Resumen HIM

Alex Martínez Ascensión

13 de enero de 2019

1. Operaciones

	Máxima	Scilab
Suma	+	+
Resta	-	-
Multiplicación	*	*
División	/	/
Potencia	** o ^	** o ^
Factorial	`factorial()`	`factorial()`
Logaritmo natural	`log()`	`log()`
Logaritmo decimal	-	`log10()`
Exponencial	`exp()`	`exp()`
Raíz cuadrada	`sqrt()`	`sqrt()`
y	`and`	&
o	`or`	|
not	`not`	~
igualdad	=	==
desigualdad	#	<>
menor	<	<
menor o igual	<=	<=
mayor	>	>
mayor o igual	>=	>=
Ejecución	is(a=b)	a==b
Producto matricial	.	*
Producto elemental	*	.*
División matricial		/
División elemental	/	./
Potencia matricial	^^	^
Potencia elemental	^	.^
Función Γ	`gamma(x)`	`gamma(x)`
Función β	`beta(x)`	`beta(x)`
Comentarios	`/* ... */`	`//`

Nota: en Máxima, si dos expresiones numéricas son diferentes y el valor es el mismo, `is(a=b)` dará fallo. Hay que escribir `is(equal(a,b))`

2. Respuestas

	Máxima	Scilab
Respuesta anterior	`%`	`ans`
Respuesta determinada	`%oX`	

3. Variables

	Máxima	Scilab
π	`%pi`	`%pi`
e	`%e`	`%e`
i	`%i`	`%i`
Verdadero	`True` o `true`	`%T` o `%t`
False	`False` o `false`	`%F` o `%f`
$+\infty$	`inf`	

4. Estructuras de datos

4.1. Arrays

	Máxima	Scilab
Inicialización	`array(nombre, N_FILAS-1, N_COLS-1)`	`nombre = cell(N_FILAS, N_COLS)|`
Asignación	`nombre[I,J]: X`	`nombre{I,J} = X`
Impresión	`listarray(nombre)`	`nombre`

4.2. Estructuras

	Máxima	Scilab
Inicialización		`nombre = struct('a1','b1','a2','b2',...,'an','bn')`
Asignación		`nombre.ak = 'bk'`
Impresión		`nombre` ->a1: b1; a2: b2; ... , an:bn `nombre.a1`

4.3. Listas

Las listas en Scilab se declaran usando los comandos `tlist` o `mlist` si son listas con tipo u orientadas a matrices, o `list` si son ordinarias. El primer elemento de la lista es el índice, el segundo el nombre de las columnas, y las siguientes, las filas restantes. Por ejemplo:

```
elementos=tlist(['Lista_elementos', 'Campos', 'Plutonio', 'Radio'],
['Masa', 'Radio', 'P_fusion', 'Densidad'],
[244, 175, 912.5, 19816],
[266.02, 215, 973, 5000])
```

	Máxima	Scilab
Inicialización	`nombre:['A', 'B', ['C1', 'C2']]`	`nombre = tlist()`
Impresión	`part(nombre, posición)`	`nombre` `nombre.Campos(1)`

4.4. Matrices

	Máxima	Scilab
Inicialización	`nombre: matrix([1,2,3],[4,5,6],[7,8,9])`	`nombre = [1 2 3; 4 5 6; 7 8 9]`
Asignación de elemento	`nombre[i,j] : X`	`nombre(i,j) = X`
Asignación de fila	-	`nombre(i,:) = [a b c]`
Asignación de columna	-	`nombre(:,) = [a b c]`
Impresión	`nombre` o `nombre[i,j]` o `submatrix(i1, i2, ..., nombre, j1, j2, ...)`	`nombre` o `nombre(i,j)` o `nombre(2,:)`
Agregación de filas/cols	`addrow(r1,r2,...)` o `addcol(c1, c2, ...)`	
Tamaño de matriz	`matrix_size(nombre)`	`size(matrix)`
Matriz inversa	`invert(nombre)` o `nombre^^-1`	`inv(A)`
Determinante	`determinant(nombre)`	`det(nombre)`
Transpuesta	`transpose(nombre)`	`nombre'`
Rango	`rank(nombre)`	`rank(nombre)`
Identidad	`ident(n)`	`eye(nombre)`
Matriz nula	`zeromatrix(i,j)`	`zeros(nombre)`
Autovalores	`eigenvalues(nombre)`	
Autovectores	`eigenvectors(nombre)`	
Devolver índices		`find(vector == x)`
Vector a lista	`list_matrix_entries(matrix)`	

5. Operaciones trigonométricas

Son operaciones iguales en máxima y scilab. La lista de operaciones comunes es `acos`, `acosh`, `acot`, `acoth`, `acsc`, `acsch`, `asec`, `asech`, `asin`, `asinh`, `atan`, `atanh`, `cos`, `cosh`, `cot`, `coth`, `csc`, `csch`, `sec`, `sech`, `sin`, `sinh`, `tan` y `tanh`.

Para operar y simplificar expresiones trigonométricas hay varias funciones:

- `trigreduce()` simplifica una expresión a sumas de senos y cosenos.
- `trigexpand()` transforma expresiones con sumas de ángulos a expresiones simplificadas.
- `trigsimp()` transforma expresiones con `tan`, `sec`, etc. a sus versiones con `sen` y `cos`.
- `trigrat()` expande funciones trigonométricas a otras con combinaciones lineales de sus argumentos.
- `exponentialize()` transforma funciones trigonométricas en forma exponencial.

6. Operaciones polinomiales

En Máxima pueden crearse polinomios del estilo `P1: x^2-2*x+1` y `P2: a*x^3+b*x`, y se pueden hacer relaciones con ellos, del tipo `P1+P2` o `3*P1/P2`.

Además de las operaciones matemáticas usuales, existen funciones que operan con polinomios.

	Máxima	Scilab
Factorización numérica y polinomial	`factor(P)`	
Agrupación de variables	`factorout(P)`	
Expansión polinómica	`expand(P)`	
Cálculo de ceros de polinomio	`allroots(P)`	
MCD Polinómico	`gcd(P)`	
División polinómica	`divide(P1, P2)`	
Simplificación polinómica racional	`partfrac(P, var)`	

7. Otras operaciones

	Máxima	Scilab

Sustitución variables	`X:matrix([a,b,c])`
	`a:1$b:2$c:3;`
	`X,num;`
		`a=1;b=2;c=3`
		`X=[a b c]`
		`X`

8. Resolución de ecuaciones

Las ecuaciones pueden resolverse simbólicamente con Máxima con diferentes funciones dependiendo del grado de complejidad de la ecuación o sistema de ecuaciones.

- Ecuación simple: `solve(eq, var)`
- Ecuación simple con varias incógnitas: `solve(eq, [v1, v2, ...])`
- Ecuación trascendente (u otro tipo, más general): `find_root(eq, var, intervalo_0, intervalo_f)`
- Sistema lineal de ecuaciones con varias incógnitas: `linsolve([eq1, eq2, ...],[v1,v2,...])`
- Sistema algebráico de ecuaciones: `algsys([eq1, eq2, ...], [v1, v2, ...])`
- Resolución por método de Newton: `load(mnewton);mnewton([eq1, eq2, ...], [v1, v2, ...], [v1_0, v2_0, ...])`. El último elemento es un vector inicial que usa `mnewton` para hallar la solución. Si el sistema tiene más de una solución, el punto de inicio aproximará una solución u otra.
- `ev(f, x, y)` te evalua una expresión poniéndole los parametros `x=a` e `y=b`, siendo por ejemplo `x` e `y` las variables a evaluar.
- Si tenemos una expresión como, por ejemplo, `exp: [x+y = 2y + a*b]`, generada por una función `solve`, o algo por el estilo, ejecutamos `rhs(part(exp,1))` para quedarnos con el primer elemento de la lista, y luego con el lado derecho de la ecuación.
- Resolución de inecuaciones: `load(fourier_elim)$ fourier_elim([expr > val], [x])`

En Scilab se puede hacer una resolución numérica de ecuaciones con `fsolve`. Suponemos el sistema de ecuaciones $y - x^2 + 1 = 0$; $y - e^x * \tan(x) = 0$. Entonces, definimos una función que toma x e y y almacena los resultados del lado izquierdo de ambas ecuaciones. Luego usamos esa función, y un vector inicial para hallar la solución.

```
function res=f(xy); res(1) = xy(2)- xy(1)^2+1; res(2) = xy(2) - exp(xy(1))*tan(xy(1)); endfunction
xy0 = [0;1];
xsol = fsolve(xy0, f)
```

El resultado es `xsol = [-0.748;-0.439]`, pero no sabemos si es la única.

9. Representaciones gráficas

9.1. Plot 2D simple

Máxima

De manera dicreta, se crean los puntos con una matriz, y luego se aplica la función.

```
x: [0,0.5,1,1.5,2,2.5,3,3,3.5]$
```

```
y: 2*x+3$
plot2d([discrete,x,y]);
```

Sin embargo, también se puede hacer de manera continua, especificando los límites y la variable.

```
plot2d(2*x+3, [x,0,3.5]);
```

Scilab

Se crea un array de puntos en x, y luego se aplica la función.

```
x = 0:0.01:10;
y = 2*x+3;
plot(x,y)
```

9.2. Plot 2D parametrizado

En Máxima podemos hacer un plot de una función parametrizada para `x(t)` y `y(t)`. La función es:

```
plot2d([parametric, x(t), y(t), [t, t_0, t_f], [nticks, X]])
```

El parámetro `nticks` puede omitirse y, en ese caso, te dibuja la función de manera "continua".

9.3. Representación de varias funciones

Tanto en Máxima como en Scilab se pueden representar funciones con una sola función. Simplemente, hay que hacer una lista de funciones individuales.

Máxima

```
plot2d([2*x+1,
       [discrete, [-1,-0.5,0,1], [0,1,0,1]],
       [parametric, sin(t), cos(t),
       [t,0,%pi*2]],
       [parametric, sin(t), cos(t^2),
       [t,0,2*%pi]]],
       [x, -1,1]);
```

```
plot2d([cos(x), x*cos(x), x^2],
[x, 0, 10]);
```

Scilab

¡El vector tiene que ser vector columna!

```
x=[0:0.01:10]';
plot(x, [cos(x), x.*cos(x), x.^2])
```

9.4. Gráficas en 3D

Máxima

Para crear una gráfica en 3D primero creamos la función `z(x,y)`, y luego hacemos el plot.

```
z(x,y) := exp(x)*sin(y)^2$
plot3d(z, [x,0,1], [y,0,1])
```

Scilab

```
x=[0:0.01:10]';
y=[0:0.01:10]';
z=exp(x)*sin(y')^2;
plot3d(x,y,z)
```

Aquí es muy importante hacer que x e y sean matrices columna. Si no, no sale el gráfico. A veces es más fácil definir la función con la función `deff` y nos ahorramos ese paso. En este caso usamos `fplot3d()` para plotear.

```
deff(z=f(x,y)', 'z=exp(x).*sin(y)^2');
fplot3d(x,y,z)
```

9.5. Gráficas de contorno

Máxima

Primero se define la función, luego unos parámetros de ploteo y, por último, se hace el contorno.

```
z(x,y) := 1+(1/4000)*(x^2+y^2)-
cos(x)*cos(y/2^0.5)$
set_plot_option ([gnuplot_preamble,
"set cntrparam levels 10"])$
contour_plot (z, [x,-6,6], [y,-6,6]);
```

Scilab

Primero se definen los vectores `x` e `y`, luego se crea la función y por último se crea el contorno.

```
x=-6:0.01:6; o x=linspace(-6,6,100);
y=-6:0.01:6;

function f = z(x,y);
f=1+(1/4000)*(x.^2+y.^2)-cos(x).*cos(y/2^0.5)
endfunction

contour (x,y,z,10)
```

9.6. Otros gráficos

Máxima	Scilab
Histograma	`histplot(nbins, data)`
Subplots	`subplot(n_rows, n_cols, idx)`

10. Límites

10.1. 1 variable

Ponemos de ejemplo el límite $\lim_{x\to\pi/2}\frac{\sin(x)}{\cos(x)}$.

Máxima

Simplemente usamos la función `limit`.

`limit(sin(x)/cos(x),x,%pi/2,minus) >>` ∞
`limit(sin(x)/cos(x),x,%pi/2,plus) >>` $-\infty$
`limit(sin(x)/cos(x),x,%pi/2) >> infinity`

Para límites en infinito se emplean los términos `inf` y `minf` en `limit`.

Scilab

Primero se definimos un `x` arbitrariamente cercano al valor del límite, y aplicamos `x` a la función.

```
x1=%pi/2 + 1D-15;
limit1=sin(x1)/cos(x1) >> -9.5D+14
x2=%pi/2 - 1D-15;
limit2=sin(x2)/cos(x2) >> 8.5D+14
```

11. Diferenciación

11.1. 1 variable

Tomamos la función $f(x) = x^2\sin(x)$.

Máxima

La función calcula la derivada n-ésima
`diff (x^2*sin(x),x,n);`

Scilab

Sólamente podemos hacer una aproximación numérica empleando la definición de derivada

$f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$.

```
x = 0:0.01:1;
h = 1D-5;
x_h = x + h;
d_f = (x_h.^2.*sin(x_h) - x.^2.*sin(x))/h;
```

11.2. Varias variables

El procedimiento es similar, en Máxima, solo que se define la variable de diferenciación:

`diff (x^2-x*y+sin(x*y^2), y).`

En el caso de tener funciones con dependencias, como $f(u, v)$ y $g(u, v)$ éstas se establecen con la función `depends`.

`depends([f,g],[u,v]) >> [f(u,v), g(u,v)]`

12. Integración

Máxima

`integrate (f,x) / integrate (f,x,a,b)`

Podemos definir cambios de variable en tres pasos:
1) Definimos la integral SIN EVALUAR
`'integrate(f,x);`
2) Cambiamos la variable definiendo la función de cambio $g(t, x)$. Por ejemplo, el cambio $t = x^2$ se definiría como `g = x^2-t`
`changevar(%, g, t, x);`
3) Evaluamos la nueva integral
`ev(%, nouns)`

Scilab

```
function y=f(x), y=x^2, endfunction
integrate('x^2', 'x', 0, 1)
intg(0, 1, f)

x = 0:0.01:1
intsplin(t, t^2)
inttrap(t, t^2)
```

12.1. Integrales dobles y triples

Máxima

Simplemente se concatenan varios integrate:
`integrate(integrate(f(u,v),v,v1,v2),u,u1,u2)`

Y lo mismo para integrales triples:

```
integrate(integrate(integrate(
f(u,v,w),w,w1,w2),
v,v1,v2), u,u1,u2)
```

Scilab

Para las integrales definidas, definimos los vectores `X`, `Y`, `Z`, que son los puntos de los triángulos o tetraedros que definen la malla de la superficie.

La función f se define con `deff`:
`deff('k=f(x,y,z)','k=x*y*z');`

```
[res, error] = int2d(X, Y, f)
[res, error] = int3d(X, Y, Z, f)
```

13. Sumas y series (Máxima)

Las sumas de una expresión se realizan con `sum(expresión, i, i_0, i_f)`. En el caso en el que se quiera resolver la suma, se añade `, simpsum` a la expresión anterior.

Los desarrollos de Taylor se hacen con la función `taylor(expresión, variable, x_0, nivel_de_truncamiento)`. El desarrollo va a tener un término más que el nivel de truncamiento.

La función `pade` hace el inverso de Taylor para funciones racionales. `pade` necesita un objeto de desarrollo de Taylor, y toma los siguientes argumentos: `pade(expresión_taylor, grado_numerador, grado_denominador)`.

`taylor (1 + x + x^2 + x^3, x, 0, 3)$ pade (%, 1, 1);`

La función `powerseries` hace un desarrollo de potencias de la expresión en x_0. La expresión es `powerseries(expresión, variable, x_0)`.

14. Funciones y bucles

14.1. Funciones

Máxima	**Scilab**

Sólo acepta un argumento de salida.
`fun(arg1, arg2, ...) := *Cuerpo de la función`
	`function [o1, o2, ...] = f(arg1, arg2, ...)`
	`ó  function o = f(arg1, arg2, ...);`
	`// Código de la función. o1, o2, ... deben`
	`// asignarse en algún punto de la función.`
	`// en el caso de ser o una lista,`
	`// se puede escribir`
	`//o(1)=... , o(2)=...`
	`endfunction`

14.2. Bucles

Máxima	**Scilab**
	Scilab permite hacer uso del típico `if`, así como del caso `select`

`if cond then resp1 else resp2`	`if (cond) then`
	`resp1`
	`elseif cond2`
	`resp2`
	`else resp3`
	`end`

Máxima no acepta cláusulas `elseif` como Scilab, sino que estas se programan directamente:	El caso `select` toma el valor de una variable, y ejecuta las condiciones según el valor de la variable.

`if cond1 then resp1`
`else if cond2 then resp2`
`else resp3`
	`select var`
	`case opcion1 resp1`
	`case opcion2 resp2`
	`else resp3`
	`end`

14.3. Bucles

Máxima	Los bucles derivan del bucle `for`, que tiene tres variantes:

```
for v: v1 step v2 thru v3 do expr
for v: v1 step v2 while cond do expr
for v: v1 step v2 unless cond do expr
for v in lista do expr
```

Por un lado, el parámetro `step` no es necesario si, por ejemplo, el paso es 1. Así, podemos conseguir hacer

```
for v: v1 thru v3 do expr
```

Por último, podemos escribir `while` y `unless` sin `for`:

```
while cond do expr
unless cond do expr
```

Scilab

```
for var = v_0:step:v_f
ó for var = v_0:v_f
ó for var = lista

expr
end
```

La estructura de los bucles `while` es:

```
while (cond) expr end
```

La expresión `break` termina el bucle cuando se invoca; y la expresión `continue` hace que se salte ese ciclo y pase al siguiente. En ambos casos se pasa por alto el resto de la expresión del `for` o `while`.

15. Ecuaciones diferenciales

Como siempre, Máxima te puede dar una solución simbólica, y Scilab te dará una solución numérica.

Máxima

En primer lugar, siempre se define la ecuación: ($\frac{d}{dx}y = \frac{d^2}{d^2x}y + x$) `ec:'diff(y,x) = x + 'diff(y, x, 2)` Si la ecuación es EDO de hasta grado 2, se resuelve con `ode2(ec, y, x) >> y=%k1*%e^x+(x^2+2*x+2)/2+%k2` Si queremos resolverla con condiciones iniciales: `ic2(%, x=1, y=-1, diff(y,x)=2)` Si por el contrario queremos poner puntos de contorno: `bc2(%, x=-1, y=3, x=0, y=0)`

Scilab

Se define la ecuación diferencial (del tipo $\frac{d}{dt}y = f(t, y)$ con una función: `function dydt=func(t,x,x0) dydt=2^t*log(2)*x0; endfunction` Ahora hacemos set de x0, t0 y t como vector de tiempos: `x0=10, t0=0, t=0:1:10` `x = ode('rk', x0, t0, t, func)`

16. Optimización

En máxima, la optimización se resuelve de manera analítica. Para resolverla de manera numérica, puede realizarse con Scilab. Para ello empleamos la función `[x, fval, exitflag, output] = fminsearch(func, x0, opt)` Donde `x` y `fval` son el valor de x óptimo y su correspondiente valor. `func` es la función a optimizar, `x0` es un valor inicial para buscar el mínimo, y `opt` son unas opciones de optimización. Un ejemplo es `opt=optimset("TolX", 0.0001)`, que establece un valor de tolerancia específico.

Para problemas más complejos de optimización, se puede emplear la función `optim`. Esta función requiere la función de coste, así como el gradiente. Un ejemplo para la función $S = ab + V/b + V/a$, $\partial S/\partial a = 2 * (b - V/a^2)$, $\partial S/\partial b = 2 * (a - V/b^2)$. Así, el código es:

```
function [S, dS, ind] = paral(x, V, ind)
a=x(1)
b=x(2)
c=V/(a*b)
S=2*(a*b+V/b+V/a)
```

```
dS(1) = 2*(b-V/a^2)
dS(2) = 2*(1-V/b^2)
endfunction
[S,x,dS]=optim(paral, [1;1])
```

17. Operaciones con ficheros

	Máxima	Scilab
Cambiar directorio de trabajo	`chdir(dir)`	`chdir(dir)`

Abrir un archivo	`read_matrix(dir)`	`a = file('open', dir, 'old')`
		`n = read(a, n_filas, n_cols)`
		`file('close', a)`

Abrir un archivo (2)	`read_list(dir)`	`n = csvRead(dir, separator))`
Escribir matriz en archivo	`write_data(n, dir)`	`csvWrite(n, dir, separator)`
Buscar un archivo	`file_search(dir)`	
Ordenar una lista/matriz		`[listaord, indices] = gsort(lista, opciones, orden)`

A la hora de ordenar, las opciones son `r`: ordena la fila, `c`: ordena la columna, `g`: ordena todo, `lr`: ordena la file lexicográficamente, `lc`: ordena la columna lexicográficamente. Con respecto al orden, `i`: increase; y `d`: decrease.

18. Estadísticos descriptivos, y ajuste de datos

	Máxima	Scilab
Rango	`range()`	`strange(dir)`
Máximo	`smax()`	`max('open', dir, 'old')`
Mínimo	`smin()`	`min()`
Media	`mean()`	`mean()`
Desviación (s/N-1)	`std1()`	`st_deviation()`
Desviación (s/N)	`std()`	

Ajuste lineal de datos

Máxima

```
m: matrix([a1,b1],[a2,b2],...,[an,bn])$
load("lsquares")$

pol1: lsquares_estimates(m, [x,y], y=a1*x+a0,
[a1, a0]), numer;
mse1: lsquares_residual_mse(mdatos, [x,y],
y=a1*x+a0, first(pol1))

pol2: lsquares_estimates(m, [x,y],
y=a2*x^2+a1*x+a0, [a2, a1, a0]), numer;
mse2: lsquares_residual_mse(mdatos, [x,y],
y=a2*x^2+a1*x+a0, first(pol2))
```

Scilab

```
[m,b,sig] = reglin(x,y)
```

Para hacer ajustes polinómicos debería crearse la función

```
function cf = polyfit(x,y,n)
A = ones(length(x),n+1)
for i=1:n
A(:,i+1) = x(:).^i
end
cf = lsq(A,y(:))
endfunction
```

Interpolación

Máxima

En el caso de máxima, las interpolaciones que realiza son una interpolación lineal, una interpolación cubica con splines y una interpolación de Lagrange. El código para las interpolaciones y las gráficas es el siguiente:

```
m: matrix([a1,b1],[a2,b2],...,[an,bn])$
load(interpol)$

y: linearinterpol(mdatos)$
ftl(x):= ''y;

y: cspline(mdatos)$
fs(x):= ''y;

y: lagrange(mdatos)$
fpl(x):= ''y;

xm:list_matrix_entries(submatrix(mdatos,2));
ym:list_matrix_entries(submatrix(mdatos,1));

plot2d([[discrete, x, ym],
ftl(x), fs(x), fpl(x)],
[x, 0.5, 6.5], [style, points,
lines, lines, lines])
```

Scilab

```
ipl = interp1(x, y, xp, tipo, 'extrap')
```

Aquí, `x, y` son los datos, `xp` es la lista de puntos de `x` donde se va a hacer la interpolación, y `tipo` es el tipo de interpolación. Por ejemplo, `'linear'` hace una interpolación linear punto a punto, `'spline'` hace una interpolación por splines, y `'nearest'` hace una interpolación escalonada.

Así por ejemplo, para hacer las interpolaciones y graficar todo a la vez:

```
mdatos = [1 0; 2 1; 3 3; 4 2; 5 4; 6 5]
x = mdatos(:, 1)
y = mdatos(:, 2)
xp = min(x) - 0,5:0.05:max(x)+0.5
ypl = interp1(x, y, xp, 'linear', 'extrap')
ypc = interp1(x, y, xp, 'spline', 'extrap')
plot(x, y, 'o', xp, ypl, 'r', xp, ypc, 'k')
```

Si queremos hallar el valor de la interpolación lineal, podemos aplicar la siguiente función:
`interpln(mdatos', 2.5);`