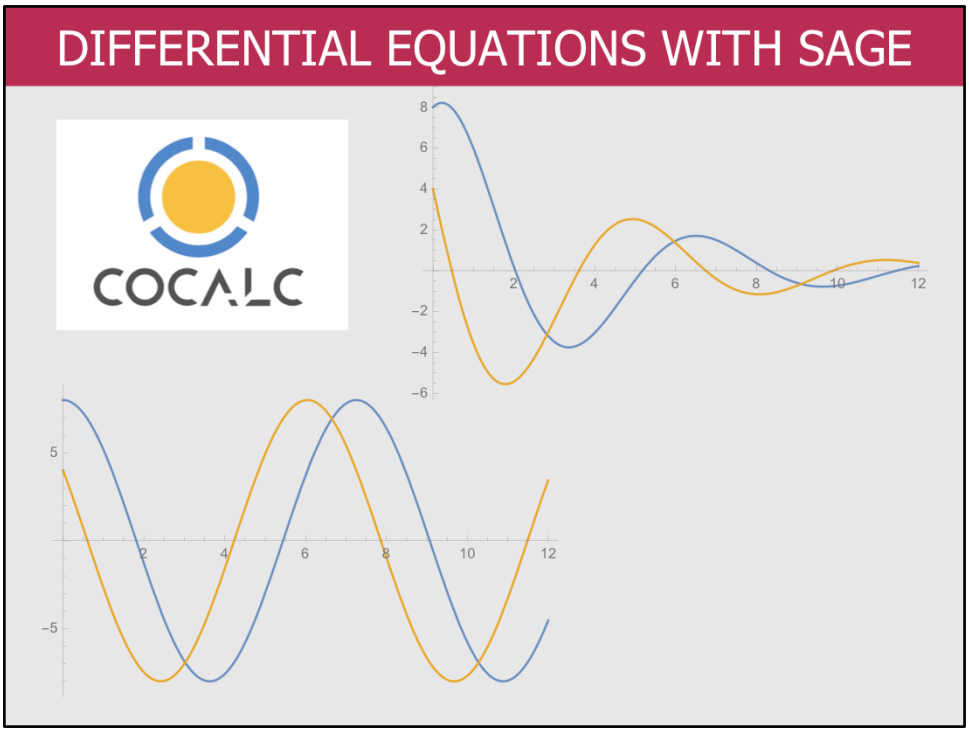
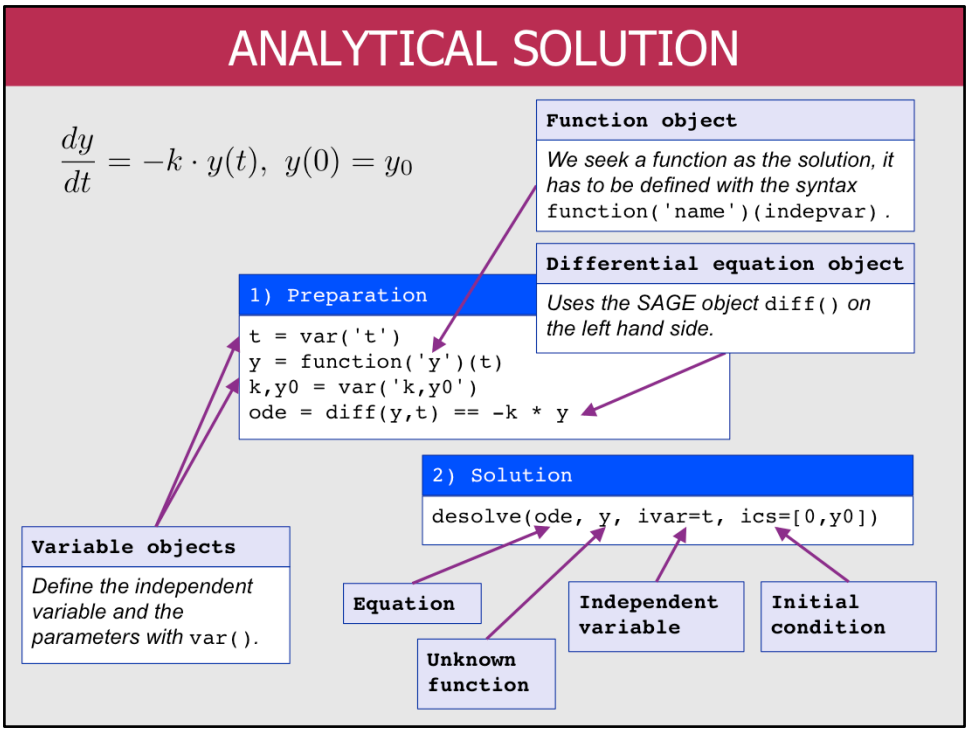
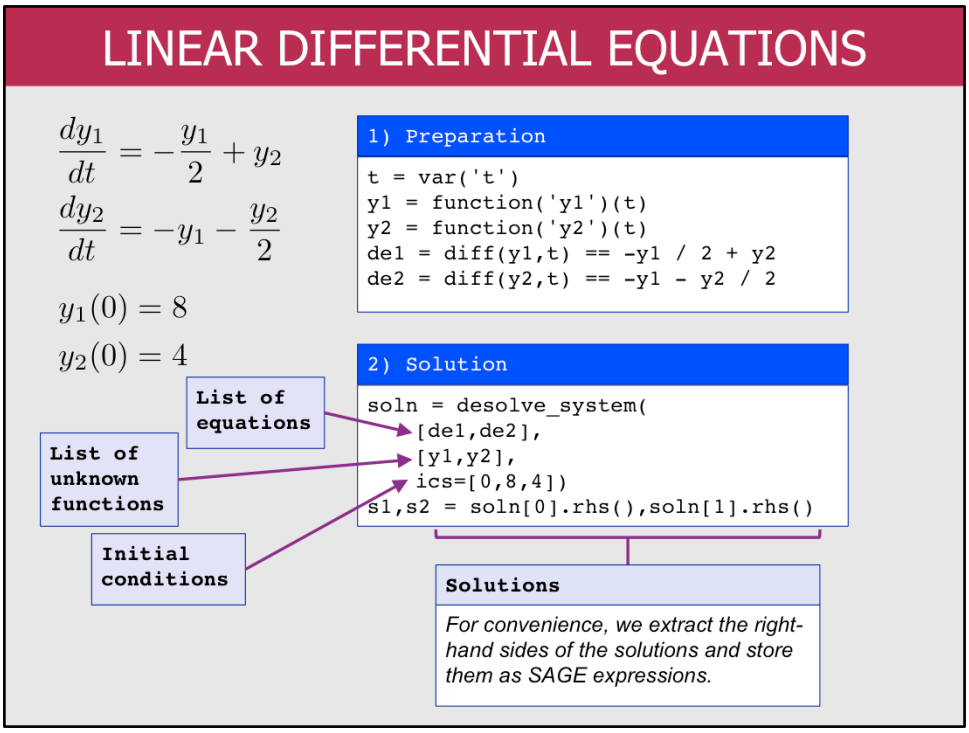
DIFFERENTIAL EQUATIONS WITH SAGE



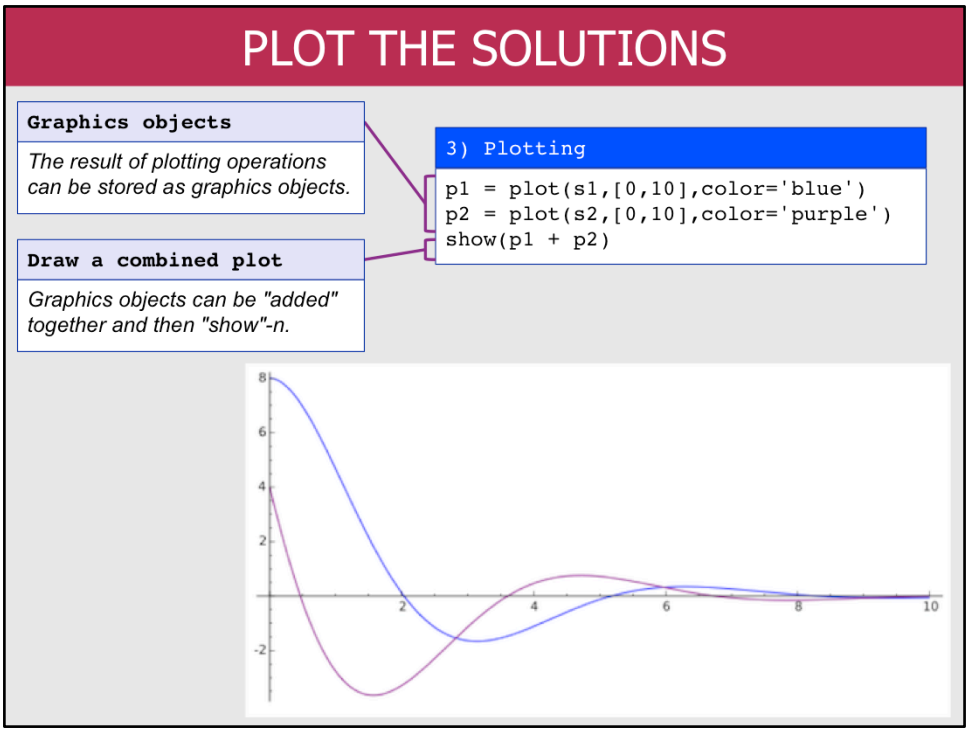
Most of the time we will solve differential equations. This handout guides you through the technical details of how to solve differential equations in the CoCalc environment.



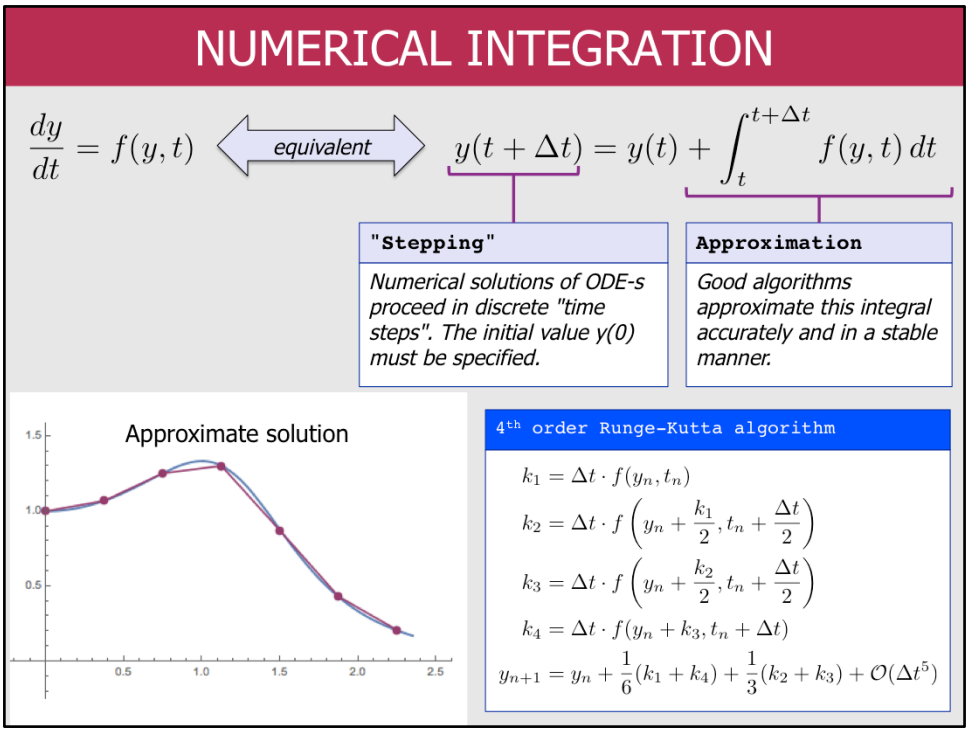
In order to perform symbolic computations, you need to tell SAGE about the variables and functions (in the mathematical sense, not in the usual Python sense!) you are going to use. This means that SAGE symbolic variables must be created by invoking a `var` constructor. Similarly, mathematical function objects are created by the `function(name)(independent variable)` syntax. Here we create an independent variable `t` (for "time"), and the function `y` which is going to represent the solution of a simple differential equation. The equation itself is defined by the syntax `diff(y,t) == right_hand_side_expression`. Equations are also SAGE objects and therefore can be saved in a (Python) variable.
The equation is solved by invoking the `desolve` method. You pass the equation, the function object representing the unknown function, the variable object representing the independent variable ("ivar"), and the initial conditions ("ics"). Because the equation in this example is very simple, `desolve` can return the solution in symbolic form as a SAGE expression.



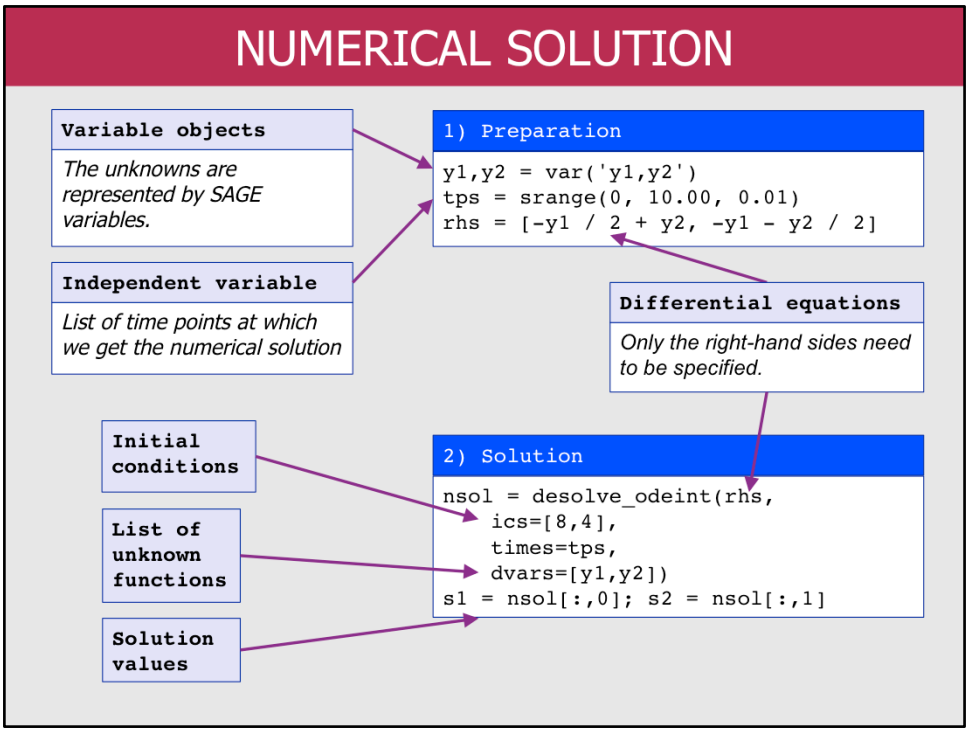
Because we have two equations, we need to define two function objects to represent the two solutions y1 and y2. The equations themselves are saved in separate SAGE equation objects and then bundled into a list, but they can also be put in a list directly.
The solution is obtained by invoking the `desolve_system` function that takes a list of differential equations, a list of the unknown functions, and the initial conditions (ics=...). The first element of the initial condition list is the value of the independent variable (usually 0), and then the initial values of the first, second, ... , n-th functions. The solution is returned as a two-element list. We extract the right-hand-sides of the solution expressions and store them in SAGE objects. This is not necessary but will simplify plotting.



The plotting commands in SAGE generate graphics objects which can be "added" to each other and `show()`-n later. If you invoke a single `plot` command then you see the resulting graphics immediately.



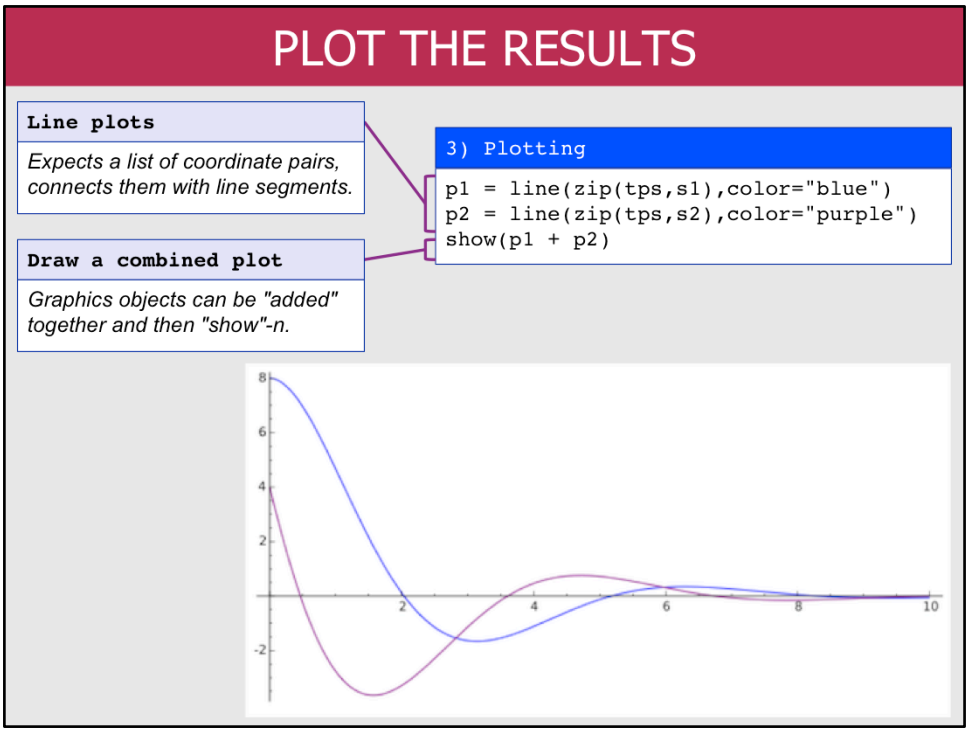
Fortunately, it is also possible to solve differential equations using digital computers. Lots of numerical methods are available to solve differential equations with high accuracy. The trick is to re-formulate the differential equation as an integral equation, and then use a stepwise approximation to the right-hand side. The slide shows the venerable "4th-order Runge-Kutta" method. Depending on the properties of the differential equation, other methods might be more efficient.



Solving differential equations numerically looks slightly different. The unknown functions are now represented by SAGE variables, not SAGE functions, because they are not symbolic expressions but rather a list of values calculated at prescribed "time points" (values of the independent variable). For the same reason the independent variable is given as a list of abscissa points. The SAGE-specific `srange` call generates a uniformly spaced list of points.

Only the right-hand sides of the equations need to be specified, `desolve_odeint` "knows" that we are solving first-order DEs. The initial conditions are now a list of the y1,y2,... initial values, because the first time point is already specified in the `tps` list. The names of the unknown function variables need to be passed as the `dvars` parameter.

The solution is returned as a 2-dimensional NumPy array. The rows correspond to the "time points", the columns to the solutions. In our example there are two columns which we extract for plotting.



Because the `line` function expects a list of point coordinate pairs, we `zip` the abscissa points in `tps` ("X" coordinates) with the solution vectors ("Y" coordinates). This is important if you specified a non-uniformly spaced set of abscissa points.
The numerical solution looks very similar to the exact solution, which is reassuring ☺