Universitatea Babeş-Bolyai, Facultatea de Matematică şi Informatică
Secţia: Informatică engleză, Curs: Dynamical Systems

1. A brief introduction to Maple 8

- Each Maple command ends with a semicolon (;) or a colon (:) and the computer does not execute the command until the line is entered (by hitting the return key). So,
  ; or : must follow every command

- Maple has Help and the question mark **?** inserted in the command line and followed by the operator, the function, the command … leads to the help page of this. You are strongly encouraged to use the on-line help, as it is emphasized also below.

- Operators (**?+;**)
  addition: **+** subtraction: **-** multiplication: ***** Division: **/**
  exponentiation: **^** or ****** repetition: **$**

- Constants (**?Pi;**)
  π: **Pi** e: **exp(1)** √ -1: **I** ∞: **infinity**

- Elementary functions (**?exp;**)
  exponential: **exp(x)** natural logarithm: **ln(x)**
  absolute value: **abs(x)** square root: **sqrt(x)**
  trigonometric: **sin(x), cos(x), tan(x), sec(x), cot(x), csc(x)**
  Note the trigonometric functions in Maple require angles measured in radians
  inverse trigonometric functions: **arcsin(x), arccos(x), arctan(x), arctan(y,x)**
  hyperbolic functions: **sinh(x), cosh(x), tanh(x), sech(x), csch(x), coth(x)**

- Assign a value to a variable using **:=**
  Examples: **a:=1;** # here the variable is denoted ‘a’ and the assigned value is a *number*
  expr:=x*y+2*x+y^3-5; #here the variable is denoted ‘expr’ and the assigned value is an *expression*
  eq:=x+2=3; #here the variable is denoted ‘eq’ and the assigned value is an *equation*
  Note that, to denote a variable one can use any letter or group of letters (word) except the ones that already are assigned (for example Pi, exp, sin, D, …).

- Unassign Examples: **a:=’a’; expr:=’expr’; eq:=’eq’;**
  Note that the command **restart;** unassign all of the variables and undo all the commands

- Evaluation of a number using floating point arithmetic
  Examples: **evalf(1/2); evalf(sqrt(3)); evalf(sqrt(3),15); evalf(Pi); evalf(Pi,20);**

- The previously computed result is returned by **%** (**?%;**)

- Useful commands when working with expressions (**?expand; ?subs; ?eval**)
  expand **factor** **simplify** **subs(x=1, expr)** **eval(expr, x=1)**

- Define a function with a given expression
  Examples: **f:=x -> 2*x+1;** # this defines a real function f of one variable x
  g:=(x,y)->x*y+x^2-3; # this defines a real function g of 2 variables (x,y)

- For a defined function f one can compute its value in a given point, or the values of its derivatives in a given point (**?D;**)
  f(1): **f(1);** f'(1): **D(f)(1);** f''(1): **(D@@2)(f)(1);**

  Note that **f(x);** # will return the expression of f, while
  D(f)(x); #will return the expression of the first order derivative of f, i.e. f'(x)
  (D@@n)(f)(x); #will return the expression of the n-th order derivative of f

- Differentiation of a function with a given expression (**?diff;**)
  Examples: **diff(2*x+3,x);** #will return 2 **diff(2*x+3,t);** will return 0
  diff(f(x),x); #will return f'(x) **diff(x(t),t);** will return x'(t)
  diff(f(x),x$n); #will return the n-th order derivative of f

- Integration of a function with a given expression (**?int;**)
  Examples**:** **int(cos(x),x);** #will return sin(x)
  int(cos(x),x=0..Pi); # will return 0

- Limit of a function with a given expression in a given point (**?limit;**)
  Examples: **limit(sin(x)/x, x = 0);** #will return 1

Note that the commands **diff, int, limit, expand, factor** require *expressions* for arguments.
If an *expression* is assigned to a variable f (for example **f:=2*x^3-3;**) then only **diff(f,x);** is correct. In this case do not write f(x)!
But if f is defined as *function* of variable x (for example **f:=x->x^2-5;**) then only **diff(f(x),x);** is correct since f(x) is an expression this time, while f is a function.

- If a variable expr is defined as an *expression*, the command **f:=unapply(expr,x);** turns the expression into a *function* f of variable x.

- Solve algebraic equations and systems with **solve** and **fsolve**
  solve(equation,x); #will give the *exact solutions*, *if possible*, of a given equation whose unknown is x. Maple may not list all solutions; for example, when trigonometric functions are involved.
  fsolve(equation,x); #will give an approximate solution found numerically
  fsolve(equation,x=a..b); #it is of great help to indicate an interval [a,b] where to look for the solution
  solve({eq1,eq2},{x,y}); #will solve a system of two equations and two unknowns denoted here x and y

- Plot the graph of a real function of one or two real variables (**?plot;**)
  Examples: **plot(x^2+2*x-1, x=-5..6);** #plots the graph of the function f(x)=x^2+2x-1 for x in the interval [-5,6]. Note the graph is a planar curve.
  plot({sin(x),cos(x)}, x=-5..6); #plots simultaneously the graphs of the functions f(x)=sin x and g(x)=cos x for x in the interval [-5,6].
  plot3d(x^2+y^2, x=-2..2, y=-2..2); #plots in 3d the graph of the function f(x,y)=x^2+y^2 for (x,y) in the rectangle [-2,2]x[-2,2]. Note the graph is a surface in 3d.

- Plot a planar curve given by parametric equations
  Examples: **plot([2-t^2, t-t^3, t=-2..2]);** #plots the planar curve given by the parametric equations x=2-t^2, y=t-t^3 when the parameter t varies in the interval [-2,2].
  plot([[2*exp(-t), exp(-2*t),t=0..3], [-2*exp(-t), -exp(-2*t),t=0..3]]);
  #plots simultaneously the planar curve given by the parametric equations x=2e^(-t), y=e^(-2t), for t in the interval [0,3] and the planar curve given by the parametric equations x=-2e^(-t), y=-e^(-2t), for t in the interval [0,3]

- Plot a planar curve given by an implicit equation (**?implicitplot;**)
  This command works only within the plot package, hence one first needs to call it using the command **with(plots):**
  Examples: **implicitplot(x^2+y^2=4,x=-3..3, y=-3..3);** #plots a circle with the center in the origin of coordinates and radius 2
  implicitplot(4*x^2+y^2=4,x=-3..3, y=-3..3); #plots an ellipse
  implicitplot(y=x^2, x=-3..3, y=0..10); #plots a parabola

- 2D and 3D animation work only after calling the plot package, i.e. first we type **with(plots):**
  Once we execute the maple command, we need to left click on the picture. At that point choose Animation and Play.
  Examples: **animate(exp(-(x + t)^2), x = -10..10, t = 0..5, frames = 50);**
  animate(sin(x*t),x=-4*Pi..4*Pi,t=0..1,color=red);
  animate([r*cos(t), r*sin(t), t = 0..5], r = 0..1, frames = 50);
  animate3d(cos(t*x)*sin(t*y), x = -Pi..Pi, y = -Pi..Pi, t = 1..2, color = cos(x*y));
  animate3d(t*y^2-x^2+x^4,x=-2..2,y=-2..2,t=0..2);

2. The command `dsolve`

Is used to find the general solution of a scalar differential equation or of a differential system. Also, it can be used to find the solution of an IVP or of a problem formed by a differential equation and any other type of conditions.

- First we show how to introduce a differential equation in Maple. Note that, first we have to clarify the notations for the unknowns and, respectively, for the independent variable. An important thing to remember is that, in Maple, we have to write each time the independent variable.

  Examples. The first order differential equation x'=x , where the unknown is the function x of independent variable t, can be introduced in Maple as follows:
  ```
  eq1:= diff(x(t),t) = x(t);
  ```
  Here we chose to assign the differential equation to the variable `eq1`

  The second order differential equation x"+5x'-6x=0 , where the unknown is the function x of independent variable t, can be introduced in MAPLE as:
  ```
  eq2:= diff(x(t),t$2)+5*diff(x(t),t)-6*x(t)=0;
  ```

- In order to find the general solution of each of the two equations we use `dsolve`

  Examples. `dsolve(eq1,x(t));` `dsolve(eq2,x(t));`

  Notice that the general solution evidentiate as many arbitrary real constants _C1, _C2, ... as the order of the differential equation.

Notice that there is the possibility to know details on the method used by Maple to solve the differential equation by setting `infolevel[dsolve]:=3` . The default setting is 1.

- In order to obtain the solution of a problem consisting of one or more differential equations and one or more conditions (like, for example, an IVP) we use again `dsolve`

  Examples. To solve the IVP x'=x, x(0)=-3 , first we prefer to assign the initial condition to another variable, `ic1:= x(0)=-3;` then to use the command `dsolve({eq1,ic1},x(t));` Maple will return `x(t) = -3*exp(t)`

  To solve the IVP x"+5x'-6x=0, x(0)=0, x'(0)=-1 again we assign the initial condition to a variable, i.e. `ic2:= x(0)=0, D(x)(0)=-1;` and then we use the command `dsolve({eq2,ic2},x(t));`

To find the general solution of the system x'=-y, y'=x+y we type
```
dsolve({diff(x(t),t)=-y(t),diff(y(t),t)=x(t)+y(t)},{x(t),y(t)});
```

- In the case that we need to work further with the expression of a solution, we need to assign to a new variable this expression. We explain in the sequel how one can do this. First it is convenient to assign to a variable the output of the command `dsolve,` i.e. `sol:=dsolve({diff(x(t),t)-x(t)=0,x(0)=3},x(t));`
  This output is the *equality* `x(t) = -3*exp(t)` , hence we do not have yet the *expression* of the solution, which is *the right-hand side* of this equality.
  One can obtain *the expression* using the `rhs` command `sol1expr:= rhs(sol);`
  With this *expression* we can do all the operations explained in Lab1.
  We explain the same in the case of an IVP for a system of 2 equations with 2 variables.
  `sol:=dsolve({diff(x(t),t)-x(t)=0,diff(y(t),t)+y(t)=0, x(0)=3, y(0)=2},x(t));`
  The output is `x(t) = -3*exp(t), y(t)=2*exp(-t)`
  One can obtain *the expression* of x(t) and, respectively y(t), using the `rhs` command as follows
  `xsolexpr:= rhs(sol[1]); ysolexpr:= rhs(sol[2]);`

3. The command piecewise

- Here we show how to define in Maple a piecewise-valued function. The command is `piecewise(cond_1, expr_1, cond_2, expr_2, ..., cond_n, expr_n, expr_otherwise);`
  where `expr_i` is an expression, `cond_i` is a relation or Boolean combination of inequalities, and `expr_otherwise` is an (optional) default expression.

The semantics are as in a case statement: if `cond_1` is true then `expr_1`, else if `cond_2` is true then `expr_2`, and so on. `expr_otherwise` gives a default case which corresponds to all conditions being false. The default for `expr_otherwise` is 0.

A condition can be a single equality or inequality, or a boolean combination of inequalities, such as `t<3 or 0<t and t<=Pi` . Equalities cannot be used in a boolean expression. The conditions can contain relations with polynomials, `abs`, `signum`, or piecewise functions, such as `0<t^2-4 and 0<x or abs(x)<4` . In all cases, `t` is assumed to be a real variable.

A function defined using `piecewise` can be differentiated, integrated, simplified, plotted, and used in some types of differential equations.

To plot a discontinuous function we recommend to use the option `discont=true` .

4. Linear Algebra

- First load some additional packages: `with(Student[LinearAlgebra]):` `with(LinearAlgebra): with(linalg):`

- This is how we introduce vectors and matrices:

  `[1,5,0]` this is a row vector

  `<1,5,0>` this is a column vector

  `Matrix([[1,5,0],[-1,-5,0]])` this is the 2x3 matrix whose first row is [1,5,0] and second row is [-1,-5,0].

  `Matrix(<<1,-1>|<5,-5>|<0,0>>)` this is the same matrix, but it is given by columns

  `DiagonalMatrix([1,2,3])` this is the 3x3 diagonal matrix with the elements 1,2,3 on the main diagonal

  `IdentityMatrix(3)` this is the 3x3 identity matrix

  Once a matrix A is introduced, its elements are `A[1,1]`, `A[1,2]` …

- This is how we compute the determinant, the inverse, the eigenvalues, the eigenvectors, the characteristic polynomial of a square matrix A:

  `Determinant(A)` or `det(A)`

  `inverse(A)` or `A^(-1)`

  `Eigenvalues(A)` the result is a column vector that contains the eigenvalues of A

  `lam,P:=Eigenvectors(A)` the result gives first a column vector that contains the eigenvalues of A (the variable named lam contains this vector) and a matrix (here named P) whose columns are linear independent eigenvectors of A corresponding to the order of the eigenvalues as in the vector lam.

  `CharacteristicPolynomial(A,r)` we can specify the name of the variable, here is r

- Multiplication of two matrices A and B is with a dot, instead of star: `A.B`

- This is how we compute the Jordan form of a square matrix A, i.e. the “simplest” matrix similar to A. We have that A is diagonalizable if and only if its Jordan form is diagonal.

  `JordanForm(A)` this is gives the Jordan form of A, denoted B

  `JordanForm(A, output='Q')` this is gives the transition (or similarity matrix), i.e. the invertible matrix P such that A=PBP^(-1). We know that, when A is diagonalizable, the columns of P are linearly independent eigenvectors of A.
- This is how we compute the exponential of a matrix A: `MatrixExponential(A)`
- This is how we compute the limit as t goes to infinity of exp(tA): `Map(limit,MatrixExponential(t*A),t=infinity)`

5. The command `dfieldplot`

- This command works only after loading the package DEtools. Hence, before we must put `with(DEtools):`
- Given either a system of two first order autonomous differential equations, or a single first order differential equation, `dfieldplot` produces a direction field plot. There can be only one independent variable.
- The direction field presented consists of a grid of arrows tangential to solution curves. For each grid point, the arrow centered at (x, y) will have slope $\frac{dy}{dx}$. For system of two first order autonomous differential equations this slope is computed using $\left(\frac{dy}{dt}\right)/\left(\frac{dx}{dt}\right)$, where these two derivatives are specified in the first argument to `dfieldplot.`
- The command is `dfieldplot(deqns, vars, trange, xrange, yrange)` where `deqns` is a list or set of first order ordinary differential equations, `vars` is a list or set of dependent variables, `trange` is the range of the independent variable, `xrange` is the range of the first dependent variable, `yrange` is the range of the second dependent variable.
- Examples: Represent the direction field of y'(x)=y(x) in the box [-1,1]x[-2,2].

```
dfieldplot(diff(y(x),x)=y(x), y(x), x=-1..1, y=-2..2);
```

Represent the direction field of the autonomous system x'(t)=-y(t), y'(t)=x(t) in the box [-1,1]x[-2,2].

```
dfieldplot([diff(x(t),t)=-y(t), diff(y(t),t)=x(t)],
[x(t),y(t)], t=-3..3, x=-1..1, y=-2..2);
```

6. The command `DEplot`

- This command works only after loading the package DEtools. Hence, before we must put `with(DEtools):`

- Given a set or list of initial conditions, and a system of first order differential equations or a single higher order differential equation, DEplot plots solution curves, *by numerical methods*. The default method of integration is `method=rkf45`.

- The command is `DEplot(deqns, vars, trange, inits)` where `deqns` is a list or set of first order ordinary differential equations, or a single differential equation of any order, `vars` is a list or set of dependent variables, `trange` is the range of the independent variable, while `inits` is a set or list of lists; initial conditions for solution curves

- Examples: Represent simultaneously the solution curves with initial values y(0)=1, and, respectively, y(0)=-1 of the differential equation y'(x)=y(x).

```
DEplot(diff(y(x),x)=y(x), y(x), x=0..2,[[y(0)=1],[y(0)=-
1]]);
```

Represent simultaneously the solution curves with initial values y(0)=1, y'(0)=0 and, respectively, y(0)=-1, y'(0)=0 of the differential equation y''(x)=y(x).

```
 DEplot(diff(y(x),x$2)=y(x), y(x), x=0..1,
[[y(0)=1,D(y)(0)=0],[y(0)=-1,D(y)(0)=0]]);
```

Represent the orbit corresponding to the initial values x(0)=1, y(0)=0 of the autonomous system x'(t)=-y(t), y'(t)=x(t) .

```
       DEplot([diff(x(t),t)=-y(t), diff(y(t),t)=x(t)],
[x(t),y(t)], t=0..9, [[x(0)=1,y(0)=0]]);
```

Represent simultaneously the orbits corresponding to the initial values x(0)=1, y(0)=0, and, respectively x(0)=1, y(0)=1 of the autonomous system x'(t)=-y(t), y'(t)=x(t) .

```
DEplot([diff(x(t),t)=-y(t), diff(y(t),t)=x(t)],
[x(t),y(t)], t=0..9, [[x(0)=1,y(0)=0],[x(0)=1,y(0)=1]],
linecolor=[green,gold]);
```

7. The command `contourplot`

- produces a set of level curves of the input function for a discrete set of values (i.e. levels) of the third coordinate. It work only after loading the package plots: **with(plots):**
- Examples: Represent the level curves of H(x,y)=sin(xy) in the box [-1,1]x[-1,1]
  `contourplot(sin(x*y), x=-3..3, y=-3..3);`

8. Finding the Jacobian matrix and its eigenvalues

We consider a function of two real variables with two real components denoted (f1(x,y), f2(x,y)). In order to find the eigenvalues of its Jacobian matrix calculated in a point (p1,p2) we use the commands `Jacobian` and `eigenvalues`. They work after loading the packages linalg and VectorCalculus, hence first write `with(linalg): with(VectorCalculus):`
Then use

```
Jm:=Jacobian([f1(x,y), f2(x,y)],[x,y]);
A:=subs([x=p1, y=p2], Jm);
eigenvalues(A);
```

9. Map iterations

We show how to find the first 30 iterations of a function f starting from x0. Then we represent them using the command pointplot.

```
> x:=x0;
> for i from 1 to 30 do x:=f(x): psi(i):=x: print(x); od:
> points:=[[n,psi(n)]$n=1..30]:with(plots): pointplot(points,
symbol=circle);
```

10. Euler's methods

We will apply Euler's method to solve the initial value problem y'=2xy+exp(x^2), y(0)=1, in the interval [0,1.5] with step size h=0.1. Since we would like to compare with the exact values, first we find the exact solution of this IVP and we represent the graph of the solution and the direction field of the differential equation.

```
> restart:
> with(DEtools):
> f:=(x,y)->2*x*y+exp(x^2);    this is the right-hand side of the differential equation
> dsolve({diff(y(x),x)=f(x,y(x)),y(0)=1});phi:=unapply(rhs(%),x);
> DEplot(diff(y(x),x)=f(x,y(x)),y(x),x=0.. 1.5, [[y(0)=1]],
y=1..25);
> h:=0.1;    # this is the stepsize. Hence, the number of steps necessary to cover the interval
[0, 1.5] is 15
> x:=0; y:=1;    # (0,1) is the starting point, as given by the initial condition
> for i from 1 to 15 do y:=y+h*f(x,y): psi(i):=y: x:=x+h:
print(x,y,phi(x),abs(y-phi(x))); od:
> points:=[[n,psi(n)]$n=1..15]:with(plots): pointplot(points,
symbol=point);plot(phi(t),y=1..1.5);
```

We will apply the now the improved Euler's numerical method. We have to change only the numerical formula. We must restart!

```
> restart:
> phi:=x->(x+1)*exp(x^2);
> h:=0.1; x:=0; y:=1;
> f:=(x,y)->2*x*y+exp(x^2);
> for i from 1 to 15 do y:=y+h/2*f(x,y)+h/2*f(x+h,y+h*f(x,y)):
psi(i):=y: x:=x+h: print(x,y,phi(x),abs(y-phi(x))); od:
> points:=[[n,psi(n)]$n=1..15]:with(plots): pointplot(points,
style=point);plot(phi(t),t=1..1.5);
```