Mathematica and Differential Equations

Review:

ReplaceAll: *expr* **/.** *rules* applies a rule or list of rules in an attempt to transform each subpart of an expression *expr*.

Example - **1+x/.x→a**
1+a

Symbolic solutions of differential equations can be computed using the DSolve function. The first argument in DSolve is an equation or list of equations. Equations are entered using Equal, and derivatives are entered using Derivative expressions. The second argument in DSolve gives a function or list of functions for which to solve, and the third argument gives the independent variable.

Here is an example showing the use of DSolve to solve a differential equation involving one function. The result is a list of solutions, and each solution is a list of rules. In this particular example there is only one solution, and that solution involves a rule for only one function, but this same form of result can also be used to represent results that involve more than one solution or rules for more than one function. The constant C[1] was generated by DSolve to represent the arbitrary constant in this solution.

```
DSolve[f'[x] == f[x], f[x], x]
{{f[x] → e^x C[1]}}
```

Initial conditions and boundary conditions can be included by including additional equations. Here is an example in which the differential equation **f''[x] + f[x] == 0** is solved with the initial conditions **f[0] == 1** and **f'[0] == 0**.

```
DSolve[{f''[x] + f[x] == 0, f[0] == 1, f'[0] == 0},
 f[x], x]
{{f[x] → Cos[x]}}
```

The independent variable can be omitted in the function specification in the second argument of DSolve, in which case the result is returned using Function expressions. Function expressions are used in *Mathematica* for representing pure functions. Pure functions are described in Section 2.2.5 of *The Mathematica Book*.

Here is an example in which the result from DSolve is returned using a Function expression. The only difference between this input and the previous input is the use of **f** rather than **f[x]** as the second argument in DSolve. The Function expression is displayed here using the notation **{{f→Function[{x},Cos[x]]}}**.

```
g[t_]=DSolve[{f''[x]+f[x]==0,f[0]==1,f'[0]==0},f,x]
{{f→Function[{x},Cos[x]]}}
```

A solution using Function expressions is often more convenient that the alternative because it avoids the need to make separate substitutions for the variable and for the function. To illustrate this difference, here are two examples.

```
DSolve[y'[x] + y[x] == 1, y[x], x]
```

$\{\{y[x] \to 1 + e^{-x} C[1]\}\}$

The rule applies only to `y[x]` itself, and not, for example, to objects like `y[0]` or `y'[x]`.

```
y[x] + 2y'[x] + y[0] /. %
```

$\{1 + e^{-x} C[1] + y[0] + 2 y'[x]\}$

If you ask `DSolve` to solve for `y`, it gives a rule for the object `y` on its own as a pure function.

```
DSolve[y'[x] + y[x] == 1, y, x]
```

$\{\{y \to \text{Function}[\{x\}, 1 + e^{-x} C[1]]\}\}$

Now the rule applies to all occurrences of `y`.

```
y[x] + 2y'[x] + y[0] /. %
```

$\{2 + C[1] - e^{-x} C[1]\}$

Substituting the solution into the original equation yields `True`.

```
y'[x] + y[x] == 1 /. %%
{True}
```

Rules returned by DSolve can be used just like rules from any other source. Here is an example in which the result from DSolve includes rules for two functions, both expressed using Function expressions.

```
sol =
 DSolve[{f'[x] == g[x] - 5 f[x],
   g'[x] == -4 f[x], f[0] == 0, g[0] == 1},
  {f, g}, x]
```

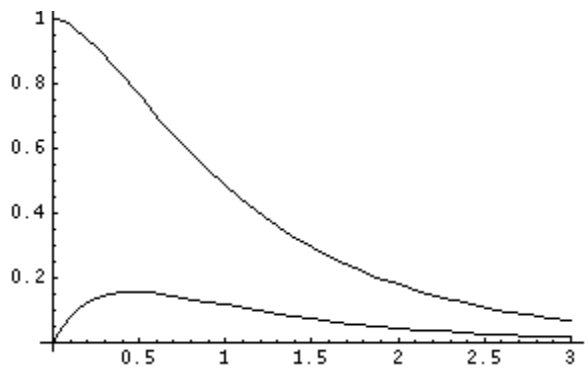
$$\left\{\left\{f \to \left(\frac{1}{3} e^{-4\#1} (-1 + e^{3\#1})\ \&\right),\ g \to \left(\frac{1}{3} e^{-4\#1} (-1 + 4 e^{3\#1})\ \&\right)\right\}\right\}$$

This gives the value of one of the solution function **g[x]** when **x** is **2.5**.

```
g[2.5] /. sol[[1]]
{0.109432}
```

Similar substitutions can be used to produce a plot showing both functions in the solution.

```
Plot[{f[x] /. sol, g[x] /. sol}, {x, 0, 3}]
```



- Graphics -

Exercise 1: Use *Mathematica* to find the analytic solution and graph for the I.V.P. $y' = 1 - t\,y$ with $y(0) = 1$ over $0 \le t \le 5$.

Numerical Solutions to Differential Equations

Numerical solutions of differential equations can be computed in *Mathematica* using the NDSolve function. This function provides access to algorithms for solving initial value problems, boundary value problems, and partial differential equations.

An *initial value problem* is a calculation specified by one or more ordinary differential equations together with values for the solution and any necessary derivatives at a single value of the independent variable. The values for the solution are referred to generically as *initial conditions*.

Here is an example showing the use of NDSolve to compute the solution to the differential equation `f'[x] == x` with initial condition `f[0] == 0`. The notation `f'[x]` is the short notation for Derivative[1][f][x], which denotes the first derivative of `f[x]` with respect to `x`. The solution is assigned as the value of `sol` for use in subsequent calculations.

```
sol = NDSolve[{f'[x] == x, f[0] == 0}, f,
  {x, 0, 10}]
{{f →
   InterpolatingFunction[{{0., 10.}}, <>]}}
```

The first argument in NDSolve is a list that gives the differential equation and the initial condition. The second argument is the function for which to solve. The third argument is a list that gives the independent variable and the starting and ending values for that variable.

The result is expressed using an InterpolatingFunction expression. In general, the result from NDSolve will be a list of solutions, each of which is a list of rules, just as in results from Solve. In this example there is only one solution, and that solution consists of a single rule. The right side of that rule is an InterpolatingFunction expression.

The InterpolatingFunction expression can be used much like any other function. For example, if the InterpolatingFunction expression returned by NDSolve is substituted for `f` in `f[5]`, the

result will be the solution to the differential equation when **x** is **5**. This substitution can be made by evaluating the following input:

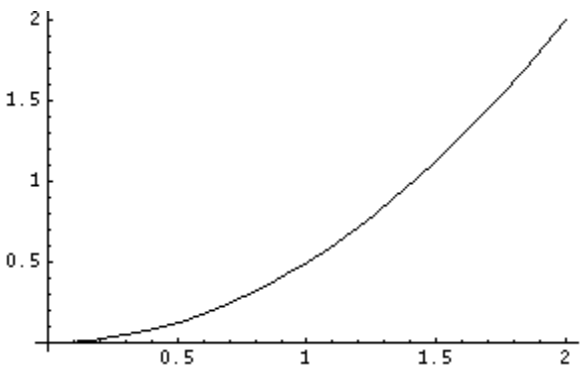
```
f[5] /. sol
{12.5}
```

This result is a list containing a number because the value of **sol** is a list that contains the solution. In this example there is only one solution, which can be selected using **sol[[1]]**. With this change the result of the substitution will be a number rather than a list containing a number.

```
f[5] /. sol[[1]]
12.5
```

The same substitution can be used to make a plot of the solution.

```
Plot[f[x] /. sol[[1]], {x, 0, 2}]
```



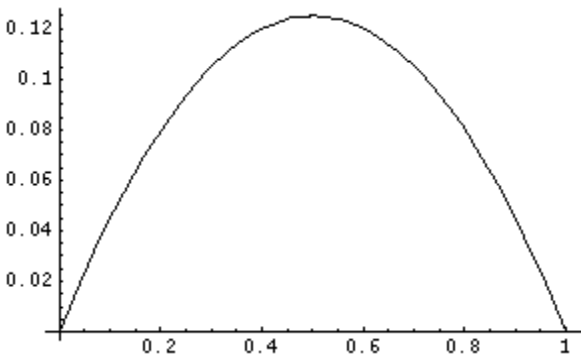

```
- Graphics -
```

Here is a more complicated initial value problem involving a system of three ordinary differential equations. This example also has two solutions, so the result is a list with two elements, each of which is a list of three rules.

```
sol =
 NDSolve[{f1'[x]^2 == f2[x], f2'[x] == f3[x],
   f3'[x] == f1[x], f1[0] == 1, f2[0] == 0,
   f3[0] == 0}, {f1, f2, f3}, {x, 0, 2}]
{{f1 → InterpolatingFunction[{{0., 2.}}, <>],
  f2 → InterpolatingFunction[{{0., 2.}}, <>],
  f3 →
   InterpolatingFunction[{{0., 2.}}, <>]},
 {f1 → InterpolatingFunction[{{0., 2.}}, <>],
  f2 → InterpolatingFunction[{{0., 2.}}, <>],
  f3 →
   InterpolatingFunction[{{0., 2.}}, <>]}}
```

An *boundary value problem* differs from an initial value problem in that values for the solution in a boundary value problem are specified at more than one value of the independent variable. Values for the solution in a boundary value problem are called *boundary conditions*. Here is a boundary value problem in which the value of the solution **f[x]** is specified by the boundary conditions **f[0] == 0** and **f[1] == 0**. The differential equation **f''[x] + 1 == 0** in this example could describe the height of a projectile, in which case the boundary conditions might indicate that the projectile should be on the ground when **x** is **0** and when **x** is **1**.

```
sol =
 NDSolve[{f''[x] + 1 == 0, f[0] == 0, f[1] == 0},
  f, {x, 0, 1}]
{{f → InterpolatingFunction[{{0., 1.}}, <>]}}
Plot[f[x] /. sol, {x, 0, 1}]
```



- Graphics -

Exercise 2: Solve the logistic growth model using the Mathematica command NDSolve.

n'[t]== r*n[t]*(1-n[t]/k),

letting k=1000, n[0]=100, tmin=0, and tmax=100.

Plot n[t] as above for the following values of r: -0.1, -1, 0.1, 1.
In which cases is n = K approached and in which cases is n = 0 approached?