foo}(x) := \mathrm{diff}(\sin(x)^2, x)$

```
(%i6) foo (u);
```

(%o6) $2\cos(u)\sin(u)$

```
(%i7) foo (1);

diff: second argument must be a variable; found 1
#0: foo(x=1)
 -- an error. To debug this try: debugmode(true);
(%i8)
```

2. **define** bir fonksiyon tanımlar. Fonksiyonu değerlendirir.
Bu örnekte **diff** sadece bir defa değerlendirilir (fonksiyon tanımlandığında). Şimdi **foo** (1) tamamdır!.

```
(%i7) define (foo (x), diff (sin(x)^2, x));
```

(%o8) $\mathrm{foo}(x) := 2\cos(x)\sin(x)$

```
(%i9) foo (u);
```

(%o9) $2\cos(u)\sin(u)$

```
(%i10) foo (1);
```

(%o10) $2\cos(1)\sin(1)$

7.2 Bir denklemi Çözmek...

```
(%i8) eq_1: a * x + b * y + z = %pi;
```

(%o8) $z+b\,y+a\,x=\pi$

```
(%i9) eq_2: z - 5*y + x = 0;
```

(%o9) $z-5\,y+x=0$

```
(%i10) s: solve ([eq_1, eq_2], [x, z]);
```

(%o10) $\left[\left[x=-\frac{(b+5)\,y-\pi}{a-1},z=\frac{(b+5\,a)\,y-\pi}{a-1}\right]\right]$

```
(%i11) length (s);
```

(%o11) 1

```
(%i12) [subst (s[1], eq_1), subst (s[1], eq_2)];
```

(%o12) $\left[\frac{(b+5\,a)\,y-\pi}{a-1}-\frac{a\,((b+5)\,y-\pi)}{a-1}+b\,y=\pi,\frac{(b+5\,a)\,y-\pi}{a-1}-\frac{(b+5)\,y-\pi}{a-1}-5\,y=0\right]$

```
(%i13) ratsimp (%);
```

(%o13) $[\pi=\pi,0=0]$

7.3 İntegral ve Diferansiyel

integrate belirli ve belirsiz integralleri hesaplar.

```
(%i14) integrate (1/(1 + x), x, 0, 1);
```

(%o14) $\log(2)$

```
(%i15) integrate (exp(-u) * sin(u), u, 0, inf);
```

(%o15) $\frac{1}{2}$

```
(%i16) assume (a>0);
```

(%o16) $[a>0]$

```
(%i17) integrate (1/(1 + x), x, 0, a);
```

(%o17) $\log(a+1)$

```
(%i18) integrate (exp(-a*u) * sin(a*u), u, 0, inf);
```

(%o18) $\frac{1}{2\,a}$

```
(%i19) integrate (exp (sin (t)), t, 0, %pi);
```

(%o19) $\int_0^{\pi}\mathrm{e}^{\sin(t)}\ \mathrm{d}\ t$

```
(%i20) 'integrate (exp(-u) * sin(u), u, 0, inf);
```

(%o20) $\int_0^{\infty}\mathrm{e}^{-u}\sin(u)\ \mathrm{d}\ u$

diff diferansiyel hesaplari yapar.

```
(%i21) diff (sin (y*x));
```

(%o21) $x\cos(x\,y)\,\mathrm{del}(y) + y\cos(x\,y)\,\mathrm{del}(x)$

```
(%i22) diff (sin (y*x), x);
```

(%o22) $y\cos(x\,y)$

```
(%i23) diff (sin (y*x), y);
```

(%o23) $x\cos(x\,y)$

```
(%i24) diff (sin (y*x), x, 2);
```

(%o24) $-y^2\sin(x\,y)$

```
(%i25) 'diff (sin (y*x), x, 2);
```

(%o25) $\frac{d^2}{d\,x^2}\sin(x\,y)$

7.4 Çizim Yapmak...

plot2d iki boyutlu çizim yapar.

```
(%i1) plot2d (exp(-u) * sin(u), [u, 0, 2*%pi]);
```

(%o1) /home/tufan/maxout.gnuplot_pipes

```
(%i2) plot2d ([exp(-u), exp(-u) * sin(u)], [u, 0, 2*%pi]);
```

(%o2) /home/tufan/maxout.gnuplot_pipes

```
(%i3) xx: makelist (i/2.5, i, 1, 10);
```

(%o3) $[0.4, 0.8, 1.2, 1.6, 2.0, 2.4, 2.8, 3.2, 3.6, 4.0]$

```
(%i4) yy: map (lambda ([x], exp(-x) * sin(x)), xx);
```

(%o4) [0.26103492114345, 0.32232886922706, 0.28072477796926, 0.20181042993345, 0.1230600\
2480577, 0.061276637261957, 0.020370650389686, −0.0023794587414574, −0.012091305769841,
−0.013861321214152]

```
(%i5) plot2d ([discrete, xx, yy]);
```

(%o5) /home/tufan/maxout.gnuplot_pipes

```
(%i6) plot2d ([discrete, xx, yy],[gnuplot_curve_styles, "with points"]);
```

(%o6) /home/tufan/maxout.gnuplot_pipes

```
(%i7)
```

7.5 Bir dosyayı Kaydetmek ve Yüklemek

save ifadeleri bir dosyaya yazar- kaydeder.

```
(%i7) a: foo - bar;
```

(%o7) foo – bar

```
(%i8) b: foo^2 * bar;
```

(%o8) bar foo^2

```
(%i9) save ("my.session", a, b);
```

(%o9) /home/tufan/my.session

```
(%i10) save ("my.session", all);
```

(%o10) /home/tufan/my.session

```
(%i11) load ("my.session");
```

assignment: cannot assign to ef˙data(exponent, reduction, primitive, cardinality, order, factors˙of˙order)
-- an error. To debug this try: debugmode(true);

```
(%i12) a
```

(%o12) foo – bar

```
(%i13) b
```

(%o13) bar foo^2

```
(%i14)
```

8 Maxima'yla Programlama

Bütün Maxima sembollerini içeren bir tane isim uzayı vardır.Başka bir isim uzayı oluşturmanın yolu yoktur.

Eğer lokal değişkenler, deklare edilmemişlerse, global değişkenlerdir. Fonksiyonlar, lambda ifadeler ve 'block'lar lokal değişkenlere sahip olabilirler.

Bir değişkenin değeri en son atanan değerdir; (bu atama işlemi) ya direkt- doğrudan (explicit) bir atama ya da bir fonksiyon, bir lambda ifadesi ve ya bir 'block' içindeki lokal bir değişkene bir değer (atanması yoluyla yapılmış olabilir). Bu dinamik içerik olarak bilinir.

Eğer bir değişken bir fonksiyon, bir lambda ifade ya da bir 'block' içinde lokal- yerel bir değişken ise değeri lokaldir fakat diğer özellikleri (**declare** ile oluşturulmuş olması gibi) globaldir. **local** fonksiyonu, bir değişkeni bütün özellikleriyle göre lokal yapar.

Varsayılan olarak bir fonksiyon tanımı, bir fonksiyonun ya da bir lambda ifadesinin ya da bir 'block' ifadesinin içinde olsa bile, globaldir. local(f), f (x) := . . . lokal bir fonksiyon tanımı oluşturur.

trace(*foo*) komutu *foo* girildiğinde ya da çıkarıldığında Maxima'nın bir mesaj vermesine neden olur.

Maxima'da bazı programlama örnekleri görelim.

1. Lokal olarak deklare edilmemişlerse bütün değişkenler globaldir. Fonksiyonlar, lambda ifadeleri ve 'block'lar lokal değişkenler içerebilir.

```
(%i14) (x: 42, y: 1729, z: foo*bar);
```

(%o14) bar foo

```
(%i15) f (x, y) := x*y*z;
```

(%o15) $f(x,y) := x\,y\,z$

```
(%i16) f (aa, bb);
```

(%o16) aa bar bb foo

```
(%i17) lambda ([x, z], (x - z)/y);
```

(%o17) $\lambda\left([x,z],\frac{x-z}{y}\right)$

```
(%i18) apply (%, [uu, vv]);
```

(%o18) $\frac{\mathrm{uu}-\mathrm{vv}}{1729}$

```
(%i19) block ([y, z], y: 65536, [x, y, z]);
```

(%o19) $[42, 65536, z]$

```
(%i20)
```

2. Bir değişkenin değeri en son atanmış olan neyse o değerdir, (bu) ister açık bir değer atanmasıdır ya da lokal bir değişkene değer atanmasıdır.

```
(%i20) foo (y) := x - y;
```

(%o20) $\mathrm{foo}(y) := x - y$

```
(%i21) x: 1729;
```

(%o21) 1729

```
(%i22) foo (%pi);
```

(%o22) $1729 - \pi$

```
(%i23) bar (x) := foo (%e);
```

(%o23) $\mathrm{bar}(x) := \mathrm{foo}(\mathrm{e})$

```
(%i24) bar (42);
```

(%o24) $42 - \mathrm{e}$

```
(%i25)
```

9 Lisp ve Maxima

:lisp *expr* yapısı, Lisp yorumlayıcısına *expr*' i değerlendirmesini söyler. Bu yapı, girdi (yapılmak) istendiğinde ve **load** ile işlenemeyen ancak **batch** ile işlenen dosyalarda görülür.

Maxima'nın *foo* sembolü Lisp'in $foo sembolüne karşılık gelir ve Lisp' teki foo Maxima'nın ?**foo** sembolüne karşılık gelir.

:**lisp** (**defun** $foo (a) (. . .)) argümanlarını değerlendiren bir 'Lisp foo fonksiyonu' tanımlar. Maxima'da bu fonksiyon **foo**(a) dır.

:**lisp** (**defmspec** $foo (e) (. . .)) argümanlarını tırnak içine alan bir Lisp **foo** fonksiyonu tanımlar. Bu fonksiyon Maxima'da **foo** (a) olarak adlandırılır. The arguments of $foo are (cdr e), and (caar e) is always $foo itself.

Lisp'in (mfuncall 0 $foo a 1 . . . a n) yapısı Maxima'da tanımlanan **foo** fonksiyonunu çağırır.

Maxima'da Lisp'e ulaşalım ve tam tersini de yapalım.

1. **:lisp** *expr* yapısı Lisp yorumlayıcısına *expr*' i değerlendirmesini söyler.

masıdır ya da lokal bir değişkene değer atanmasıdır.

```
(%i25) (aa + bb)^2;
```

(%o25) $(\mathrm{bb}+\mathrm{aa})^2$

```
(%i26) :lisp $%
```

```
((MEXPT SIMP) ((MPLUS SIMP) $AA $BB) 2)
```

```
(%i26) kill(all)$
(%i1) :lisp (defun $foo (a b) `((mplus) ((mtimes) ,a ,b) $%pi))
```

```
$FOO
```

```
(%i1) (p: x + y, q: x - y);
```

(%o1) $x-y$

```
(%i2) foo (p, q);
```

(%o2) $(x-y)\,(y+x)+\pi$

```
(%i3)
```