Lecture 21: Higher-Order Ordinary Differential Equations

Reading:
Kreyszig Sections: 2.1, 2.2 (pages45–52, 53–58)

Higher-Order Equations: Background

For first-order ordinary differential equations (ODEs), $F(y'(x), y(x), x)$, one value $y(x_o)$ was needed to specify a particular solution. Recall the example in Lecture 19 of a first-order differencing scheme: at each iteration the function grew proportionally to its current size. In the limit of very small forward differences, the scheme converged to exponential growth.

Now consider a situation in which function's current rate of growth increases proportionally to two terms: its current rate of growth and its size.

$$\text{Change in Value's Rate of Change} + \alpha\ (\text{the Value}) + \beta\ (\text{Value's Rate of Change}) = 0$$

To calculate a forward differencing scheme for this case, let Δ be the forward-differencing increment.

$$\left(\frac{\frac{F_{i+2}-F_{i+1}}{\Delta} - \frac{F_{i+1}-F_i}{\Delta}}{\Delta} \right) + \alpha F_i + \beta \left(\frac{F_{i+1} - F_i}{\Delta} \right) = 0$$

and then solve for the "next increment" F_{i+2} if F_{i+1} and F_i are known.

This indicates that, for second-order equations, two independent values are needed to generate the 'solution trajectory.'

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

A Second-Order Forward Differencing Example

A second order differencing formula is developed for second-order equations. The current value of the right-hand side is used, and therefore this is an explicit method.

```
ChangePerΔ[F_, i_, Δ_] :=
 (F[i + 1] - F[i]) / Δ
```
1

```
ChangeofChangeperΔ[
  F_, i_, Δ_] :=
 Simplify[(ChangePerΔ[
      F, i + 1, Δ] / Δ -
    ChangePerΔ[F, i, Δ])]
```
2

```
DifferenceRelation =
 ChangeofChangeperΔ[
   F, i, Δ] ==
 -β ChangePerΔ[F, i, Δ] -
  α F[i]
```
3

```
ForDiffSol = Collect[
  Solve[DifferenceRelation,
   F[i + 2]], Δ]
```
4

Replace to find the form of the solution

```
ForDiffSolV2 =
 ForDiffSol /. i → j - 2
```
5

```
{{F[j] →
    -α Δ^2 F[-2 + j] + F[-1 + j] +
     Δ (-F[-2 + j] + β F[-2 + j] +
        F[-1 + j] - β F[-1 + j])}}
```

1: *ChangeperΔ* is an example of a first-order finite difference approximation to the first derivative.

2: *ChangeofChangeperΔ* is the second order difference operation, it is obtained by applying the first-order difference operator twice. Note that two sequential values appear and that the differencing operator is proportional to $1/\Delta^2$. This is a general feature of high-order difference operators—with higher difference operations goes, the number of surrounding points required to evaluate the difference gets larger and larger (e.g., for the second order difference, function values are needed at three different i compared to two different i for the first-order case.

3: For a particular case of $d^2y/dx^2 = -\alpha dy/dx - \beta y$, the two difference operators replace the derivatives and a *difference relation* can be derived as a function of parameters α and β. We assign the equation representing the difference relation to *DifferenceRelation* .

4: The difference operator is derived by solving the difference relation for F_{i+2}—it will depend on the immediate last value F_{i+1} and that value's antecedent F_i. Therefore, any value—*including the first one calculated*—requires *two values* to be specified.

5: Typically, the current j–value is expressed in terms of the $(j-1)$ and $(j-2)$–values. This form is generated by the replacement $i \rightarrow j-2$.

3.016 Home
◀◀ ◀ ▶ ▶▶
Full Screen
Close
Quit

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

A Second-Order Forward Differencing Example

Iteration sequences are produced using the explicit differencing operators produced above. A particular example of a second-order method with constant coefficients is presented.

```
GrowList[ValuesList_List,
  Δ_ , α_ , β_] :=
 Module[{Minus1 =
    ValuesList[[-1, 2]],
   Minus2 = ValuesList[[-2,
     2]], LastX =
    ValuesList[[-1, 1]]},
  Append[ValuesList,
   {LastX + Δ,
    2*Minus1 - Minus2 +
     Δ*(β*(Minus2 -
           Minus1) -
        α*Δ*Minus2)}]]
```

1

```
InitVals =
  {{0, 1}, {.001, 1}};
```

2

```
result = GrowList[
  InitVals, .001, 1, .1]
```

3

```
result = GrowList[
  result, .001, 1, .1]
```

4

```
NestWhile[
 GrowList[#, .001, 1, .1] &,
 InitVals,
 (Last[#][[1]] < 0.02) &]
```

5

1: The difference operator is incorporated in *GrowList* : a function that grows a list (input as `ValuesList`) using a difference Δ and parameters α and β. The two previous values in the list become *localized variables* in a `Module` function. The `Module` returns a new list that is created using `Append` to place the current value at end of the input list.

2: Because two values are required, a list of size two must be provided. These values could represent the initial value and the initial value of the derivative.

3: Here is an example of using *GrowList* once.

4: Using `NestWhile` the list can be grown iteratively until the first value (i.e., the current value of $x_i = x_{i-1} + \Delta$) reaches are specified value (here 0.02).

3.016 Home

Full Screen

Close

Quit

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

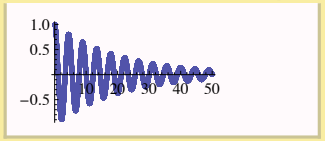
Visualization of Second-Order Forward Differencing

A second order differencing formula is developed for the case of constant growth and acceleration coefficients.

Visualize results out to x=50 (takes about 24 seconds to visualize)

```
ListPlot[
 NestWhile[GrowList[#,
    .001, 2, .1] &, InitVals,
  (Last[#][[1]] < 50) &]]
```

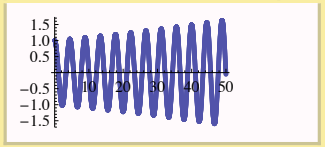
1



Change parameters for Growth Function (this example shows that the numerical solution does not converge to the accurate solution):

```
ListPlot[NestWhile[
  GrowList[#, 0.01, 2, 0] &,
  InitVals,
  (Last[#][[1]] < 50) &]]
```

2



1: `ListPlot` visualizes the results for growth constants $\alpha = 2$ and $\beta = 0.1$. `NestWhile` operates on a pure function constructed from *GrowList* until $x = 50$. The solution is oscillatory with fixed frequency, but with exponentially decreasing amplitude. This behavior is correct, although we have not analyzed the numerical stability or the accuracy of this method.

2: Here is another example with $\beta = 0$. With a small enough time step, this should mimic a harmonic oscillator. The increasing amplitude indicates a numerical inaccuracy at this time-step size.

3.016 Home
Full Screen
Close
Quit

Linear Differential Equations; Superposition in the Homogeneous Case

A linear differential equation is one for which the function and its derivatives are each linear—that is they appear in distinct terms and only to the first power. In the case of a homogeneous linear differential equation, the solutions are *superposable*. In other words, sums of solutions and their multiples are also solutions.

Therefore, a linear heterogeneous ordinary differential equation can be written as a product of general functions of the dependent variable and the derivatives for the n-order linear case:

$$\begin{aligned} 0 &= f_0(x) + f_1(x)\frac{dy}{dx} + f_2(x)\frac{d^2y}{dx^2} + \cdots + f_n(x)\frac{d^ny}{dx^n} \\ &= (f_0(x), f_1(x), f_2(x), \ldots, f_n(x)) \cdot \left(1, \frac{dy}{dx}, \frac{d^2y}{dx^2}, \ldots, \frac{d^ny}{dx^n}\right) \\ &= \vec{f}(x) \cdot \vec{D_n}y \end{aligned} \tag{21-1}$$

The homogeneous n^{th}-order linear ordinary differential equation is defined by $f_0(x) = 0$ in Eq. 21-1:

$$\begin{aligned} 0 &= f_1(x)\frac{dy}{dx} + f_2(x)\frac{d^2y}{dx^2} + \cdots + f_n(x)\frac{d^ny}{dx^n} \\ &= (0, f_1(x), f_2(x), \cdots, f_n(x)) \cdot \left(1, \frac{dy}{dx}, \frac{d^2y}{dx^2}, \ldots, \frac{d^ny}{dx^n}\right) \\ &= \vec{f_{hom}}(x) \cdot \vec{D_n}y \end{aligned} \tag{21-2}$$

Equation 21-1 can always be multiplied by $1/f_n(x)$ to generate the general form:

$$\begin{aligned} 0 &= F_0(x) + F_1(x)\frac{dy}{dx} + F_2(x)\frac{d^2y}{dx^2} + \cdots + \frac{d^ny}{dx^n} \\ &= (F_0(x), F_1(x), F_2(x), \ldots, 1)) \cdot (1, \frac{dy}{dx}, \frac{d^2y}{dx^2}, \ldots, \frac{d^ny}{dx^n}) \\ &= \vec{F}(x) \cdot \vec{D_n}y \end{aligned} \tag{21-3}$$

3.016 Home
Full Screen
Close
Quit

For the second-order linear ODE, the heterogeneous form can always be written as:

$$\frac{d^2y}{dx^2} + p(x)\frac{dy}{dx} + q(x)y = r(x) \tag{21-4}$$

and the homogeneous second-order linear ODE is:

$$\frac{d^2y}{dx^2} + p(x)\frac{dy}{dx} + q(x)y = 0 \tag{21-5}$$

Basis Solutions for the homogeneous second-order linear ODE

Because two values must be specified for each solution to a second order equation—the solution can be broken into two basic parts, each deriving from a different constant. These two independent solutions form a *basis pair* for any other solution to the homogeneous second-order linear ODE that derives from any other pair of specified values.

The idea is the following: suppose the solution to Eq. 21-5 is found the particular case of specified parameters $y(x = x_0) = A_0$ and $y(x = x_1) = A_1$, the solution $y(x; A_0, A_1)$ can be written as the sum of solutions to two *other problems*.

$$y(x; A_0, A_1) = y(x, A_0, 0) + y(x, 0, A_1) = y_1(x) + y_2(x) \tag{21-6}$$

where

$$\begin{aligned} y(x_0, A_0, 0) = A_0 \quad &\text{and} \quad y(x_1, A_0, 0) = 0 \\ y(x_0, 0, A_1) = 0 \quad &\text{and} \quad y(x_1, 0, A_1) = A_1 \end{aligned} \tag{21-7}$$

from these two solutions, any others can be generated.

The two arbitrary integration constants can be included in the definition of the general solution:

$$\begin{aligned} y(x) &= C_1 y_1(x) + C_1 y_2(x) \\ &= (C_1, C_2) \cdot (y_1, y_2) \end{aligned} \tag{21-8}$$

Second Order ODEs with Constant Coefficients

The most simple case—but one that results from models of many physical phenomena—is that functions in the homogeneous second-order linear ODE (Eq. 21-5) are constants:

$$a\frac{d^2y}{dx^2} + b\frac{dy}{dx} + cy = 0 \tag{21-9}$$

If two independent solutions can be obtained, then any solution can be formed from this basis pair.

Surmising solutions seems a sensible strategy, certainly for shrewd solution seekers. Suppose the solution is of the form $y(x) = \exp(\lambda x)$ and put it into Eq. 21-9:

$$(a\lambda^2 + b\lambda + c)e^{\lambda x} = 0 \tag{21-10}$$

which has solutions when and only when the quadratic equation $a\lambda^2 + \lambda x + c = 0$ has solutions for λ.

Because two solutions are needed and because the quadratic equation yields two solutions:

$$\begin{aligned} \lambda_+ &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} \\ \lambda_- &= \frac{-b - \sqrt{b^2 - 4ac}}{2a} \end{aligned} \tag{21-11}$$

or by removing the redundant coefficient by diving through by a:

$$\begin{aligned} \lambda_+ &= \frac{-\beta}{2} + \sqrt{(\frac{\beta}{2})^2 - \gamma} \\ \lambda_- &= \frac{-\beta}{2} - \sqrt{(\frac{\beta}{2})^2 - \gamma} \end{aligned} \tag{21-12}$$

where $\beta \equiv b/a$ and $\gamma \equiv c/a$.

Therefore, any solution to Eq. 21-9 can be written as

$$y(x) = C_+ e^{\lambda_+ x} + C_- e^{\lambda_- x} \tag{21-13}$$

This solution recreated with a slightly different method in the following MATHEMATICA® example.

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

Deriving the Solutions to the Homogeneous Linear Second Order ODE with Constant Coefficients

Even though MATHEMATICA® is able to determine solutions to linear second-order ODEs with constant coefficients directly, it is still instructive to use MATHEMATICA® to derive these solutions.

```
TheODE[function_, var_] :=
 D[function[var],
   {var, 2}] +
  β D[function[var], var] +
  γ function[var]
```
1

```
TheODE[y, x]
```
2

Guess a solution and substitute it into the left-hand side of the ODE:

```
TheGuess[x_] := Exp[λ x]
```
3

```
TheODE[TheGuess, x]
```
4

This will be a solution when the resulting quadratic expression in λ is equal to 0:

```
λSolution = Solve[
  TheODE[TheGuess, x] == 0,
  λ]
```
5

```
{λMinus, λPlus} =
 λ /. λSolution
```
6

```
GenSol[x_] :=
 C[LPlus] Exp[λPlus x] +
  C[LMinus] Exp[λMinus x]
```
7

```
TheODE[GenSol, z]
```
8

```
Simplify[TheODE[GenSol, z]]
```
9

1–2: *TheODE* represents the left-hand side of any second-order ODE with constant coefficients—it is the differential representation of the second-order differencing method that was developed above. *TheODE* an argument for the name of the function (i.e., y) and the dependent variable (i.e., x in $y(x)$).

3: This will serve as a 'guess' of a solution—if we can find λ(s) that satisfy the ODE, then the solution(s) are determined.

4: The guess is inserted as the first argument to *TheODE* . The property of the exponential function, $de^{\alpha x}/dx = d(\text{guess})/dx = \alpha e^{\alpha x} = \alpha\text{guess}$ will permit factoring of the 'guessed' solution.

5: Using `Solve` with the guess inserted into *TheODE* will determine solution conditions on λ—this will be a quadratic equation in λ. The quadratic equation's solution ensures that, if the solution is complex, the two λ are complex conjugates.

6: By extraction the solution from the rules returned from `Solve`, assignments can be made to the two possible λ.

7: This is the form of the general solution in terms of two arbitrary constants.

8: This should show that the general solution always satisfies the ODE.

3.016 Home
◀◀ ◀ ▶ ▶▶
Full Screen
Close
Quit

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

Characterizing the Solution Behavior for the Second-Order ODE with Constant Coefficients

Because the fundamental solution depend on only two parameters β and γ, the behavior (i.e., whether $\Re\lambda \overset{>}{<} 0$ and $\Im\lambda \overset{?}{=} 0$) of all solutions can be visualized in the γ-β plane.

```
RealsCond =
 Reduce[λPlus ∈ Reals &&
   λMinus ∈ Reals,
  {β, γ}, Reals]
```
1

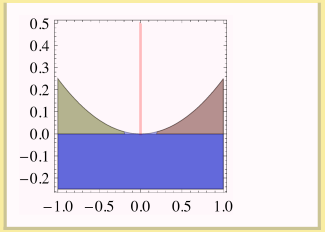
The Complex and Real Domains A

The Sign of the Real Part of Complex Solutions B

Real Solutions, Both Positive or Both Negative C

Real Solutions, Opposite Signs D

```
Show[CplexPlot,
 RealRootsPos,
 MixedRealRoots,
 RealRootsNeg]
```
15



1: `Reduce` is a function for determining the conditions on parameters (here β and γ assumed to be real numbers) such that an expression satisfies particular constraints. The results from `Reduce` will be used to build up a graphical representation of solution behavior by incremental steps.

A: (Algorithm suppressed in class-notes) `RegionPlot` is used in conjunction with the results produced by `Reduce` to illustrate the domains where the λ are complex and real. The regions are annotated and saved as a graphic to be concatenated below. This will create a plot that distinguishes two regions in the γ-β plane: above $\gamma = \beta^2/4$, the λ are real; below, the λ are complex and oscillatory solutions appear (because $\exp(r + \imath\theta) = \exp(r)(\cos(\theta) + \imath\sin(\theta))$).

B: The sign of the real part of the complex λ changes at $\beta = 0$. Here a graphic and its notation are created.

C: Using `Reduce` and `RegionPlot` in a manner that parallels **A**, domains for the λ having two real roots with the same sign are plotted and annotated.

D: Finally, the domain when the λ are real with opposite sign are illustrated.

2: Concatenating all the graphical objects together with `Show` produces the image that was used to construct Fig. 21-21.

3.016 Home
Full Screen
Close
Quit

The behavior of all solutions can be collected into a simple picture:

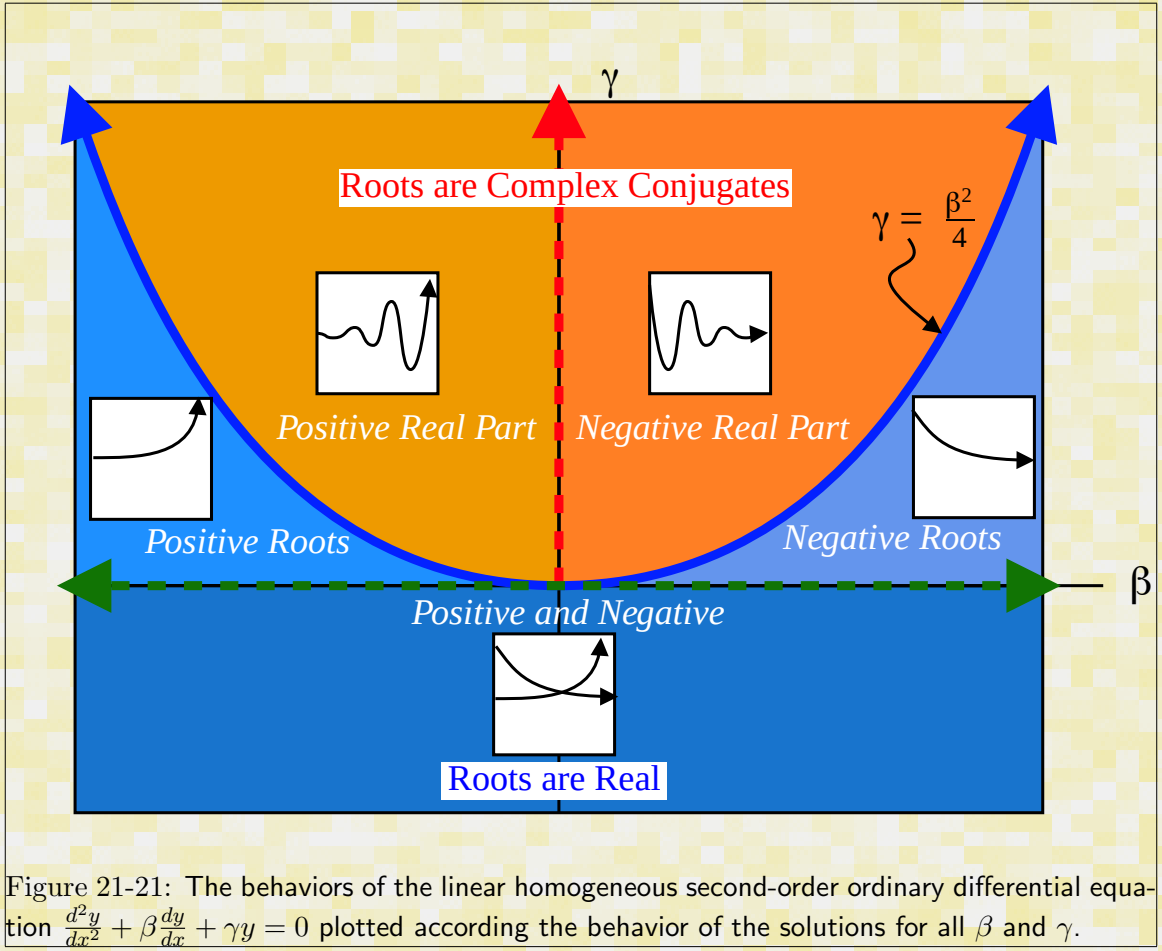


Figure 21-21: The behaviors of the linear homogeneous second-order ordinary differential equation $\frac{d^2y}{dx^2} + \beta\frac{dy}{dx} + \gamma y = 0$ plotted according the behavior of the solutions for all β and γ.

The case that separates the complex solutions from the real solutions, $\gamma = (\beta/2)^2$ must be treated separately, for the case $\gamma = (\beta/2)^2$ it can be shown that $y(x) = \exp(\beta x/2)$ and $y(x) = x\exp(\beta x/2)$ form an independent basis pair (see Kreyszig

3.016 Home
Full Screen
Close
Quit

AEM, p. 74).

Boundary Value Problems

It has been shown that all solutions to $\frac{d^2y}{dx^2} + \beta\frac{dy}{dx} + \gamma y = 0$ can be determined from a linear combination of the basis solution. Disregard for a moment whether the solution is complex or real, and ignoring the special case $\gamma = (\beta/2)^2$. The solution to any problem is given by

$$y(x) = C_+ e^{\lambda_+ x} + C_- e^{\lambda_- x} \tag{21-14}$$

How is a solution found for a particular problem? Recall that *two values* must be specified to get a solution—these two values are just enough so that the two constants C_+ and C_- can be obtained.

In many physical problems, these two conditions appear at the boundary of the domain. A typical problem is posed like this:

Solve

$$m\frac{d^2y(x)}{dx^2} + \nu\frac{dy(x)}{dx} + ky(x) = 0 \qquad \text{on } 0 < x < L \tag{21-15}$$

subject to the boundary conditions

$$y(x=0) = 0 \qquad \text{and} \qquad y(x=L) = 1$$

or, solve

$$m\frac{d^2y(x)}{dx^2} + \nu\frac{dy(x)}{dx} + ky(x) = 0 \qquad \text{on } 0 < x < \infty \tag{21-16}$$

subject to the boundary conditions

$$y(x=0) = 1 \qquad \text{and} \qquad y'(x=L) = 0$$

When the value of the function is specified at a point, these are called *Dirichlet* conditions; when the derivative is specified, the boundary condition is called a *Neumann* condition. It is possible have boundary conditions that are mixtures of Dirichlet and Neumann.

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

Determining Solution Constants from Boundary Values

Here is an example of taking the general solution with undetermined constants and using boundary conditions to determine a specific solution.

*Second order ODEs require that **two** conditions to generate a particular solution.*

```
GenSol[x]
```
1

Example of y(0) = 0 and y(L)=1:

```
SolutionOne =
 Solve[{GenSol[0] == 0,
   GenSol[L] == 1},
  {C[LPlus], C[LMinus]}]
```
2

Write the resulting solution:

```
SpecificSolutionOne =
 Simplify[
  GenSol[x] /. SolutionOne]
```
3

Second example: y(0) = 1 and y'(0)=0

```
DGen = D[GenSol[x], x]
```
4

Now the constants CPlus and CMinus are found by solving:

```
SolutionTwo =
 Solve[{GenSol[0] == 1,
   (DGen /. x → 0) == 0},
  {C[LPlus], C[LMinus]}]
```
5

```
SpecificSolutionTwo =
 Simplify[
  GenSol[x] /. SolutionTwo]
```
6

1: *GenlSol* is the solution to $y'' + \beta y' + \gamma y = 0$ with undetermined constants `Cplus` and `Cminus` that we derived above.

2: To demonstrate the method of using boundary conditions for a particular solution the boundary conditions, we use the example $y(0) = 0$ and $y(L) = 1$. `Solve` is used for the two conditions to find a rule-set for solutions in terms of the general-solution constants.

3: The form of the particular solution is obtained by back-substituting the solution for the constants into the general solution.

4: For application of a Neumann condition, the symbolic form of the derivative is required.

5–6: The particular solution for boundary conditions $y'(0) = y(0) = 0$ is obtained by inserting these equations into `Solve` and subsequent replacement into the general solution.

3.016 Home

Full Screen

Close

Quit

Fourth Order ODEs, Elastic Beams

Another linear ODE that has important applications in materials science is that for the deflection of a beam. The beam deflection $y(x)$ is a linear fourth-order ODE:

$$\frac{d^2}{dx^2}\left(EI\frac{d^2y(x)}{dx^2}\right) = w(x) \tag{21-17}$$

where $w(x)$ is the load density (force per unit length of beam), E is Young's modulus of elasticity for the beam, and I is the moment of inertia of the cross section of the beam:

$$I = \int_{\mathrm{A}_{\times-sect}} y^2 dA \tag{21-18}$$

is the second-moment of the distribution of heights across the area.

If the moment of inertia and the Young's modulus do not depend on the position in the beam (the case for a uniform beam of homogeneous material), then the beam equation becomes:

$$EI\frac{d^4y(x)}{dx^4} = w(x) \tag{21-19}$$

The homogeneous solution can be obtained by inspection—it is a general cubic equation $y_{homog}(x) = C_0 + C_1x + C_2x^2 + C_3x^3$ which has the four constants that are expected from a fourth-order ODE.

The particular solution can be obtained by integrating $w(x)$ four times—if the constants of integration are included then the particular solution naturally contains the homogeneous solution.

The load density can be discontinuous or it can contain Dirac-delta functions $F_o\delta(x-x_o)$ representing a point load F_o applied at $x = x_o$.

It remains to determine the constants from boundary conditions. The boundary conditions can be determined because each derivative of $y(x)$ has a specific meaning as illustrated in Fig. 21-22.

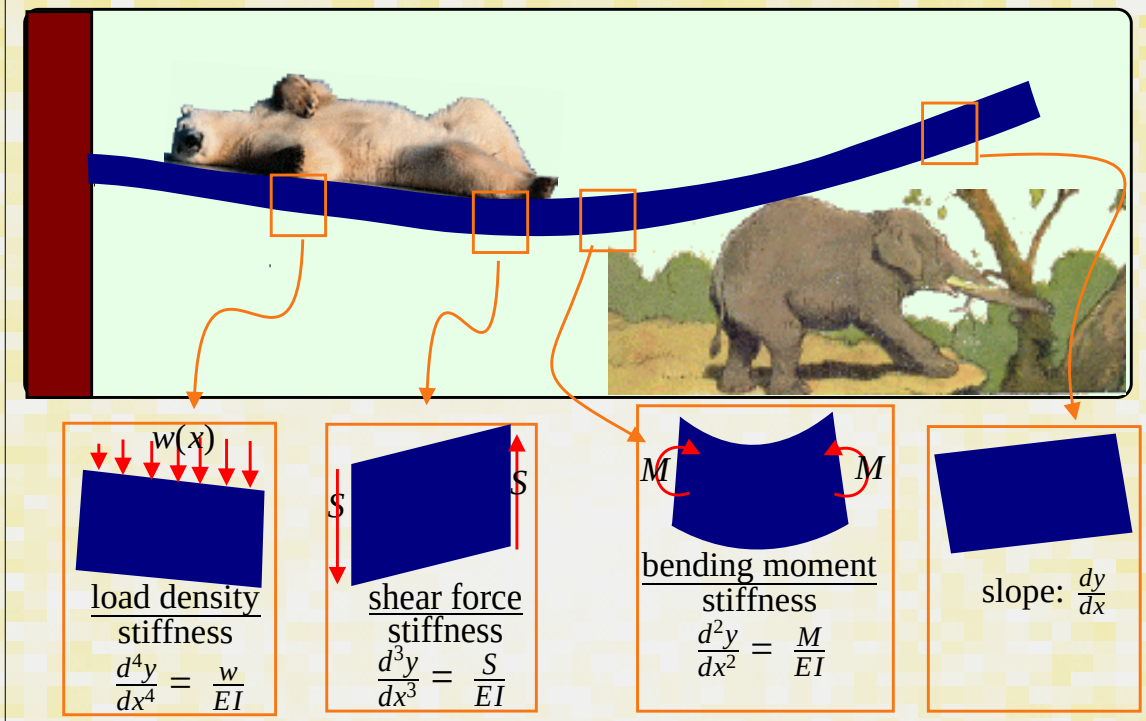


Figure 21-22: The shape of a loaded beam is determined by the loads applied over its length and its boundary conditions. The beam curvature is related to the local moment (imagine two handles rotated in opposite directions on a free beam) divided by the effective beam stiffness. Shear forces are related to the rate of change of moment along the beam.
(Polar Bear Photo Art Wolfe The Zone Network
http://classic.mountainzone.com/climbing/greenland/graphics/polar-bear.html)

There are common loading conditions that determine boundary conditions:

Free No applied moments or applied shearing force:

$$M = \left.\frac{d^2y}{dx^2}\right|_{boundary} = 0$$

$$S = \left.\frac{d^3y}{dx^3}\right|_{boundary} = 0$$

Point Loaded local applied moment, displacement specified.

$$M = \left.\frac{d^2y}{dx^2}\right|_{boundary} = M_o$$

$$y(x)|_{boundary} = y_o$$

Clamped Displacement specified, slope specified

$$\left.\frac{dy}{dx}\right|_{boundary} = s_o$$

$$y(x)|_{boundary} = y_o$$

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

A Function to Solve Beam Deflections for Common Boundary Conditions

A function for solving and visualizing the deflection of a uniform beam is developed for typical boundary conditions and load distributions

Set up the ODE solution (we will specify a unit beam stiffness) EI = 1

```
BeamEquation[y_ , x_ ,
  w_, BC1_ , BC2_] :=
 DSolve[{y''''[x] ==
     w[x], BC1, BC2},
   y[x], x] // Flatten
```

1

Functions for Typical Boundary Conditions and Beam Loading

A

```
BeamEquation[y, x,
 noload, Clamp[y, 0, 0, 0],
 Knob[y, 1, -.1, 0]]
```

10

$$\{y[x] \to -0.15\, x^2 + 0.05\, x^3\}$$

1: *BeamEquation* takes arguments for the (unknown) deflection y and its dependent argument x, a loading density $w(x)$, and boundary condition lists `BC1` and `BC2`, and uses `DSolve` to return replacement rules for a particular solution to the beam deflection equation (i.e., $d^4y/dx^4 = w(x)$).

A: The boundary conditions are defined as functions that return lists of equations for many common conditions (definitions suppressed in class-notes):

i *Clamp* `[y,x,position,slope]` fixed position and derivative at a specified point x of y.

ii *Knob* `[y,x,position,moment]` fixes the position and the moment at specified point.

iii *FreeBeam* `[y,x]` specifies that the beam is free to move at a specified point.

Particular types of applied loading functions ($w(x)$) are defined.

i *noload* `[x]`: no distributed load.

ii *unitload* `[x]`: unit (uniform) distributed load (i.e. 1 load unit/length).

iii *midload* `[x]` specifies that a (downward) load of magnitude 10 is applied at $x = 1/2$.

iv *boxload* `[x,position, width, magnitude]` specifies a uniform distributed load with `magnitude` over the domain `(position - width/2 < x < position + width/2)`

v *linearload* `[x]` specifies a linearly increasing load is applied, starting at -250 at $x = 0$ and increasing to +250 at $x = 1$.

10: An example of the use of *BeamEquation* with a clamp and a know is demonstrated.

3.016 Home

Full Screen

Close

Quit

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

Visualization of Beam Deflections

A graphical function is produced to visualize beam deflections for the specified boundary conditions and distributed loads.

```
Plot[Evaluate[
  y[x] /. BeamEquation[
    y, x, noload,
    Clamp[y, 0, 0, 0],
    Knob[y, 1, -.25, 0]]
], {x, 0, 1}]
```

1

BeamViz: Visualization Function

A

```
BeamViz[unitload,
 Clamp[y, 0, 0, 0],
 FreeBeam[y, 1]]
```

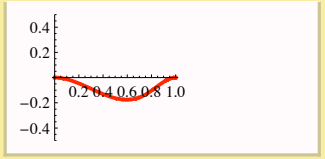
3

```
BeamViz[linearload,
 Knob[y, 0, 0, 0],
 Knob[y, 1, 0, 0]]
```

4

```
BeamViz[boxload[#,
   3 / 4, 1 / 8, -500] &,
 Clamp[y, 0, 0, 0],
 Clamp[y, 1, 0, 0]]
```

5



1: To plot the beam deflection, the solution condition is used as rule for replacement to $y(x)$

A: The function *BeamViz* collects the solution with the visualization for beams of unit normalized length, and uniform normalized stiffness EI.

3–4: These are examples of different loading conditions and boundary conditions are visualized as examples of *BeamViz* .

5: In this case, we have to force the loading condition $w(x)$ to be a function of a single variable. Because *boxload* takes four arguments, it is necessary to use it to construct a pure function.

3.016 Home
Full Screen
Close
Quit

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

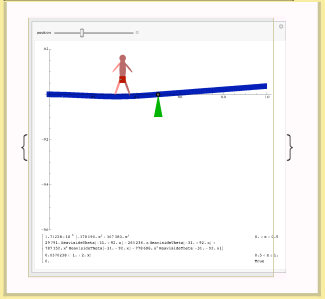
A Gratuitous Animations of Deflections of a Diving Board

A diving board can be approximated as a beam of uniform stiffness which is cantilevered at one end, and has an adjustable pin near its center. We work out a solution and visualization for a diver walking out on a diving board with the pin fixed at the center.

We wish to simulate the deflection of a diving board as a diver walks toward the end. A diving board may be modeled as beam with constant cross-section. The boundary conditions are that the board is clamped at the beginning; has a pivot located somewhere near the center; and the "diving" end . For "competitive" diving boards, the pivot point is adjustable. Here we fix the pivot at x=1/2.

This next function is a silly little graphic for the walking diver.

Graphics Functions for Animating Board and Diver **A**



A: (Solution and Algorithm Suppressed) We start by defining a set of graphics that take a single argument to represent the diver, the argument is designed to make the diver appear to be walking.

The solution to this deflection of the diving board is not trivial. The diver is modeled as a point load. The board is modeled as a piecewise solution for two beams that share a common boundary condition. We will need to match boundary conditions at the pin. The method is to find a solution in terms of an arbitrary coefficient, and then solve for the instance of that coefficient that makes the board deflection continuous.

If the diver is located between the cantilever and the pin, then the deflection in this region is determined with simple clamp and pin boundary conditions and the `DiracDelta` function for the distributed load. From this solution, the slope at the mid-point is calculated and this value is used as one of the boundary condition between the pin and the free-end.

If the diver is located between the pin and the free-end, the the slope at the pin is an undermined constant first in this region and then in the region between the cantilever and the pin. `Solve` is used to match the slope at the pin, and that solution is applied to the previously obtained solutions (i.e., the solutions from `DSolve`.

`Piecewise` is used in the definition of the solution-function that is returned.

`Manipulate` is used to specify the position of the diver.

3.016 Home

Full Screen

Close

Quit

Index