Numerical Solutions to Differential Equations

Lecture Notes #19 — BVP: The Shooting Method, continued

Peter Blomgren,
⟨blomgren.peter@gmail.com⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://terminus.sdsu.edu/

Spring 2015

Outline

Quick Recap — Boundary Value Problems

Last time:

- Physical motivation for boundary value problems — bending beams (constructing bridges and buildings); cooling fins — keeping those processors running!
- The shooting method — convert a BVP into a sequence of IVPs and apply techniques from the first half of the semester!
  - Variational approach — add ODEs for the **sensitivity variables**.
  - Finite difference approach — approximate the sensitivity by differences of the results of different initial guesses.

Today's Lecture... and Looking Forward

- **Theory**
  - Generalize shooting methods to larger systems (n simultaneous ODEs).
- **Example**
  - Shooting for a 4th order ODE — Beam Bending.
- **Other Approaches**
  - Finite Difference Methods (next time)
- **Higher Order Equations (FD)**
- **Nonlinear Equations**
- **"Topics"**

Generalizing Shooting to Systems of ODEs

Given a system of simultaneous ODEs

$$\begin{cases} y_1'(x) &= f_1(x, y_1, y_2, \dots, y_n) \\ y_2'(x) &= f_2(x, y_1, y_2, \dots, y_n) \\ &\vdots \\ y_n'(x) &= f_n(x, y_1, y_2, \dots, y_n) \end{cases} \qquad x \in [a, b]$$

with boundary conditions

$$\begin{aligned} y_i(b) &= y_i^b, \quad i = 1, 2, \dots, k \\ y_i(a) &= y_i^a, \quad i = k+1, k+2, \dots, n \end{aligned}$$

In order to convert this to an initial value problem, we have to replace the first k terminal conditions with k (guessed) initial conditions.

k-dimensional Initial Guesses

We want to find k initial guesses

$$y_i(a) = Y_i, \quad i = 1, 2, \dots, k$$

so that the solution to the initial value problem

$$\begin{cases} y_1'(x) &= f_1(x, y_1, y_2, \dots, y_n) \\ y_2'(x) &= f_2(x, y_1, y_2, \dots, y_n) \\ &\vdots \\ y_n'(x) &= f_n(x, y_1, y_2, \dots, y_n) \end{cases} \qquad x \in [a, b]$$

with **initial** conditions

$$\begin{aligned} y_i(a) &= Y_i, \quad i = 1, 2, \dots, k \\ y_i(a) &= y_i^a, \quad i = k+1, k+2, \dots, n \end{aligned}$$

Satisfies the terminal conditions $y_i(b) = y_i^b, \quad i = 1, 2, \dots, k$.

k-dimensional Discrepancy Functions

- Let $\tilde{\mathbf{Y}} = \{Y_1, Y_2, \dots, Y_k\}^T$ be the vector of guessed initial values.
- Let $y_i(b; \tilde{\mathbf{Y}})$, $i = 1, 2, \dots, k$ be the terminal values.
- Define $h_i(\tilde{\mathbf{Y}}) = y_i(b; \tilde{\mathbf{Y}}) - y_i^b$, $i = 1, 2, \dots, k$ be the discrepancy functions — measuring how far off the computed terminal solutions are from the desired values of the terminal conditions.
- We are now looking for a correction $\Delta\tilde{\mathbf{Y}}$ to the guesses $\tilde{\mathbf{Y}}$, so that the corrected initial conditions lead to a solution with $h(\tilde{\mathbf{Y}} + \Delta\tilde{\mathbf{Y}}) = 0$.
- We use our favorite mathematical tool — the Taylor Expansion — to get an equation for the correction.

Taylor Expanding and Truncating

We Taylor expand, and throw out terms of order ≥ 2 (just as in the Taylor-derivation of Newton's Method — Math 541):

$$0 = h(\tilde{\mathbf{Y}} + \Delta\tilde{\mathbf{Y}}) = h_i(\tilde{\mathbf{Y}}) + \sum_{j=1}^{k} \left[\frac{\partial h_i}{\partial Y_j} \Delta Y_j \right], \quad i = 1, 2, \dots, k$$

We end up with the following $k \times k$ system of equations

$$\begin{array}{ccccccc} \left[\frac{\partial h_1}{\partial Y_1}\right] \Delta Y_1 &+& \left[\frac{\partial h_1}{\partial Y_2}\right] \Delta Y_2 &+ \cdots +& \left[\frac{\partial h_1}{\partial Y_k}\right] \Delta Y_k &=& -h_1(\tilde{\mathbf{Y}}) \\ \left[\frac{\partial h_2}{\partial Y_1}\right] \Delta Y_1 &+& \left[\frac{\partial h_2}{\partial Y_2}\right] \Delta Y_2 &+ \cdots +& \left[\frac{\partial h_2}{\partial Y_k}\right] \Delta Y_k &=& -h_2(\tilde{\mathbf{Y}}) \\ \vdots && \vdots && \vdots && \\ \left[\frac{\partial h_k}{\partial Y_1}\right] \Delta Y_1 &+& \left[\frac{\partial h_k}{\partial Y_2}\right] \Delta Y_2 &+ \cdots +& \left[\frac{\partial h_k}{\partial Y_k}\right] \Delta Y_k &=& -h_k(\tilde{\mathbf{Y}}) \end{array}$$

A Little Bit of Matrix Notation

- Let $\Delta\tilde{\mathbf{Y}} = \{\Delta Y_1, \Delta Y_2, \ldots, \Delta Y_k\}^T$ be the vector of updates.
- Let $\tilde{\mathbf{h}}(\tilde{\mathbf{Y}}) = \{h_1(\tilde{\mathbf{Y}}), h_2(\tilde{\mathbf{Y}}), \ldots, h_k(\tilde{\mathbf{Y}})\}^T$ be the vector of discrepancy functions.
- Let the matrix $J(\tilde{\mathbf{Y}}, b)$ be the matrix the **Jacobian**, with entries

$$J_{i,j} = \left.\frac{\partial h_i}{\partial Y_j}\right|_{x=b}$$

- Then the equation for the update becomes

$$\Delta\tilde{\mathbf{Y}} = -\left[\mathbf{J}(\tilde{\mathbf{Y}}, \mathbf{b})\right]^{-1} \tilde{\mathbf{h}}(\tilde{\mathbf{Y}})$$

Computing the Entries of the Jacobian at $x = b$

The entries of the Jacobian are the partial derivatives of the discrepancy functions with respect to the guessed initial values **computed at the terminal point.**

As in the one-constraint problem we looked at last time, we have to derive **additional ODEs** to get equations for the needed values.

We differentiate the ODEs we already have, with respect to the guessed initial values; apply the chain rule, and the fact that we can switch the order of differentiation... we get...

$$\frac{\partial}{\partial Y_j}\left[\frac{dy_i}{dx}\right] = \frac{d}{dx}\left[\frac{\partial y_i}{\partial Y_j}\right] = \sum_{k=1}^{n} \frac{\partial f_i}{\partial y_k}\frac{\partial y_k}{\partial Y_j}$$

where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, k$.

Equations for the Sensitivity Functions I/II

We define the sensitivity functions

$$g_{ij} = \frac{\partial y_i}{Y_j} \quad \left[= \frac{\partial h_i}{\partial Y_j}\right]$$

and get the following set of $n \times k$ ODEs:

$$\frac{dg_{ij}}{dx} = \sum_{k=1}^{n}\left[g_{kj}\frac{\partial f_i}{\partial y_k}\right], \quad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, k$$

Equations for the Sensitivity Functions II/II

The initial conditions for the sensitivity functions are

$$g_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}; \quad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, k$$

This makes sense since at $x = a$ there is no mixing of the guessed values —

- $y_i(a) \equiv Y_i$, $i = 1, 2, \ldots, k$, and
- $y_i(a) = y_i^a$, $i = k+1, k+2, \ldots, n$.

Putting the Pieces Together I/II

Now, we solve the following IVP consisting of $(n + n \times k)$ simultaneous ODEs:

$$\begin{cases} y_1'(x) &= f_1(x, y_1, y_2, \dots, y_n) \\ y_2'(x) &= f_2(x, y_1, y_2, \dots, y_n) \\ &\vdots \\ y_n'(x) &= f_n(x, y_1, y_2, \dots, y_n) \\ g_{ij}' &= \sum_{k=1}^{n} \left[g_{kj} \frac{\partial f_i}{\partial y_k} \right], \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, k \end{cases}$$

with **initial** conditions

$$\begin{aligned} y_i(a) &= Y_i, \quad i = 1, 2, \dots, k \\ y_i(a) &= y_i^a, \quad i = k+1, k+2, \dots, n \\ g_{ij} &= \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}; \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, k \end{aligned}$$

Putting the Pieces Together II/II

At the terminal point $x = b$, we compute the discrepancy functions $\tilde{\mathbf{h}}(\tilde{\mathbf{Y}})$, and the entries of the Jacobian $J_{i,j} = g_{i,j}(b)$.

If

$$\|\tilde{\mathbf{h}}(\tilde{\mathbf{Y}})\| > \textit{tolerance},$$

(or other stopping criteria) then we update the guess

$$\tilde{\mathbf{Y}}^{(s+1)} = \tilde{\mathbf{Y}}^{(s)} - \left[J(\tilde{\mathbf{Y}}^{(s)}, b) \right]^{-1} \tilde{\mathbf{h}}(\tilde{\mathbf{Y}}^{(s)}),$$

and start over.

Comments...

- We started with n ODEs.
- The equations for the sensitivity functions added $(\mathbf{n} \times \mathbf{k})$ ODEs.
- That can be a high price to pay! If $n = 1000$ and $k = 500$ (a very reasonably sized problem), then the extended system has 501,000 equations!
- **The good news:** The ODEs and initial conditions for the additional equations are very easy to write down:

$$\begin{aligned} g_{ij}' &= \sum_{k=1}^{n} \left[g_{kj} \frac{\partial f_i}{\partial y_k} \right], \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, k \\ g_{ij}(a) &= \delta_{ij} \end{aligned}$$

The Numerical Alternative — Suitable for Concurrency

- If/When the price is too high, we can compute numerical (difference) approximations of the terminal values of the sensitivity functions.
- Let $\tilde{\mathbf{Y}}_j^\epsilon = \epsilon\{\delta_{1j}, \delta_{2j}, \dots, \delta_{kj}\}$, *i.e.* the vector of all zeros, except the value ϵ in the jth position.
- If we solve the initial value problem for the two initial guesses $\tilde{\mathbf{Y}}$ and $\tilde{\mathbf{Y}} + \tilde{\mathbf{Y}}_j^\epsilon$ we can compute the difference approximations

$$\left.\frac{\partial h_i}{\partial Y_j}\right|_{x=b} \approx \frac{h_i(\tilde{\mathbf{Y}} + \tilde{\mathbf{Y}}_j^\epsilon) - h_i(\tilde{\mathbf{Y}})}{\epsilon}, \quad i = 1, 2, \dots, k$$

  Let $j = 1, 2, \dots, k$ gives us approximations to all entries of the Jacobian.
- **The price:** Solving the system of n ODEs $(\mathbf{k} + \mathbf{1})$ times.

Example: Shooting for the Euler-Bernoulli Beam Equation

Transverse deflection of a beam $w(x)$ subject to distributed load, $p(x)$

$$\frac{d^2}{dx^2}\left[E(x)I(x)\frac{d^2w(x)}{dx^2}\right] = p(x).$$

Here, we will assume a uniform beam — *i.e.* $E(x)$ and $I(x)$ are constant. For simplicity $E(x)I(x) = 1$.

We'll let the beam have length $L = 1$, and be fixed at the end points (like a book-shelf).

We use a non-uniform load function:

$$p(x) = e^{\frac{(x-L/2)^2}{(L/8)^2}}.$$

Shooting for Beam Bending: Equations

Our problem is:

$$\frac{d^4w(x)}{dx^4} = e^{\frac{(x-L/2)^2}{(L/8)^2}},$$

subject to

$$w(0) = w'(0) = w(1) = w'(1) = 0.$$

We introduce $y_i = \frac{d^{i-1}y(x)}{dx^{i-1}}$, $i = 1, 2, 3, 4$ and get the following system of ODEs:

$$\frac{d}{dx}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} y_2 \\ y_3 \\ y_4 \\ e^{\frac{(x-L/2)^2}{(L/8)^2}} \end{bmatrix}, \quad \begin{cases} y_1(0) = 0 \\ y_2(0) = 0 \\ \mathbf{y_1(1) = 0} \\ \mathbf{y_2(1) = 0} \end{cases}$$

Shooting for Beam Bending: IVP

We are going to solve the following IVP

$$\frac{d}{dx}\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} y_2 \\ y_3 \\ y_4 \\ e^{\frac{(x-L/2)^2}{(L/8)^2}} \end{bmatrix}, \quad \begin{cases} y_1(0) = 0 \\ y_2(0) = 0 \\ \mathbf{y_3(0) = A} \\ \mathbf{y_4(0) = B} \end{cases}$$

and numerically determine the parameters A and B so that the terminal conditions $y_1(1) = 0$ and $y_2(1) = 0$.

Code: RKF45 Shooting for Beam Bending I/V

Code: Shooting Segment #1

```
% Shooting for a uniform fixed Beam --- Octave Code [www.octave.org]
% E(x) = Constant, I(x) = Constant
clear all

% Length of the Beam
global L;
L = 1;

% The Load Function
function p = p(x)
  global L
  p = -exp(-(x-L/2).^2/(L/8)^2);
endfunction

% The Forcing Function of the System of ODEs
function rhs_rkf45 = rhs_rkf45(x,w)
  rhs_rkf45 = [w(2:4); p(x)];
endfunction
```

Code: RKF45 Shooting for Beam Bending II/V

Code: Shooting — Segment #2

```
function [y,xv] = RKF45(y0,x0,L)
  c   = [0 1/4 3/8 12/13 1 1/2];
  A   = [0 0 0 0 0 0; 1/4 0 0 0 0 0; 3/32 9/32 0 0 0 0];
  A   = [A; 1932/2197 -7200/2197 7296/2197 0 0 0];
  A   = [A; 439/216 -8 3680/513 -845/4104 0 0];
  A   = [A; -8/27 2 -3544/2565 1859/4104 -11/40 0];
  b1  = [ 25/216 0 1408/2565 2197/4104 -1/5 0];
  b2  = [ 16/135 0 6656/12825 28561/56430 -9/50 2/55];
  E   = [ 1/360 0 -128/4275 -2197/75240 1/50 2/55];
  h   = L/16;
  TOL = 1e-12;
  y = y0;  yN = y0;  xv = x0;  x = x0;
  while( x < L )
    if( x+h > L )
      h = L-x;
    end
    k    = zeros(4,6);
    k(:,1) = rhs_rkf45( x+h*c(1), yN+h*(A(1,:)*k.').' );
    k(:,2) = rhs_rkf45( x+h*c(2), yN+h*(A(2,:)*k.').' );
    k(:,3) = rhs_rkf45( x+h*c(3), yN+h*(A(3,:)*k.').' );
    k(:,4) = rhs_rkf45( x+h*c(4), yN+h*(A(4,:)*k.').' );
    k(:,5) = rhs_rkf45( x+h*c(5), yN+h*(A(5,:)*k.').' );
    k(:,6) = rhs_rkf45( x+h*c(6), yN+h*(A(6,:)*k.').' );
```


Code: RKF45 Shooting for Beam Bending III/V

Code: Shooting — Segment #3

```
    yNext  = yN + h*(b1*k.').';
    yErr   = h*(E*k.').';
    yErrAbs = norm(yErr);
    if( yErrAbs < TOL )
      y       = [y yNext];
      yN      = yNext;
      xv      = [xv x+h];
      x       = x+h;
      if( yErrAbs*20 < TOL )
        h = h*2;
      end
    else
      h = h/2;
    end
  end
endfunction
```


Code: RKF45 Shooting for Beam Bending IV/V

Code: Shooting — Segment #4

```
% Initial initial Values
w0 = [0 0 0 0].';

tol     = 1e-8;
Perturb = 10*tol;
Err     = 2 *tol;
while( Err > tol )
  [y,xv]          = RKF45(w0,0,L);
  Y_nonperturbed = y;
  Y_np_final      = y(:,length(xv));
  W1_discr        = y(1,length(xv));
  W2_discr        = y(2,length(xv));
  P_factor        = min(diff(xv));
  Err             = norm([W1_discr W2_discr])

  % Skip out of the loop when tolerance is met
  if( Err <= tol ) break; end
```


Code: RKF45 Shooting for Beam Bending V/V

Code: Shooting — Segment #5

```
  wA0 = w0;  wA0(3) = wA0(3) + Perturb;
  [y,xv]          = RKF45(wA0,0,L);
  Y_perturb_w3    = y;
  Y_w3_final      = y(:,length(xv));

  wB0 = w0;  wB0(4) = wB0(4) + Perturb;
  [y,xv]          = RKF45(wB0,0,L);
  Y_perturb_w4    = y;
  Y_w4_final      = y(:,length(xv));

  J11 = (Y_w3_final(1)-Y_np_final(1)) / (Perturb) ;
  J12 = (Y_w4_final(1)-Y_np_final(1)) / (Perturb) ;
  J21 = (Y_w3_final(2)-Y_np_final(2)) / (Perturb) ;
  J22 = (Y_w4_final(2)-Y_np_final(2)) / (Perturb) ;
  Ja = [J11 J12; J21 J22];

  w0(3:4) = w0(3:4) - Ja\[W1_discr; W2_discr];
end
```

Beam Bending: Numerical Results

Iteration	Discrepancy
1	0.029003
2	1.7206e–11

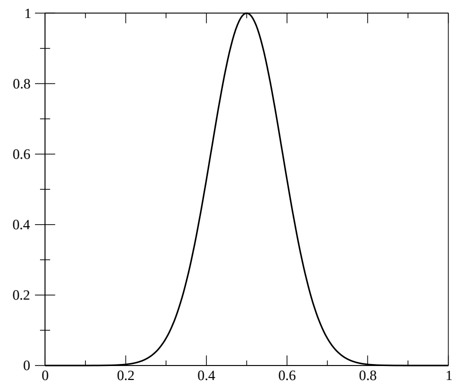

FIGURE 1: The distributed load.

Beam Bending: Numerical Results — The Displacement

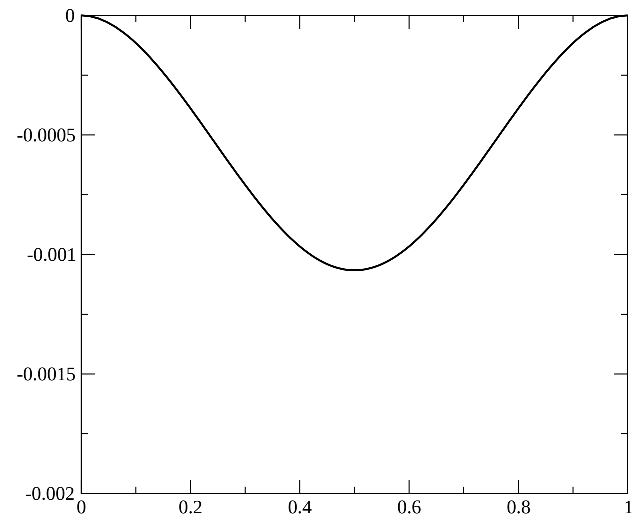

FIGURE 2: The displacement due to the load, $w(x)$, $[y_1(x)]$.

Beam Bending: Numerical Results — $w'(x)$

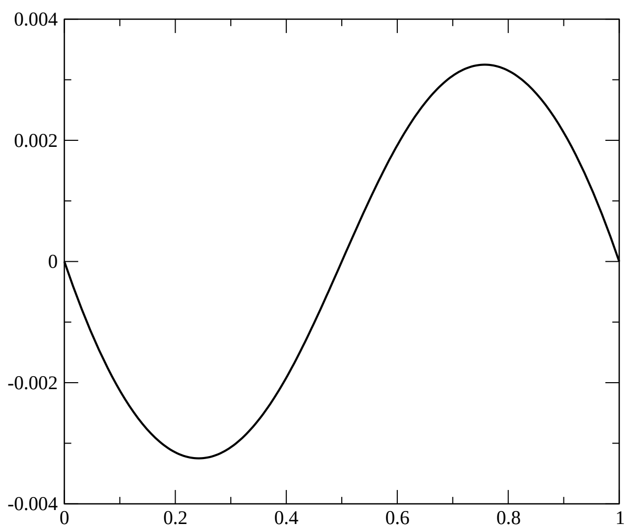

FIGURE 3: $w'(x)$, $[y_2(x)]$.

Beam Bending: Numerical Results — Curvature, $w''(x)$ — Bending Moment

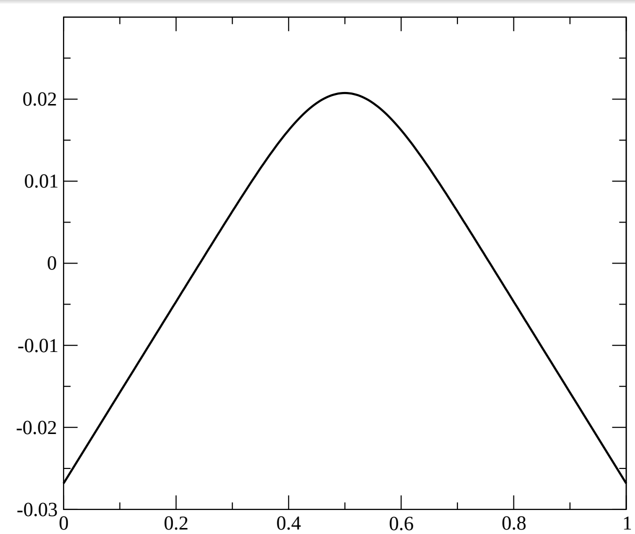

FIGURE 4: $w''(x)$, $[y_3(x)]$.

Beam Bending: Numerical Results — $w'''(x)$ — Shear Force

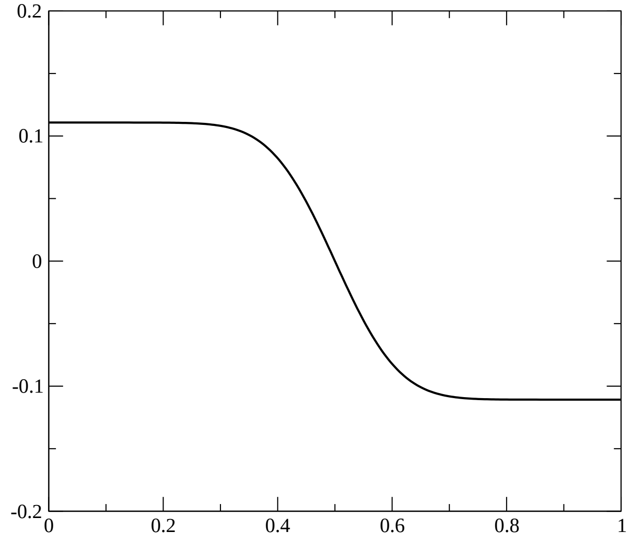

FIGURE 5: $w'''(x)$, $[y_4(x)]$.

Bending Moment

Wikipedia

Bending Moment

A bending moment exists in a structural element when a moment is applied to the element so that the element bends. Moments and torques are measured as a force multiplied by a distance so they have as unit newton-metres (N·m) , or foot-pounds force (ft·lbf).

Tensile stresses and compressive stresses increase proportionally with bending moment, but are also dependent on the second moment of area of the cross-section of the structural element. Failure in bending will occur when the bending moment is sufficient to induce tensile stresses greater than the yield stress of the material throughout the entire cross-section. It is possible that failure of a structural element in shear may occur before failure in bending, however the mechanics of failure in shear and in bending are different.

Shearing

Wikipedia

Shearing (physics)

Shearing in continuum mechanics refers to the occurrence of a shear strain, which is a deformation of a material substance in which parallel internal surfaces slide past one another. It is induced by a shear stress in the material.

Often, the verb shearing refers more specifically to a mechanical process that causes a plastic shear strain in a material, rather than causing a merely elastic one. A plastic shear strain is a continuous (non-fracturing) deformation that is irreversible, such that the material does not recover its original shape. It occurs when the material is yielding.