WOLFRAM U

Scientific Computing with Mathematica

PART (2): ORDINARY DIFFERENTIAL EQUATIONS

NDSolve
NDSolveValue
AsymptoticDSolveValue
DSolveValue
DSolve

Outline

Classification of Ordinary Differential Equations (ODEs)

Symbolic Solution to ODEs

Numerical Solution to ODEs

Asymptotic Solution to ODEs

Classification of Differential Equations

- The solution method used by DSolve and the nature of the solutions depend heavily on the class of equation being solved.

Order

- The order of a differential equation is the order of the highest derivative in the equation.

Here is the general solution to a first-order ODE:

In[ ]:=
```
DSolve[(x^3- x)y'[x] == x,y[x],x]
```

Out[ ]= $\left\{\left\{y[x] \to c_1 + \frac{1}{2}\,\text{Log}[1-x] - \frac{1}{2}\,\text{Log}[1+x]\right\}\right\}$

Here is the general solution to a fourth-order ODE:

In[ ]:=
```
DSolve[y''''[x] + y''[x] == x,y[x],x]
```

Out[ ]= $\left\{\left\{y[x] \to \frac{x^3}{6} + c_3 + x\,c_4 - c_1\,\text{Cos}[x] - c_2\,\text{Sin}[x]\right\}\right\}$

Classification of Differential Equations

Linearity

- A differential equation is linear if the equation is of the first degree in y and its derivatives, and if the coefficients are functions of the independent variable.
- It's rare to find symbolic solutions to Nonlinear ODEs.

This is a nonlinear second-order ODE that represents the motion of a circular pendulum:

In[ ]:=
```
DSolve[y''[x] + Sin[y[x]] == 0, y[x], x]
```

Solve : Inverse functions are being used by Solve, so some solutions may not be found; use Reduce for complete solution information.

Out[ ]=
$$\left\{\left\{y[x] \to -2\,\text{JacobiAmplitude}\left[\frac{1}{2}\sqrt{(2+c_1)(x+c_2)^2}\,,\,\frac{4}{2+c_1}\right]\right\},\ \left\{y[x] \to 2\,\text{JacobiAmplitude}\left[\frac{1}{2}\sqrt{(2+c_1)(x+c_2)^2}\,,\,\frac{4}{2+c_1}\right]\right\}\right\}$$

Another Nonlinear first-order ODE:

In[ ]:=
```
DSolve[{y'[x] + y[x]^2 == 0, y[0]==1}, y[x], {x, 0, 1}]
```

Out[ ]=
$$\left\{\left\{y[x] \to \frac{1}{1+x}\right\}\right\}$$

Classification of Differential Equations

ODEs with Constant Coefficients

- When the coefficients of a linear ODE do not depend on x, the ODE is said to have constant coefficients.

This is an ODE with constant coefficients:

In[ ]:=
```
DSolve[2y''[x] + 5y'[x] + 6y[x] == 0,y[x],x]
```

Out[ ]=
$$\left\{\left\{y[x] \to e^{-5\,x/4}\,c_2\,\text{Cos}\left[\frac{\sqrt{23}\,x}{4}\right] + e^{-5\,x/4}\,c_1\,\text{Sin}\left[\frac{\sqrt{23}\,x}{4}\right]\right\}\right\}$$

This is an ODE with nonconstant coefficients:

In[ ]:=
```
DSolve[2y''[x] + y'[x] +(5x)*y[x] == 0,y[x],x]
```

Out[ ]=
$$\left\{\left\{y[x] \to e^{-x/4}\,\text{AiryAi}\left[-(-1)^{1/3}\left(\frac{2}{5}\right)^{2/3}\left(\frac{1}{16} - \frac{5\,x}{2}\right)\right]c_1 + e^{-x/4}\,\text{AiryBi}\left[-(-1)^{1/3}\left(\frac{2}{5}\right)^{2/3}\left(\frac{1}{16} - \frac{5\,x}{2}\right)\right]c_2\right\}\right\}$$

Classification of Differential Equations

Non-Homogenous ODEs

- When all terms contain y or derivatives of y and its right-hand side is zero, the equation is called homogeneous

In[ ]:=
```
DSolve[2y''[x] + 5y'[x] + 6y[x] == 0,y[x],x]
```

Out[ ]= $\left\{\left\{y[x] \to e^{-5\,x/4}\,c_2\,\text{Cos}\left[\frac{\sqrt{23}\,x}{4}\right] + e^{-5\,x/4}\,c_1\,\text{Sin}\left[\frac{\sqrt{23}\,x}{4}\right]\right\}\right\}$

- Adding a function of the independent variable makes the equation inhomogeneous.
- The general solution to an inhomogeneous equation with constant coefficients is obtained by adding a particular integral to the solution to the corresponding homogeneous equation.

In[ ]:=
```
DSolve[2y''[x] + 5y'[x] + 6y[x] == x,y[x],x]//FullSimplify
```

Out[ ]= $\left\{\left\{y[x] \to \frac{1}{36} \times (-5 + 6\,x) + e^{-5\,x/4}\left(c_2\,\text{Cos}\left[\frac{\sqrt{23}\,x}{4}\right] + c_1\,\text{Sin}\left[\frac{\sqrt{23}\,x}{4}\right]\right)\right\}\right\}$

ODEs of Exact Solution

- The study of exact solutions has the longest history, dating back to the period just after the discovery of calculus by Sir Isaac Newton and Gottfried Wilhelm von Leibniz. The following table introduces the types of equations that can be solved by DSolve.

name of equation	*general form*	*date of discovery*	*mathematician*
Separable	$y'(x) = f(x)\,g(y)$	1691	G. Leibniz
Homogeneous	$y'(x) = f\left(\frac{x}{y(x)}\right)$	1691	G. Leibniz
Linear first-order ODE	$y'(x) + P(x)\,y(x) = Q(x)$	1694	G. Leibniz
Bernoulli	$y'(x) + P(x)\,y(x) = Q(x)$	1695	James Bernoulli
Riccati	$y'(x) = f(x) + g(x)\,y(x) + h(x)\,y(x)^2$	1724	Count Riccati
Exact first-order ODE	$M\,dx + N\,dy = 0$ with $\frac{\partial M}{\partial y} = \frac{\partial N}{\partial x}$	1734	L. Euler
Clairaut	$y(x) = x\,y'(x) + f(y'(x))$	1734	A-C. Clairaut
Linear with constant coefficients	$y^{(n)}(x) + a_{n-1}\,y^{(n-1)}(x) + \ldots + a_0\,y(x) = P(x)$ with a_1 constant	1743	L. Euler
Hypergeometric	$x(1-x)\,y''(x) + (c - (a+b+1)\,x)\,y'(x) - a\,b\,y(x) = 0$	1769	L. Euler
Legendre	$(1-x^2)\,y''(x) - 2\,x\,y'(x) + n(n+1)\,y(x) = 0$	1785	M. Legendre
Bessel	$x^2\,y''(x) + x\,y'(x) + (x^2 - n^2)\,y(x) = 0$	1824	F. Bessel
Mathieu	$y''(x) + (a - 2\,q\cos(2\,x))\,y(x) = 0$	1868	E. Mathieu
Abel	$y'(x) = f(x) + g(x)\,y(x) + h(x)\,y(x)^2 + k(x)\,y(x)^3$	1834	N. H. Abel
Chini	$y'(x) = f(x) + g(x)\,y(x) + h(x)\,y(x)^n$	1924	M. Chini

- Examples of ODEs belonging to each of these types are given in other online tutorials

DSolve/DSolveValue

Basic Syntax

In[ ]:= `? DSolve`

Out[ ]=

Symbol

DSolve [*eqn*, *u*, *x*] solves a differential equation for the function *u*, with independent variable *x*.

DSolve [*eqn*, *u*, {*x*, x_{min}, x_{max}}] solves a differential equation for *x* between x_{min} and x_{max}.

DSolve [{eqn_1, eqn_2, …}, {u_1, u_2, …}, …] solves a list of differential equations .

DSolve [*eqn*, *u*, {x_1, x_2, …}] solves a partial differential equation .

DSolve [*eqn*, *u*, {x_1, x_2, …} ∈ Ω] solves the partial differential equation *eqn* over the region Ω.

- DSolve returns results as lists of rules:

In[ ]:=
```
DSolve[{y'[x] == y[x]},y[x],x]
```

Out[ ]= {{y[x] → e^x C_1}}

- You can pick out a specific solution by using ./ (ReplaceAll).:

In[ ]:=
```
y[x] /. DSolve[{y'[x] == y[x]},y[x],x]
```

Out[ ]= {e^x C_1}

- DSolveValue is the same as DSolve, but returns the solution expression (not as a rule)

In[ ]:=
```
DSolveValue[{y'[x] == y[x]},y[x],x]
```

Out[ ]= e^x C_1

General Solution and Initial/Boundary Conditions

- A general solution contains arbitrary parameters C_i that can be varied to produce particular solutions for the equation:

In[ ]:=
```
DSolveValue [{y''[x] == y[x]},y[x],x]
```

Out[ ]= $e^{x}\, C_1 + e^{-x}\, C_2$

- When an adequate number of initial conditions are specified, DSolve returns particular solutions to the given equations:

In[ ]:=
```
sol = DSolveValue [{y''[x] == -y[x],y[0] ==1, y'[0] == 0},y[x],x]
```

Out[ ]= Cos[x]

Visualizing ODE Solutions

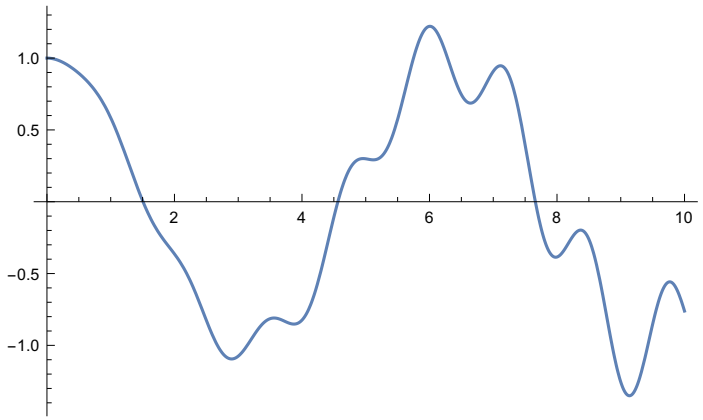
In[ ]:=
```
sol = DSolveValue [{y''[x] == x*Sin[5x]-y[x],y[0] ==1, y'[0] == 0},y[x],x]//FullSimplify
```

Out[ ]= $\frac{1}{288} \times (293 \text{ Cos}[x] - 5 \text{ Cos}[5 x] - 12 x \text{ Sin}[5 x])$

- You can plot the solution with Plot or its variants:

In[ ]:=
```
Plot[sol,{x,0,10}]
```

Out[ ]=



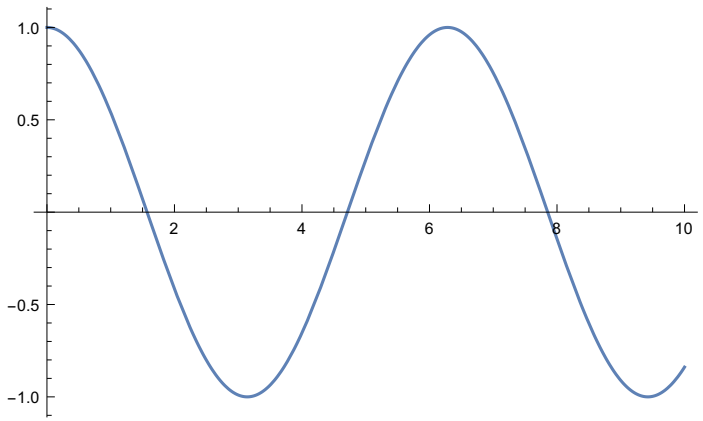
- When using DSolve, you have to use /. to extract the solution expression:

In[ ]:=
```
sol = DSolve [{y''[x] == -y[x],y[0] ==1, y'[0] == 0},y[x],x]
```

Out[ ]= {{y[x] → Cos[x]}}

In[ ]:=
```
Plot[y[x]/. sol,{x,0,10}]
```

Out[ ]=



Numerical Differential Equation Solving in Mathematica

- The Mathematica function NDSolve is a general numerical differential equation solver.

Syntax

In[ ]:= `? NDSolve`

Out[ ]=

Symbol

NDSolve [*eqns*, *u*, {*x*, x_{min}, x_{max}}] finds a numerical solution to the ordinary differential equations *eqns* for the function *u* with the independent variable *x* in the range x_{min} to x_{max}.

NDSolve [*eqns*, *u*, {*x*, x_{min}, x_{max}}, {*y*, y_{min}, y_{max}}] solves the partial differential equations *eqns* over a rectangular region.

NDSolve [*eqns*, *u*, {*x*, *y*} ∈ Ω] solves the partial differential equations *eqns* over the region Ω.

NDSolve [*eqns*, *u*, {*t*, t_{min}, t_{max}}, {*x*, *y*} ∈ Ω] solves the time-dependent partial differential equations *eqns* over the region Ω.

NDSolve [*eqns*, {u_1, u_2, ...}, ...] solves for the functions u_i.

In[ ]:=
```
sol = NDSolve[{y''[x] == -y[x],y[0] ==1, y'[0] == 0},y[x],{x,0,1}]
```

Out[ ]= {{y[x] → InterpolatingFunction [Domain : {{0., 1.}} Output : scalar][x]}}

- NDSolve represents solutions for the functions x i as InterpolatingFunction objects.
- In general, NDSolve finds solutions iteratively. It starts at a particular value of t_min, then takes a sequence of steps, trying eventually to cover the whole range t_min to t_max .
- NDSolve has to be given appropriate initial or boundary conditions and the range of solution.

- The initial or boundary conditions you give must be sufficient to determine a unique solution.

NDSolve

Initial Value Problems

- In a typical case, if you have differential equations with up to nth derivatives, then you need to either give initial conditions for up to (n - 1)th derivatives, or give boundary conditions at n points.

This solves an initial value problem for a second-order equation, which requires two conditions, and are given at t == 0 .

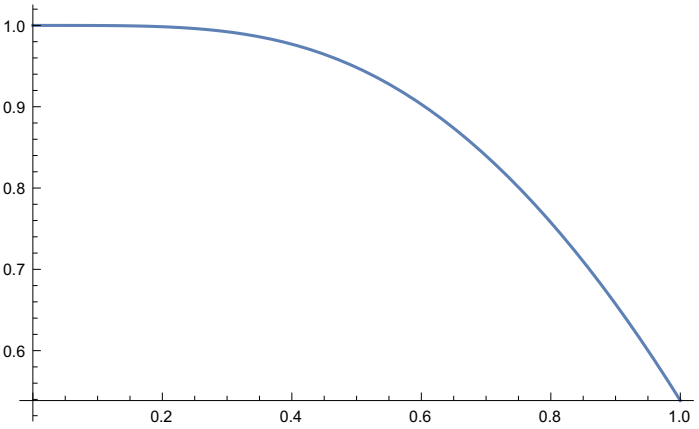
In[ ]:=
```
sol = NDSolveValue [{y''[x] == -y[x]^2 + Log[y[x]] + Cos[5x*y[x]],y[0] ==1, y'[0] == 0},y[x],{x,0,1}]
```

Out[ ]= InterpolatingFunction [Domain : {{0., 1.}} Output : scalar][x]

This plots the solution obtained:

In[ ]:=
```
Plot[sol, {x, 0, 1}]
```

Out[ ]=



NDSolve

Boundary Value Problems

- In a typical case, if you have differential equations with up to nth derivatives, then you need to either give initial conditions for up to (n - 1)th derivatives, or give boundary conditions at n points.

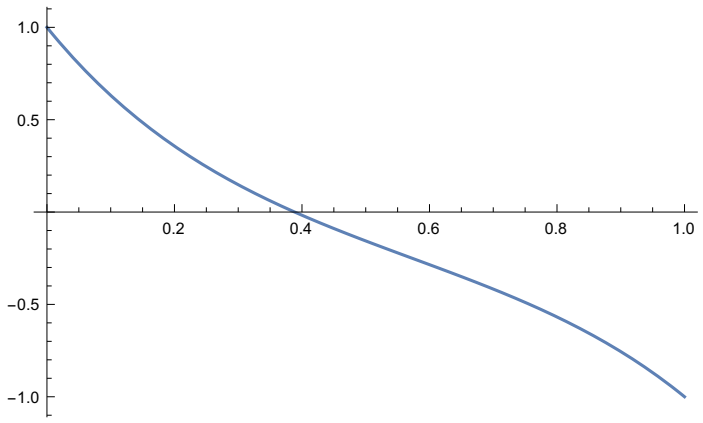
This plots the solution obtained:

Here is a simple boundary value problem.

In[ ]:=
```
sol = NDSolveValue [{y''[x] == 10y[x] + 3Cos[x],y[0] ==1, y[1] == -1},y[x],{x,0,1}]
Plot[sol,{x,0,1}]
```

Out[ ]= InterpolatingFunction [Domain : {{0., 1.}} Output : scalar][x]

Out[ ]=



NDSolve

Coupled Systems of ODEs

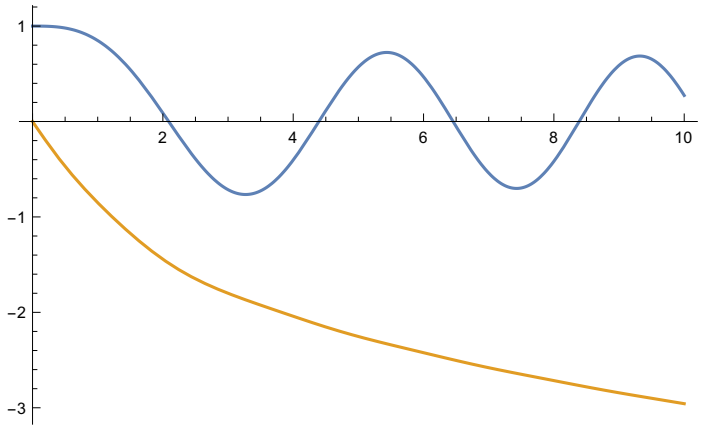
- You can use NDSolve to solve systems of coupled differential equations as long as each variable has the appropriate number of conditions.

In[ ]:=
```
eq1 = x''[t] == x[t]×y[t]
eq2 = y'[t] == -1/(x[t]^2+ y[t]^2)
bc1 = x[0] == 1;
bc2 = x'[0] == 0;
bc3 = y[0] == 0;
{X,Y} = NDSolveValue [{eq1,eq2,bc1,bc2,bc3},{x[t],y[t]}, {t,0,10}];
Plot[{X,Y},{t,0,10}]
```

Out[ ]= $x''[t] == x[t] \times y[t]$

Out[ ]= $y'[t] == -\frac{1}{x[t]^2 + y[t]^2}$

Out[ ]=



NDSolve

Differential-algebraic equations (DAEs)

- Mixed systems of differential and algebraic equations are called differential-algebraic equations (DAEs).

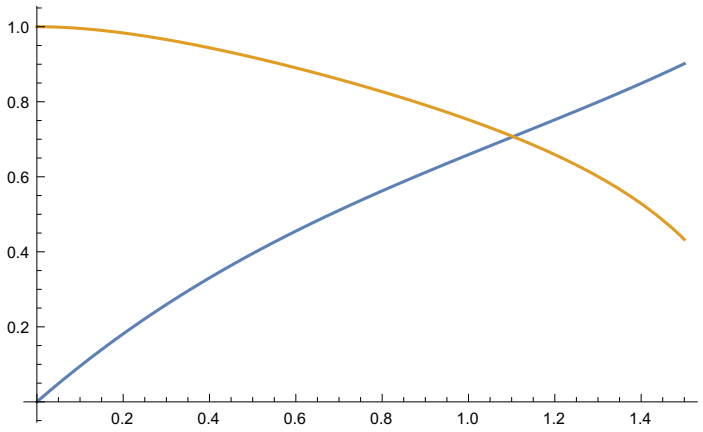
Here is a simple DAE.

In[ ]:=
```
eq1 = x''[t] + y[t] == x[t]
eq2 = x[t]^2+ y[t]^2 == 1
bc1 = x[0] == 0;
bc2 = x'[0] == 1;
{X,Y} = NDSolveValue [{eq1,eq2,bc1,bc2},{x[t],y[t]}, {t,0,1.5}];
Plot[{X,Y},{t,0,1.5}]
```

Out[ ]= $y[t] + x''[t] == x[t]$

Out[ ]= $x[t]^2 + y[t]^2 == 1$

Out[ ]=



NDSolve

Choosing a numerical method

- NDSolve has several different methods built in for computing solutions.
- With the default setting Method -> Automatic, NDSolve will choose a method which should be appropriate for the differential equations.
- NDSolve will choose the write method and step size for you

▪ Possible explicit time integration settings for the Method option include:

"Adams"	predictor-corrector Adams method with orders 1 through 12
"BDF"	implicit backward differentiation formulas with orders 1 through 5
"ExplicitRungeKutta"	adaptive embedded pairs of 2(1) through 9(8) Runge–Kutta methods
"ImplicitRungeKutta"	families of arbitrary-order implicit Runge–Kutta methods
"SymplecticPartitionedRungeKutta"	interleaved Runge–Kutta methods for separable Hamiltonian systems

In[ ]:=
```
sol = NDSolveValue [{y''[x] == 10y[x] + 3Cos[x],y[0] ==1, y[1] == -1},y[x],{x,0,1},Method→"ImplicitRu
sol = NDSolveValue [{y''[x] == 10y[x] + 3Cos[x],y[0] ==1, y[1] == -1},y[x],{x,0,1},Method->"Adams "]
```

Out[ ]= InterpolatingFunction [ [x]

Out[ ]= InterpolatingFunction [ [x]

NDSolve

Choosing a numerical method

- NDSolve has several different methods built in for computing solutions.
- With the default setting Method -> Automatic, NDSolve will choose a method which should be appropriate for the differential equations.
- NDSolve will choose the write method and step size for you

▪ Possible explicit time integration settings for the Method option include:

"Adams"	predictor-corrector Adams method with orders 1 through 12
"BDF"	implicit backward differentiation formulas with orders 1 through 5
"ExplicitRungeKutta"	adaptive embedded pairs of 2(1) through 9(8) Runge–Kutta methods
"ImplicitRungeKutta"	families of arbitrary-order implicit Runge–Kutta methods
"SymplecticPartitionedRungeKutta"	interleaved Runge–Kutta methods for separable Hamiltonian systems

In[ ]:=
```
sol = NDSolveValue [{y''[x] == 10y[x] + 3Cos[x],y[0] ==1, y[1] == -1},y[x],{x,0,1},Method→"ImplicitRu
sol = NDSolveValue [{y''[x] == 10y[x] + 3Cos[x],y[0] ==1, y[1] == -1},y[x],{x,0,1},Method->"Adams "]
```

Out[ ]= InterpolatingFunction [ [x]

Out[ ]= InterpolatingFunction [ [x]

NDSolve

Choosing a numerical method

- NDSolve has several different methods built in for computing solutions.
- With the default setting Method -> Automatic, NDSolve will choose a method which should be appropriate for the differential equations.
- NDSolve will choose the write method and step size for you

▪ Possible explicit time integration settings for the Method option include:

"Adams"	predictor-corrector Adams method with orders 1 through 12
"BDF"	implicit backward differentiation formulas with orders 1 through 5
"ExplicitRungeKutta"	adaptive embedded pairs of 2(1) through 9(8) Runge–Kutta methods
"ImplicitRungeKutta"	families of arbitrary-order implicit Runge–Kutta methods
"SymplecticPartitionedRungeKutta"	interleaved Runge–Kutta methods for separable Hamiltonian systems

In[ ]:=
```
sol = NDSolveValue [{y''[x] == 10y[x] + 3Cos[x],y[0] ==1, y[1] == -1},y[x],{x,0,1},Method→"ImplicitRu
sol = NDSolveValue [{y''[x] == 10y[x] + 3Cos[x],y[0] ==1, y[1] == -1},y[x],{x,0,1},Method->"Adams "]
```

Out[ ]= InterpolatingFunction [ [x]

Out[ ]= InterpolatingFunction [ [x]

NDSolve

Step Size

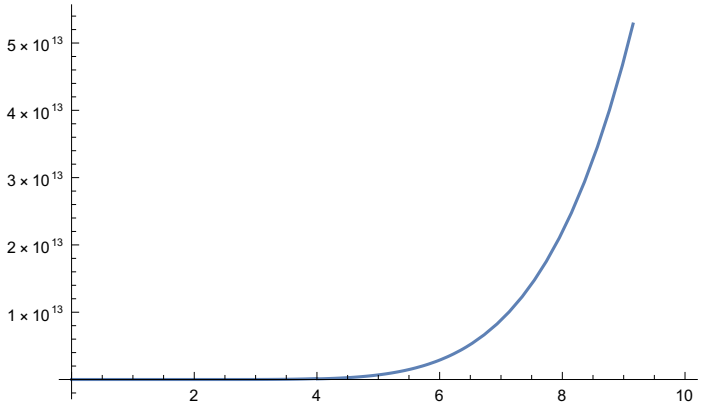
- NDSolve will choose a step size the optimizes speed and accuracy at the same time.
- The option MaxStepSize fixes the step size.

In[ ]:=
```
sol = NDSolveValue [{-y''[x] ==1Sin[x]+x y[x],y[0] ==1, y[1] == -1},y[x],{x,0,2},MaxStepSize →.00001 ]
Plot[sol,{x,0,10}]
```

Out[ ]= InterpolatingFunction [Domain : {{0., 2.}} Output : scalar][x]

Data not in notebook ; Store now »

Out[ ]=

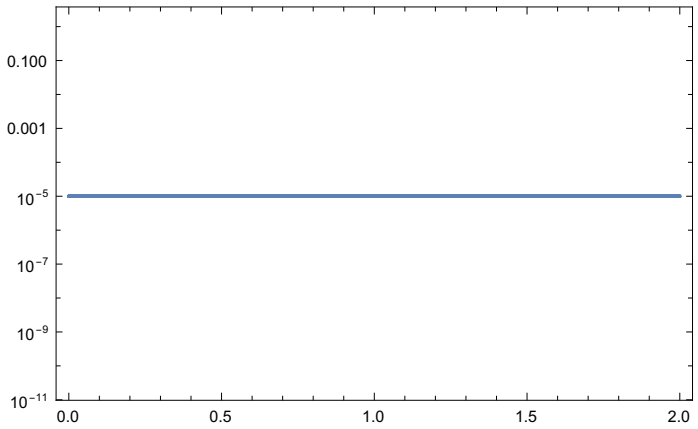


The step-size can be visualized with DifferentialEquations`NDSolveUtilties`

In[ ]:=

```
Needs["DifferentialEquations`NDSolveUtilities`  "]
StepDataPlot[sol]
```

Out[ ]=



NDSolve

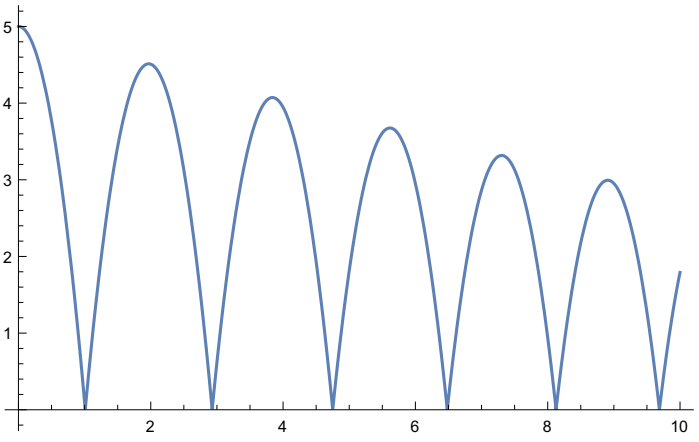
WhenEvent

- WhenEvent specifies an *action* that occurs when the *event* triggers it for equations in NDSolve and related functions.

Simulate a bouncing ball that retains 95% of its velocity in each bounce:

In[ ]:=
```
NDSolve[{y''[t]==-9.81,y[0]==5,y'[0]==0,WhenEvent[y[t]==0,y'[t]->-0.95 y'[t]]},y,{t,0,10}];
Plot[y[t]/.%,{t,0,10}]
```

Out[ ]=



Asymptotic Differential Equation Solving in Mathematica

AsymptoticDSolveValue

- Asymptotic approximations to differential equations are also known as asymptotic expansions, perturbation solutions, regular perturbations and singular perturbations
- Asymptotic approximations are typically used to solve problems for which no exact solution can be found or to get simpler answers for computation, comparison and interpretation.

Find a series solution for a differential equation :

In[ ]:=
```
order = 3;
sol1 = AsymptoticDSolveValue [{y''[x]+Sin[x]y[x]==0,y[0]==1,y'[0]==0},y[x],{x,0,order}]
sol2 = NDSolveValue [{y''[x]+Sin[x]y[x]==0,y[0]==1,y'[0]==0},y[x],{x,0,3}]
```

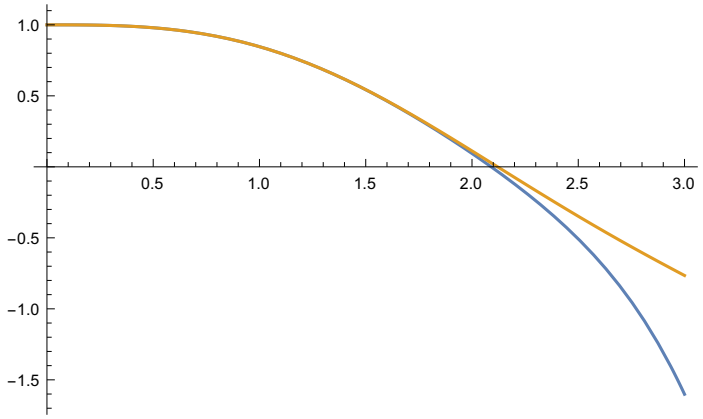
Out[ ]= $1 - \frac{x^3}{6}$

Out[ ]= InterpolatingFunction [][x]

The

In[ ]:=
```
Plot[{sol1,sol2},{x,0,3}]
```

Out[ ]=



Asymptotic Differential Equation Solving in Mathematica

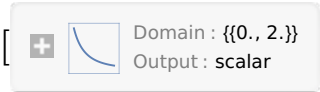
Perturbation problems

Find an asymptotic expansion for a perturbation problem:

In[ ]:=
```
order = 3;
sol1 = AsymptoticDSolveValue [{ y'[x]+y[x]+ϵ y[x]^2==0, y[0]==1},y[x],x, {ϵ,0,order}]
sol2 = NDSolveValue [{ y'[x]+y[x]+y[x]^2==0, y[0]==1},y[x], {x,0,2}]

Plot[{sol2 ,Evaluate [Table [sol1 ,{ϵ,{1}}]]},{x,0,1}]
```

Out[ ]= $e^{-x} - e^{-2x}(-1+e^{x})\epsilon + e^{-3x}(-1+e^{x})^2\epsilon^2 - e^{-4x}(-1+e^{x})^3\epsilon^3$

Out[ ]= InterpolatingFunction [ [x]

Out[ ]=