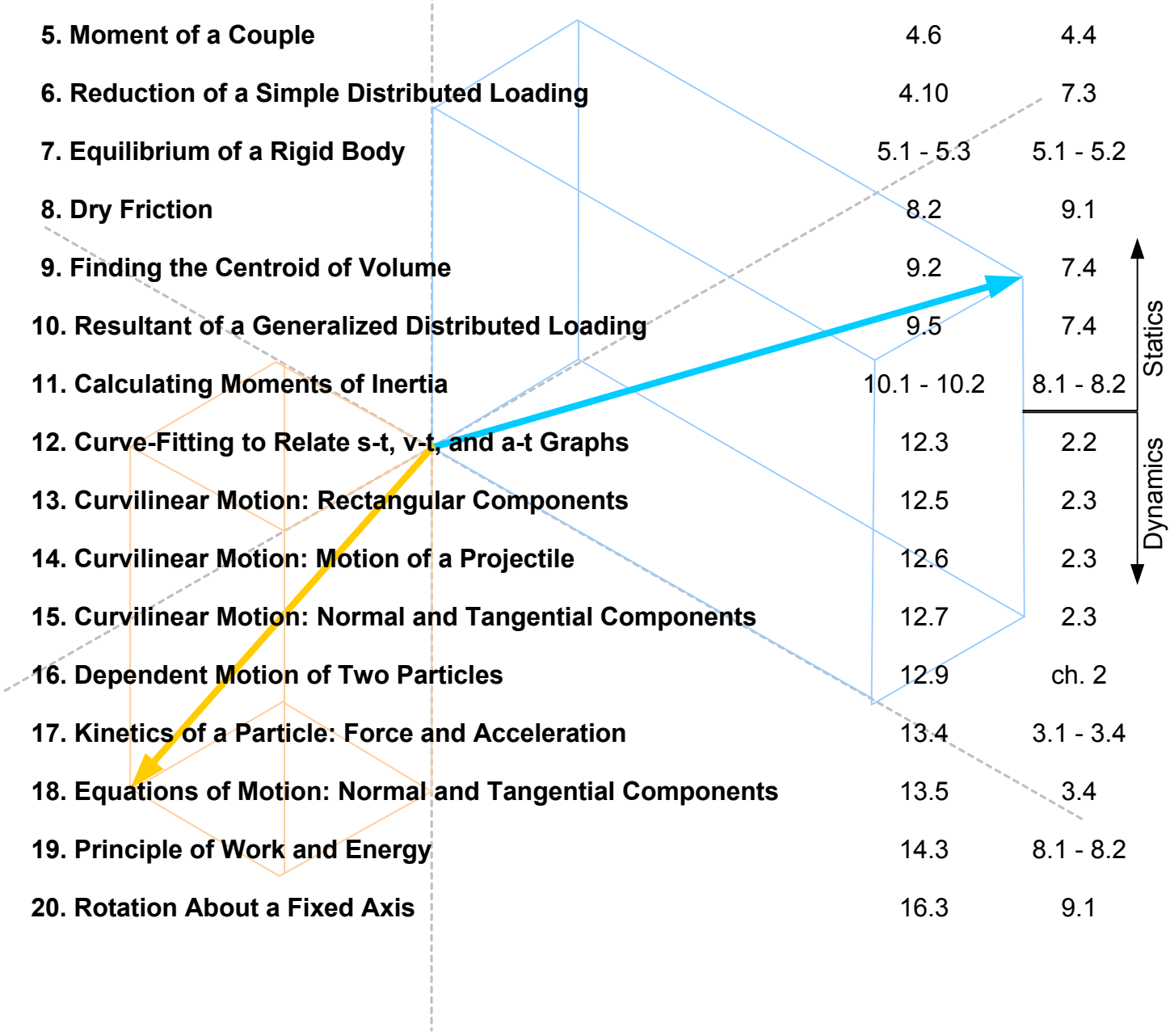
Using MATLAB for Statics and Dynamics

by Ron Larsen and Steve Hunt

	Hibbeler	Bedford and Fowler
1. Resolving Forces, Calculating Resultants	2.4 - 2.6	2.3 - 2.5
2. Dot Products	2.9	2.5
3. Equilibrium of a Particle, Free-Body Diagrams	3.3	3.2 - 3.3
4. Cross Products and Moments of Forces	4.2 - 4.3	2.6, 4.3
5. Moment of a Couple	4.6	4.4
6. Reduction of a Simple Distributed Loading	4.10	7.3
7. Equilibrium of a Rigid Body	5.1 - 5.3	5.1 - 5.2
8. Dry Friction	8.2	9.1
9. Finding the Centroid of Volume	9.2	7.4
10. Resultant of a Generalized Distributed Loading	9.5	7.4
11. Calculating Moments of Inertia	10.1 - 10.2	8.1 - 8.2
12. Curve-Fitting to Relate s-t, v-t, and a-t Graphs	12.3	2.2
13. Curvilinear Motion: Rectangular Components	12.5	2.3
14. Curvilinear Motion: Motion of a Projectile	12.6	2.3
15. Curvilinear Motion: Normal and Tangential Components	12.7	2.3
16. Dependent Motion of Two Particles	12.9	ch. 2
17. Kinetics of a Particle: Force and Acceleration	13.4	3.1 - 3.4
18. Equations of Motion: Normal and Tangential Components	13.5	3.4
19. Principle of Work and Energy	14.3	8.1 - 8.2
20. Rotation About a Fixed Axis	16.3	9.1



1 Resolving Forces, Calculating Resultants

Ref: Hibbeler § 2.4-2.6, Bedford & Fowler: Statics § 2.3-2.5

Resolving forces refers to the process of finding two or more forces which, when combined, will produce a force with the same magnitude and direction as the original. The most common use of the process is finding the components of the original force in the Cartesian coordinate directions: x, y, and z.

A *resultant* force is the force (magnitude and direction) obtained when two or more forces are combined (i.e., added as vectors).

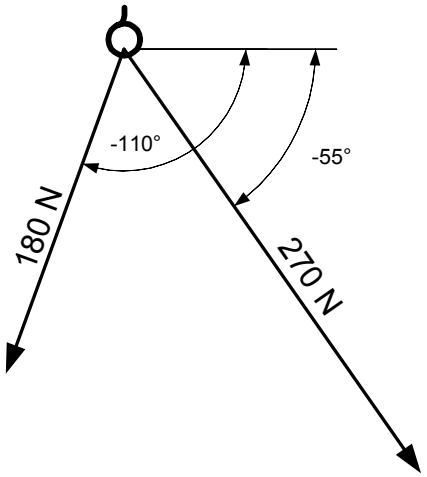
Breaking down a force into its Cartesian coordinate components (e.g., F_x, F_y) and using Cartesian components to determine the force and direction of a resultant force are common tasks when solving statics problems. These will be demonstrated here using a two-dimensional problem involving co-planar forces.

Example: Co-Planar Forces

Two boys are playing by pulling on ropes connected to a hook in a rafter. The bigger one pulls on the rope with a force of 270 N (about 60 lb_f) at an angle of 55° from horizontal. The smaller boy pulls with a force of 180 N (about 40 lb_f) at an angle of 110° from horizontal.

a. Which boy is exerting the greatest vertical force (downward) on the hook?

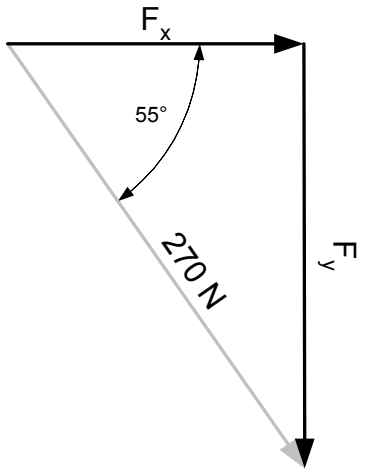
b. What is the net force (magnitude and direction) on the hook – that is, calculate the resultant force.



Note: The angles in this figure have been indicated as *coordinate direction angles*. That is, each angle has been measured from the positive x axis.

Solution

First, consider the 270 N force acting at 55° from horizontal. The x- and y-components of force are indicated schematically, as



The x- and y-components of the first force (270 N) can be calculated using a little trigonometry involving the included angle, 55°:

$$\cos(55°) = \frac{F_{x1}}{270\ \text{N}}, \quad \text{or} \quad F_{x1} = (270\ \text{N})\cos(55°)$$

and

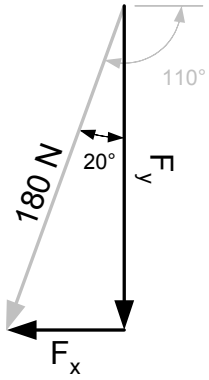
$$\sin(55°) = \frac{F_{y1}}{270\ \text{N}}, \quad \text{or} \quad F_{y1} = (270\ \text{N})\sin(55°).$$

MATLAB can be used to solve for F_{x1} and F_{y1} using its built-in sin() and cos() functions, but these functions assume that the angle will be expressed as radians, not degrees. The factor pi/180 is used to convert the angle from degrees to radians. Note that pi is a predefined variable in MATLAB.

```
» F_x1 = 270 * cos( 55 * pi/180 )
 F_x1 =
   154.8656
» F_x1 = 270 * sin( 55 * pi/180 )
 F_x1 =
   221.1711
```

Your Turn

Show that the x- and y-components of the second force (180 N acting at 110° from the x-axis) are 61.5 N (-x direction) and 169 N (-y direction), respectively. Note that trigonometry relationships are based on the included angle of the triangle (20°, as shown at the right), not the coordinate angle (-110° from the x-axis).

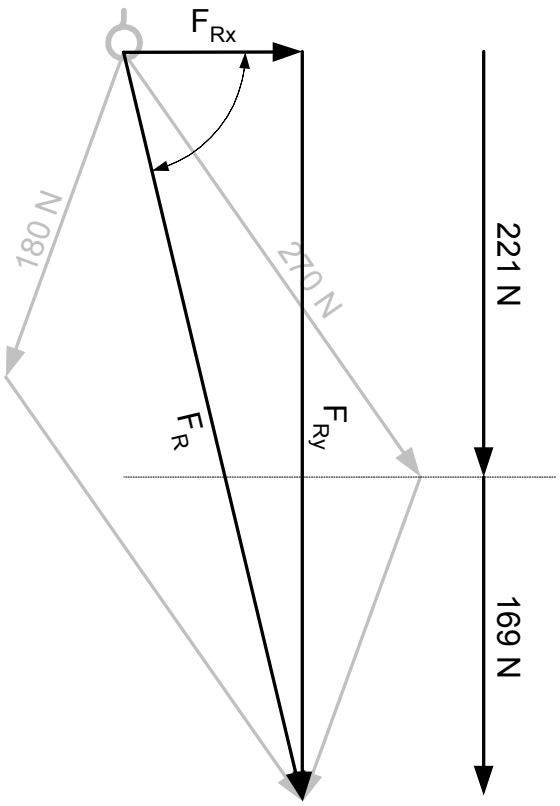


Answer, part a)

The larger boy exerts the greatest vertical force (221 N) on the hook. The vertical force exerted by the smaller boy is only 169 N.

Solution, continued

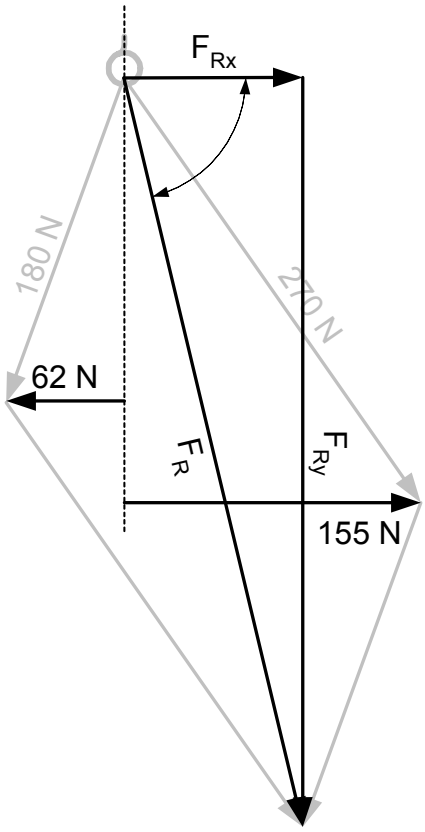
To determine the combined force on the hook, F_R, first add the two y-components calculated above, to determine the combined y-directed force, F_{Ry}, on the hook:




```
» F_Ry = F_y1 + F_y2

F_Ry =

  390.3157
```

The y-component of the resultant force is 390 N (directed down, or in the –y direction). Note that the direction has not been accounted for in this calculation.

Then add the two x-components to determine the combined x-directed force, F_{Rx}, on the hook. Note that the two x-component forces are acting in opposite directions, so the combined x-directed force, F_{Rx}, is smaller than either of the components, and directed in the +x direction.




```
» F_Rx = F_x1 + ( -F_x2 )
F_Rx =
  93.3020
```

The minus sign was included before F_{x2} because it is directed in the –x direction. The result is an x-component of the resultant force of 93 N in the +x direction.

Once the x- and y-components of the resultant force have been determined, the magnitude can be calculated using

$$F_R = \sqrt{F_{Rx}{}^2 + F_{Ry}{}^2}$$

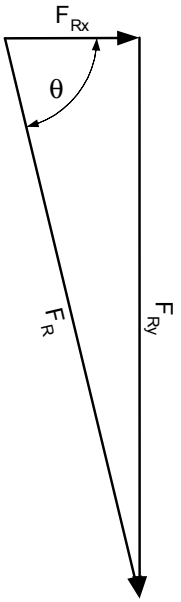
The MATLAB calculation uses the built-in square-root function sqrt().

```
» F_R  = sqrt( F_Rx ^ 2 + F_Ry ^ 2 )
F_R =
  401.3124
```

The angle of the resultant force can be calculated using any of three functions in MATLAB:

Function	*Argument(s)*	*Notes*
atan(abs(Fx / Fy))	one argument: $\text{abs}(F_x / F_y)$	Returns the included angle
atan2(F_y, F_x)	two arguments: F_x and F_y	Returns the coordinate direction angle Angle value is always between 0 and π radians (0 and 180°) A negative sign on the angle indicates a result in one of the lower quadrants of the Cartesian coordinate system
cart2pol (F_x, F_y)	two arguments: F_x and F_y	Returns the positive angle from the positive x-axis to the vector Angle value always between 0 and 2π radians (0 and 360°) An angle value greater than 180° (π radians) indicates a result in one of the lower quadrants of the Cartesian coordinate system

The atan2() function is used here, and F_{Ry} is negative because it is acting in the –y direction.




```
» F_Rx  =   93.302;
» F_Ry  = -390.316;
» theta   = 180/pi * atan2( F_Ry, F_Rx )
theta =
  -76.5562
```

Answer, part b)

The net force (magnitude and direction) on the hook is now known:

F_R = 401 N (about 90 lb_f) acting at an angle 76.6° below the x-axis.

Annotated MATLAB Script Solution

```
% Determine the x- and y-components of the two forces
% (270 N at -55°, and 180 N at -110°)
%
%   Note: These trig. Calculations use the included angles
%   (55° & 20°), with minus signs added to both y-component
%   equations to indicate the forces act in the -y direction,
%   and the F_x2 equation to show that this force acts in
%   the -x direction.

% Calculate the x- and y- components of the first force (270 N)
F_x1 =  270 * cos( 55 * pi/180 );
F_y1 = -270 * sin( 55 * pi/180 );
fprintf('\nF_x1 = %8.3f N\t F_y1 = %+9.3f N\n',F_x1,F_y1);

% Calculate the x- and y- components of the first force (180 N)
F_x2 = -180 * sin( 20 * pi/180 );
F_y2 = -180 * cos( 20 * pi/180 );
fprintf('F_x2 =  %7.3f N\t F_y2 = %9.3f N\n',F_x2,F_y2);

% Sum the y-components of the two forces to determine the
% y-component of the resultant force
F_Ry = F_y1 + F_y2;

% Sum the x-components of the two forces to determine the
% x-component of the resultant force
F_Rx = F_x1 + F_x2;
fprintf('F_Rx =  %7.3f N\t F_Ry = %9.3f N\n\n',F_Rx,F_Ry);

% Calculate the magnitude of the resultant force
F_R  = sqrt( F_Rx ^ 2 + F_Ry ^ 2 );
fprintf('F_R  = %8.3f N\n',F_R);

% Calculate the angle of the resultant force
% (in degrees from the x-axis)
theta = atan2( F_Ry, F_Rx ) * 180/pi;
fprintf('theta = %7.3f N\n\n',theta);
```

2 Dot Products

Ref: Hibbeler § 2.9, Bedford & Fowler: Statics § 2.5

Taking the *dot product* of an arbitrary vector with a *unit vector* oriented along a coordinate direction yields the *projection* of the arbitrary vector along the coordinate axis. When this scalar (magnitude) is multiplied by a unit vector in the coordinate direction, the result is the vector component in that coordinate direction. This is one common use of the dot product. The other is finding the angle between two vectors.

The dot product (or scalar product) can be calculated in two ways:

- In trigonometric terms, the equation for a dot product is written as

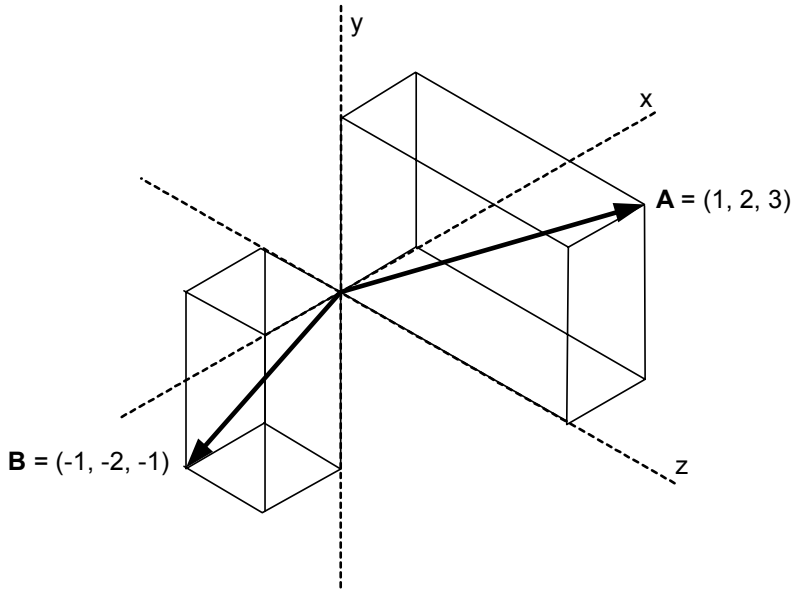
  $$\mathbf{A} \bullet \mathbf{B} = A\, B \cos(\theta)$$

  Where θ is the angle between arbitrary vectors **A** and **B**.

- In matrix form, the equation is written

  $$\mathbf{A} \bullet \mathbf{B} = A_x\, B_x + A_y\, B_y + A_z\, B_z$$

MATLAB provides a dot product function, dot() to automatically perform the calculations required by the matrix form of the dot product. If you have two vectors written in matrix form, such as



Then **A•B** is the projection of **A** onto **B** (a magnitude, or scalar). Using MATLAB, the dot product is calculated like this (bold letters have not been used for matrix names in MATLAB):

```
» A = [ 1  2  3];
» B = [-1 -2 -1];
» dot(A,B)
ans =
   -8
```

To verify this result, we can do the math term by term…

```
» x = 1;      y = 2;      z = 3;                % coordinate index definitions
» A(x) * B(x) + A(y) * B(y) + A(z) * B(z)
ans =
   -8
```

And we can use the trig. form of the dot product to find the angle between the two vectors. First, calculate the magnitude of the **A** and **B** vectors…

```
» A_mag = sqrt( A(x)^2 + A(y)^2 + A(z)^2 )
A_mag =
   3.7417
» B_mag = sqrt( B(x)^2 + B(y)^2 + B(z)^2 )
B_mag =
   2.4495
```

Then find the angle between the vectors using MATLAB's acos() function. The factor 180/pi has been included to convert the angle to degrees.

```
» theta = 180/pi * acos( dot(A,B)/( A_mag * B_mag ) )
theta =
  150.7941
```

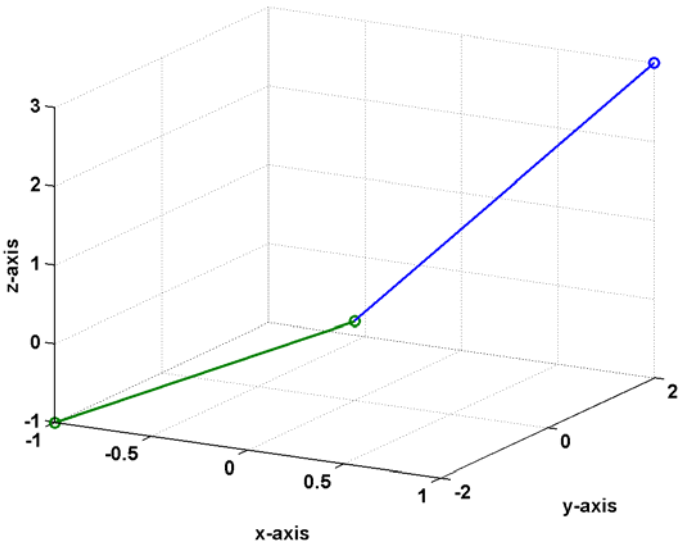
Finally, you can use a 3-d scatter plot in MATLAB to help you visualize these vectors, to see if your computed results make sense. To create the 3-d scatter plot, you must define the x, y, and z coordinates of the endpoints of each vector.

```
» X = [ 0   0;  1  -1];      Y = [ 0   0;  2  -2];      Z = [ 0   0;  3  -1];
```

Here, the top-left elements form the starting coordinates for the **A** vector: (0, 0, 0). The lower-left elements for the end coordinates for the **A** vector: (1, 2, 3). Similarly, the right columns in each matrix represent the starting (0, 0, 0) and ending (-1, -2, -1) coordinates of the **B** vector.

To create the graph, issue the following statements to the MATLAB command window.

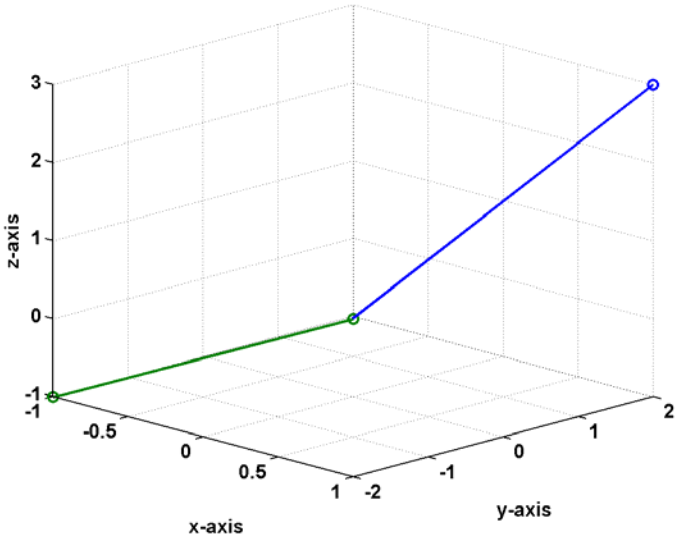
```
» plot3(X,Y,Z, '-o')  % Creates plot
» grid                % Draws gridlines
» xlabel('x-axis')    % Labels x-axis
» ylabel('y-axis')    % Labels y-axis
» zlabel('z-axis')    % Labels z-axis
» view(29,20);        % Rotates graph
```



Note that the axes do not intercept at (0, 0, 0). Actually, the middle point on the plot is at (0, 0, 0). The '-o' option in the plot3 function is a line specification that determines line type, and marker symbol of the plotted lines. Here, the '-' represents a solid line and the 'o' is used for circular marker symbols.

The usefulness of the 3-d plot for visualization is the ability to rotate the graph to see how the plot looks from different angles. To rotate the plot, use the menu commands: Tools/Rotate 3D, then simply click inside the plot area, hold the left mouse button down, and drag the mouse pointer around the plot. The curve will respond to the location of the pointer.

In the graph below, the plot has been rotated until the plane of the vectors is approximately parallel with the page. In this view, the calculated angle of 150° looks to be about right for these vectors.



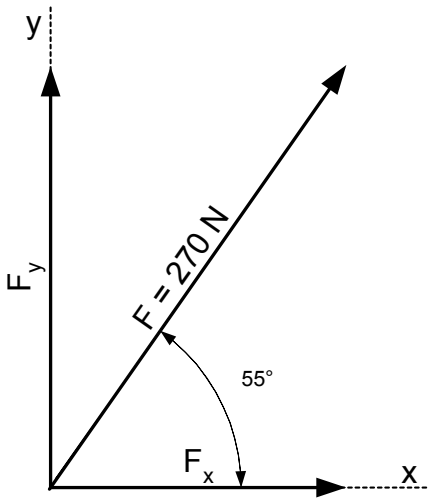
Annotated MATLAB Script Solution

```
% Define the vectors
A = [ 1  2  3];
B = [-1 -2 -1];

% Take the dot product
dot(A,B)

% Check MATLAB's dot product operator by calculating the dot product
% explicitly...
x = 1;   y = 2;   z = 3;              % coordinate index definitions

A(x) * B(x) + A(y) * B(y) + A(z) * B(z)

% To calculate the angle between the vectors, first need the magnitude
% of each vector.
A_mag = sqrt( A(x)^2 + A(y)^2 + A(z)^2 );
B_mag = sqrt( B(x)^2 + B(y)^2 + B(z)^2 );

% Then calculate the angle
theta = 180/pi * acos( dot(A,B)/( A_mag * B_mag ) );

% To visualize the vectors, create matrices describing the starting and
% ending coordinates of each vector.
X = [ 0  0
      1 -1];

Y = [ 0  0
      2 -2];

Z = [ 0  0
      3 -1];

% Then create a 3-d Scatter Plot
plot3(X, Y, Z, '-o')          % Creates plot
grid;                         % Draws gridlines
xlabel('\bf{x-axis}');        % Labels x-axis
ylabel('\bf{y-axis}');        % Labels y-axis
zlabel('\bf{z-axis}');        % Labels z-axis
view(29,20);                  % 3-D graph viewpoint specification
```

Example: Finding the Components of a Force Vector

A force acts on a fixed pin with a magnitude of 270 N (about 60 lb_f) at an angle of 55° from horizontal.

a. Find the horizontal and vertical components of the force using dot products.



Solution

To find the magnitude of the x-component of **F**, calculate the dot product of **F** and a unit vector in the x direction, $\mathbf{u}_x$.

$$F_x = F\cos(\theta) = \frac{\mathbf{F} \bullet \mathbf{u}_x}{u_x}$$

Note: Most texts do not show u_x in the denominator of the last term of this series of equalities, since the magnitude of a unit vector is known to be one.

```
» F = 270;
» theta = 55 * pi/180;                % 55 deg converted to radians
» F_x = F * cos( theta )
F_x =
  154.8656
```

Similarly, the vertical component of **F** is calculated as:

```
» F_y = F * cos( (90-55) * pi/180 )
F_y =
  221.1711
```

Alternatively, the **F** and $\mathbf{u}_x$ vectors can be written in vector form.

```
» F(1) = 270 * cos(55 * pi/180);
» F(2) = 270 * cos( (90-55) * pi/180 );
» F(3) = 0;
» u_x  = [ 1   0   0 ];
```

And the dot product can be used to calculate the horizontal component of **F**.

```
» F_x = dot( F, u_x )
F_x =
  154.8656
```

Similarly, we can define a unit vector in the y-direction, and calculate the vertical component of the force.

```
» u_y  = [ 0   1   0 ];
» F_y = dot( F, u_y )
F_y =
  221.1711
```

3 Equilibrium of a Particle, Free-Body Diagrams

Ref: Hibbeler § 3.3, Bedford & Fowler: Statics § 3.2-3.3

When a body is either not moving (zero velocity), or moving at a constant velocity (speed and direction), the sum of the external forces on the body is zero, and the body is said to be in *equilibrium*.

$$\sum \mathbf{F} = 0$$

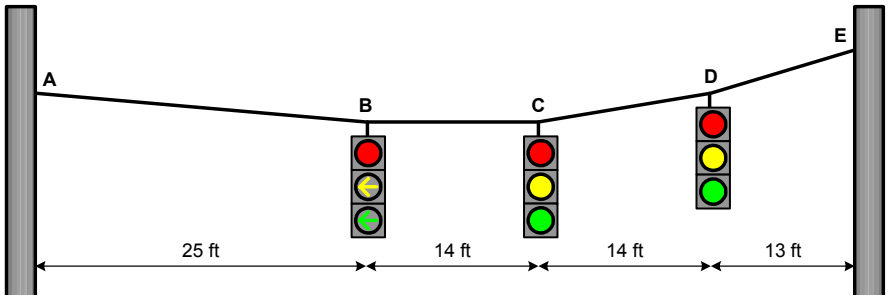
or, for two-dimensional equilibrium,

$$\sum F_x = 0$$
$$\sum F_y = 0$$

The picture showing the external forces acting on the object is called a *free-body diagram*. A free-body diagram is used to help solve problems both in statics and dynamics.

Example: Forces on Traffic Light Suspension Cables

Three traffic lights have been suspended between two poles at an intersection, as shown below.



Each of the three lights has a mass of 18 kg (approx. 40 lb). (Assume zero mass cables.) The tension in the cables has been adjusted such that the lights at points B and C are at the same height, and cable section AB is at an angle of 5° from horizontal.

a. Determine the downward force at points B, C, and D due to the mass of the lights.

b. Draw a free-body diagram at point B, and use it to find the vertical and horizontal components of force in cable AB.

c. Repeat part b. for the other lights, determining the force components in each cable section, and the angle of each cable section (measured from the +x direction).

Solution: Part a.

The mass of each traffic light is being acted on by gravity, so the force is calculated as

$$F_y = m\,g$$

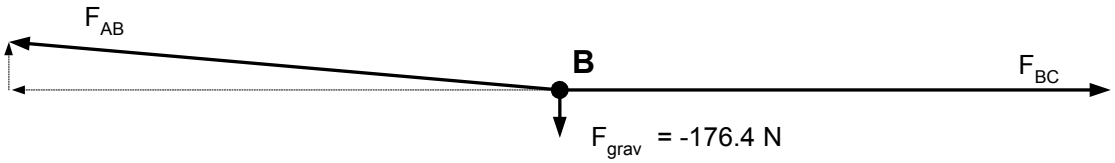
In MATLAB, this calculation looks like this:

```
» m = 18;                    %Mass of light (kg)
» g = 9.8;                   %Gravitational constant (m/s^2)
» F_grav = -m * g
F_grav =
 -176.4000
```

So the downward force exerted by each traffic light is 176.4 N. The minus sign has been included to show that the force is downward, i.e., in the –y direction.

If the lights are not moving up or down, the cable must apply an equal but oppositely directed force on the light, since the vertical force components must sum to zero if the body is in equilibrium.

Part b. – Free-Body Diagram for Light at B



Since cable section BC is horizontal, there is no vertical component of force in cable section BC. So, all of the weight of the traffic light at B must be carried by cable AB, more specifically, by the vertical component of force in cable AB.

```
» F_ABy = -F_grav
F_ABy =
  176.4000
```

The horizontal component of force in cable AB can be determined using the specified angle (cable AB is 5° from horizontal, or 175° from +x).

```
F_ABx = F_ABy / tan( 175 * pi/180)
F_ABx =
 -2016.3000
```

The horizontal component is 2016 N (about 450 lb_f) acting in the –x direction.

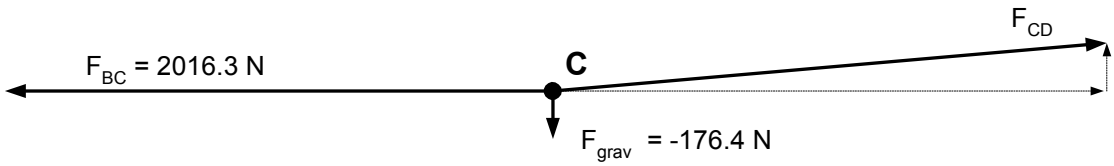
The force acting in the direction of the cable has a magnitude of 2024 N.

```
» F_AB  = sqrt( F_ABx ^ 2 + F_ABy ^ 2 )
F_AB =
  2024.0000
```

Finally, if the light at point B is not moving to the left or right, the sum of the horizontal forces acting on point B must be zero, so the horizontal component of force in section BC is +2016 N. Since there is no vertical component of force in section BC, this is also the total force on section BC at point B.

```
» F_BCx = -F_ABx;        % Since the sum of the x-components of force is zero
» F_BCy = 0;
» F_BC  = sqrt( F_BCx ^ 2 + F_BCy ^ 2 )
F_BC =
  2016.3000
```

Part b. – Free-Body Diagram for Light at C



The free body diagram for the light at point C is constructed using the following concepts:

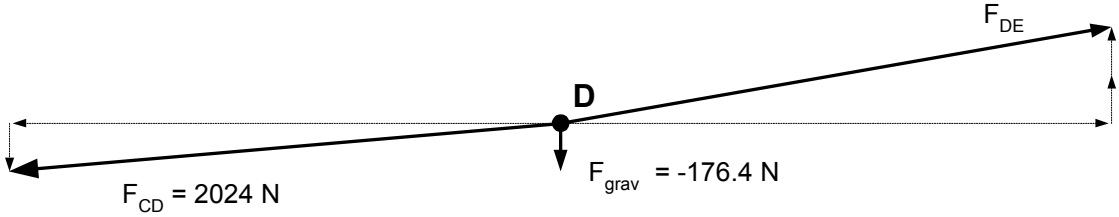
- If the light at point C is not moving left or right, then the horizontal component of force F_{CD} must be +2016 N.
- If the light at point C is not moving up or down, then the vertical component of force F_{CD} must be +176 N.

This is essentially the same as the free-body diagram at point B, just flipped left to right. So the magnitude of force F_{CD} is 2024 N, and acts at 5° from horizontal. This can be verified as follows:

```
» F_CDx = -F_BCx

F_CDx =

  2016.3000

» F_CDy = -F_grav

F_CDy =

  176.4000

» F_CD  = sqrt( F_CDx ^ 2 + F_CDy ^ 2 )

F_CD =

  2024.0000

» theta_CD = atan(F_CDy/F_CDx) * 180/pi

theta_CD =

    5.0000
```

Note: The value of F_{BC} used in this section is –2016.3 N, as shown in the free-body diagram for point C. See the Annotated MATLAB Script Solution to see how this sign change is handled for the complete problem solution.

Part b. – Free-Body Diagram for Light at D



The weight of the traffic light at point D exerts a downward force, F_{grav}, of 176.4 N. In addition, force F_{CD}, acting on point D, has a vertical component of 176.4 N, also acting downward. If the light at point D is not moving up or down, then these downward forces must be counterbalanced by the vertical component of force F_{DE}.

```
» F_DEy = -( F_grav + F_CDy )
F_DEy =
  352.8000
```

The horizontal component of force FDE must be equal to $-F_{CD\_x}$ if the light at point D is to be stationary.

```
F_DEx =
  2016.3000
```

Once the horizontal and vertical components are known, the angle (in degrees) of cable DE can be determined.

```
» theta_DE = atan(F_DEy/F_DEx) * 180/pi
theta_DE =
   9.9250
```

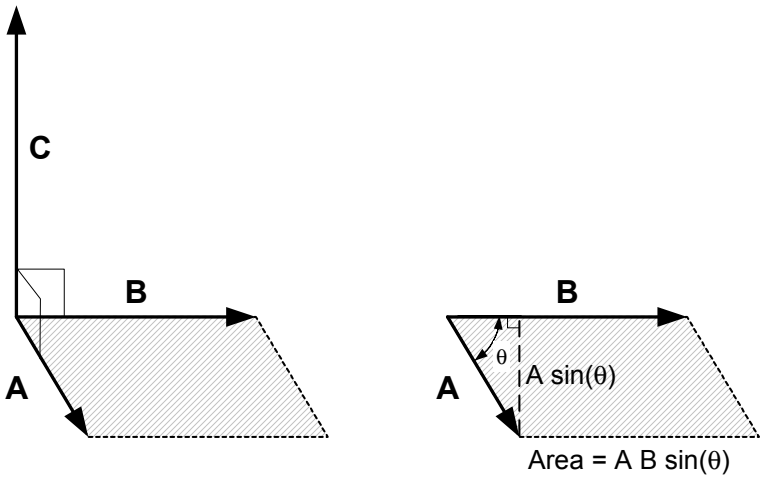
Annotated MATLAB Script Solution

```
%Traffic Light Suspension Cable
%
% Part a.
m = 18;                                 %Mass of light (kg)
g = 9.8;                                %Gravitational constant (m/s^2)
F_grav = -m * g;                        %Force exerted by traffic light
fprintf('Part a.\n')
fprintf('m      = %8.1f kg\t\t g = %8.1f m/s^2\n',m,g);
fprintf('F_grav = %8.2f N\n\n',F_grav);


% Part b. - Light at Point B
F_ABy = -F_grav;                        %Vertical component of AB
F_ABx = F_ABy / tan( 175 * pi/180 );    %Horizontal component of AB
F_AB  = sqrt( F_ABx ^ 2 + F_ABy ^ 2 ); %Magnitude of AB
fprintf('Part b. - Light at Point B\n')
fprintf('F_ABx = %+8.1f N\t\t F_ABy = %+8.1f N\n',F_ABx,F_ABy)
fprintf('F_AB  = %+8.1f N\n\n',F_AB);

F_BCx = -F_ABx;                         %Horizontal component of BC
F_BCy = 0;                              %Vertical component of BC
F_BC  = sqrt( F_BCx ^ 2 + F_BCy ^ 2 ); %Magnitude of BC
fprintf('F_BCx = %+8.1f N\t\t F_BCy = %+8.1f N\n',F_ABx,F_ABy)
fprintf('F_BC  = %+8.1f N\n\n',F_BC);

% Part b. - Light at Point C
F_BCx = -F_BCx;         %F_BC acting on point C is in the -x direction
F_CDx = -F_BCx;                         %Horizontal component of CD
F_CDy = -F_grav;                        %Vertical component of CD
F_CD  = sqrt( F_CDx ^ 2 + F_CDy ^ 2 ); %Magnitude of CD
theta_CD = atan(F_CDy/F_CDx) * 180/pi; %Angle from horizontal force acts
fprintf('Part b. - Light at Point C\n')
fprintf('F_CDx = %+8.1f N\t\t F_CDy    = %+8.1f N\n',F_CDx,F_CDy)
fprintf('F_CD  = %+8.1f N\t\t theta_CD = %+8.3f deg\n\n',F_CD,theta_CD);

% Part b. - Light at Point D
F_CDx = -F_CDx;         %F_CD acting on point d is in the -x direction
F_CDy = -F_CDy;         %F_CD acting on point C is in the -y direction
F_DEy = -( F_grav + F_CDy );            %Vertical component of DE
F_DEx = -F_CDx;                         %Horizontal component of DE
F_DE  = sqrt( F_DEx ^ 2 + F_DEy ^ 2 ); %Magnitude of DE
theta_DE = atan(F_DEy/F_DEx) * 180/pi; %Angle from horizontal force acts
fprintf('Part b. - Light at Point D\n')
fprintf('F_DEx = %+8.1f N\t\t F_DEy    = %+8.1f N\n',F_DEx,F_DEy)
fprintf('F_DE  = %+8.1f N\t\t theta_DE = %+8.3f deg\n\n',F_DE,theta_DE);
```

4 Cross Products and Moments of Force

Ref: Hibbeler § 4.2-4.3, Bedford & Fowler: Statics § 2.6, 4.3

In geometric terms, the *cross product* of two vectors, **A** and **B**, produces a new vector, **C**, with a direction perpendicular to the plane formed by **A** and **B** (according to right-hand rule) and a magnitude equal to the area of the parallelogram formed using **A** and **B** as adjacent sides.



The cross product is used to find the moment of force. An example of this will be shown after describing the basic mathematics of the cross product operation.

The cross product (or vector product) can be calculated in two ways:

- In trigonometric terms, the equation for a dot product is written as

  $$\mathbf{C} = \mathbf{A} \times \mathbf{B} = A\, B \sin(\theta)\, \mathbf{u}_C$$

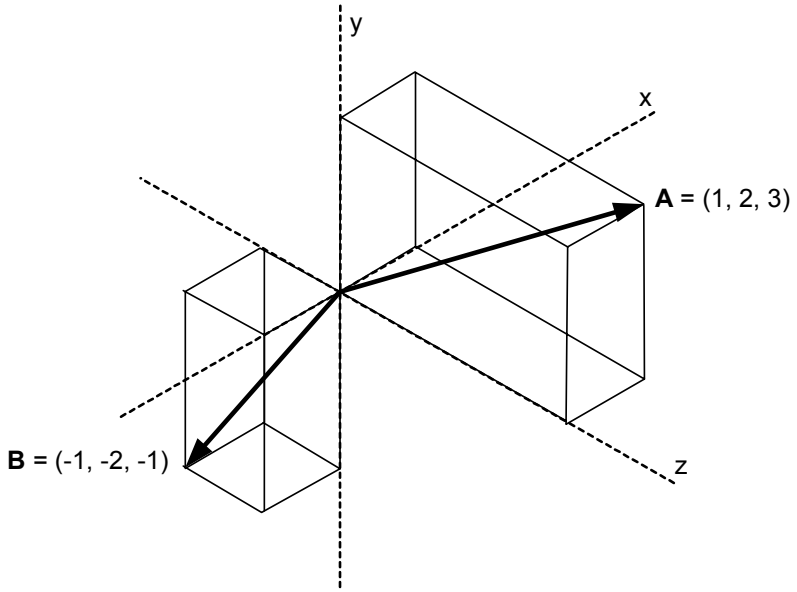
  Where θ is the angle between arbitrary vectors **A** and **B**, and $\mathbf{u}_C$ is a unit vector in the direction of **C** (perpendicular to **A** and **B**, using right-hand rule).

- In matrix form, the equation is written in using components of vectors **A** and **B**, or as a determinant. Symbols **i**, **j**, and **k** represent unit vectors in the coordinate directions.

  $$\mathbf{A} \times \mathbf{B} = \left(A_y B_z - A_z B_y\right)\mathbf{i} - \left(A_x B_z - A_z B_x\right)\mathbf{j} + \left(A_x B_y - A_y B_x\right)\mathbf{k}$$

  $$= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix}$$

MATLAB provides a cross product function to automatically perform the calculations required by the matrix form of the dot product. If you have two vectors written in matrix form, such as



Then **A**×**B** can be calculated like this (bold letters have not been used for matrix names in MATLAB):

```
» A = [1   2  3];
» B = [-1 -2 -1];
» A_x_B = cross(A,B)              %Uses MATLAB's cross product function
A_x_B =
    4    -2     0
```

To verify this result, we can do the math term by term…

```
» x = 1;  y = 2;  z = 3;                %Coordinate index definitions
» [  A(y)*B(z)-A(z)*B(y)     -(A(x)*B(z)-A(z)*B(x))     A(x)*B(y)-A(y)*B(x)  ]
ans =
    4    -2     0
```

Or, we can use the trig. form of the cross product. First, we calculate the norm, or magnitude, of the **A** and **B** vectors using the norm function in MATLAB…

```
» A_mag = norm(A)
A_mag =
   3.7417
» B_mag = norm(B)
B_mag =
   2.4495
```

Then find the angle between vectors **A** and **B** using MATLAB's acos() function.

```
» theta = 180/pi * acos(dot(A,B)/(A_mag * B_mag))
theta =
  150.7941
```

The magnitude of the **C** matrix can then be calculated…

```
» C_mag = A_mag * B_mag * sin(theta * pi/180)
C_mag =
   4.4721
```

The direction of **C** is perpendicular to the plane formed by **A** and **B**, and is found using the cross product. To obtain the direction cosines of C, divide the cross product of A and B by its magnitude.

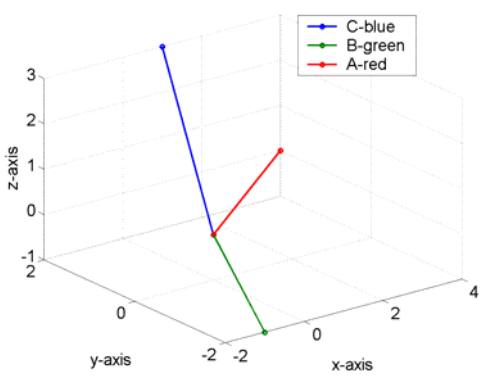
```
» cross(A,B)/norm(cross(A,B))
ans =
   0.8944   -0.4472         0
» alpha = acos(0.8944) * 180/pi          %from x+
alpha =
   26.5685
» beta  = acos(-0.4472) * 180/pi         %from y+
beta =
  116.5642
» gamma = atan(0) * 180/pi               %from z+
gamma =
     0
```

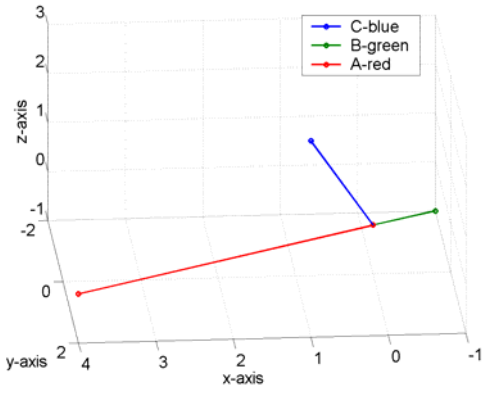
This is, of course, equivalent to

```
» C = cross(A,B);
» C/C_mag
ans =
   0.8944   -0.4472         0
```

The vectors can be graphed to see how the cross product works. The plot on the left shows the original plot, with the axes oriented in the same way as the drawing on the second page. In the plot on the right the axes have been rotated to show that vector **C** is perpendicular to the plane formed by vectors **A** and **B**.

```
x = [ 0  0 0;          y = [ 0  0 0;          z = [ 0  0 0;
      1 -1 4];               2 -2 2];               3 -1 0];
```





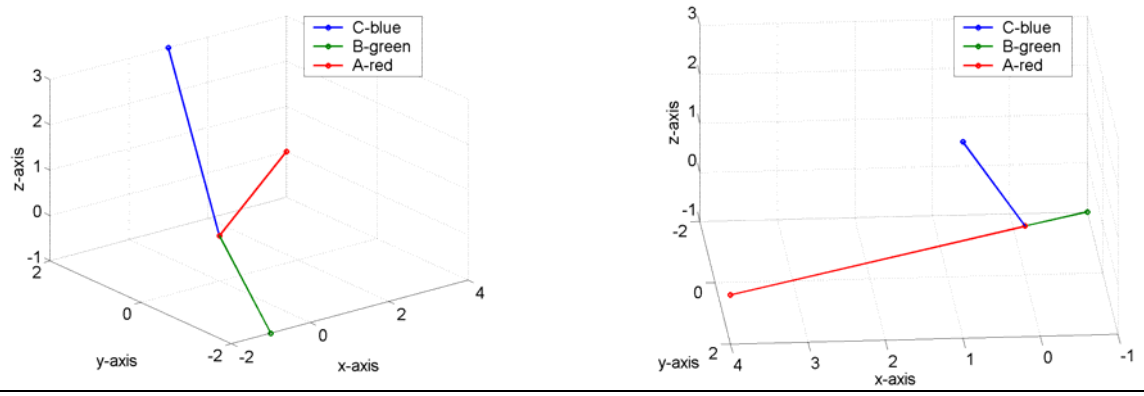
Annotated MATLAB Script Solution

```
%Define the vectors
A = [1   2  3];
B = [-1 -2 -1];

%Take the cross product
A_x_B = cross(A,B);
fprintf('A x B = [ %1.4f %1.4f %1.4f]\n\n', A_x_B)

%Check MATLAB's cross product operator by calculating the cross product
explicitly...
x = 1;  y = 2;  z = 3;  %  Define coordinate index definitions
A_x_B_exp = [A(y)*B(z)-A(z)*B(y) -(A(x)*B(z)-A(z)*B(x)) A(x)*B(y)-A(y)*B(x)];
fprintf('A x B calculated explicitly= [ %1.4f %1.4f %1.4f]\n\n', A_x_B_exp)
```

```
%Use the trigonometric form of the cross product operator to find the
magnitude of the C vector.
%  First, find the magnitude of the A & B vectors using the norm function.
A_mag = norm(A);
B_mag = norm(B);
fprintf('Magnitude of vector A = %1.4f \n', A_mag)
fprintf('Magnitude of vector B = %1.4f \n', B_mag)
%  Then, find the angle between vectors A and B.
theta = 180/pi * acos(dot(A,B)/(A_mag * B_mag));
fprintf('Angle between vectors A and B = %1.4f deg\n', theta)
%  Finally, solve for the magnitude of the C vector.
C_mag = A_mag * B_mag * sin(theta * pi/180);
fprintf('Magnitude of vector C = %1.4f \n\n', C_mag)
```

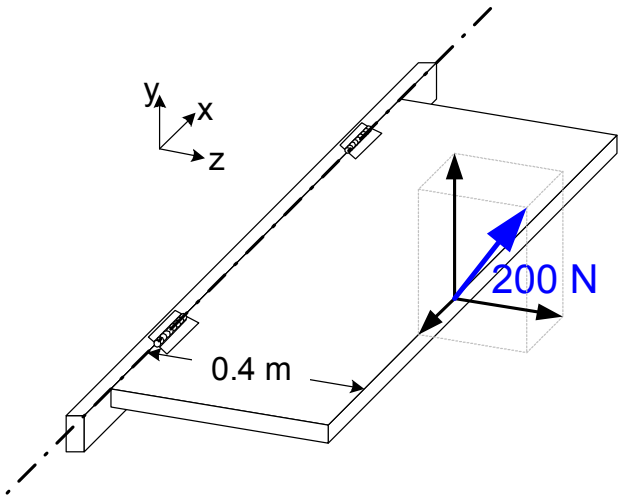
```
%Solve for the direction of the C vector
cross(A,B)/norm(cross(A,B)); % or C/C_mag where C = cross(A,B)

alpha = acos(0.8944) * 180/pi;
beta  = acos(-0.4472) * 180/pi;
gamma = atan(0) * 180/pi;
fprintf('alpha = %1.4f deg from +x\n', alpha)
fprintf('beta  = %1.4f deg from +y\n', beta)
fprintf('gamma = %1.4f deg from +x\n', gamma)
```

```
%Plot the A, B, and C Vectors
x = [ 0  0 0;
      1 -1 4];

y = [ 0  0 0;
      2 -2 2];

z = [ 0  0 0;
      3 -1 0];

plot3(x,y,z,'-o','LineWidth',2,'MarkerSize',5);
set(gca,'FontSize',18)
grid;
xlabel('x-axis');
ylabel('y-axis');
zlabel('z-axis');
legend('C-blue','B-green', 'A-red',2);
```



Example: Find the Moment of a Force on a Line

A force of **F** = 200 N acts on the edge of a hinged shelf, 0.40 m from the pivot point.



The 200 N force has the following components:

F_x = -40 N

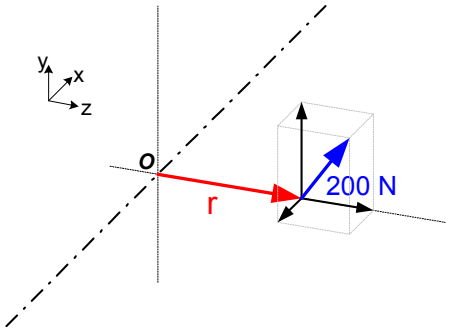
F_y = 157 N

F_z = 118 N

Only the y-component of **F** will tend to cause rotation on the hinges. What is the moment of the force about the line passing through the hinges (the x axis)?

Solution Using the Cross Product

We begin by defining the vector **r** which starts at the x axis (the line through the hinges) and connects to the point at which force **F** acts.



Since the shelf was 0.40 m wide, **r** has a magnitude of 0.40, is oriented in the +z direction, and can be written in component form as

```
» r = [0 0 0.4];
» F = [-40 157 118];
```

The moment of force **F** about the point O is found using the cross product of **r** with **F**.

```
» M_o  = cross(r,F)
M_o =
  -62.8000  -16.0000         0
» M_mag = norm(M_o)
```

```
M_mag =

   64.8062
```

However, the moment of force **F** about the x axis requires an additional dot product with a unit vector in the x-direction, and is found as

```
» u_x = [1 0 0];

» M_L = dot(u_x, cross(r,F))

M_L =

  -62.8000
```

The minus sign indicates that the moment is directed in the –x direction.

Solution Using Scalars

The moment of force **F** about the x axis can also be determined by multiplying the y-component of **F** and the perpendicular distance between the point at which **F** acts and the x axis.

$$M_L = F_y\, d$$

Since the component of **F** in the y-direction is known (157 N), and the perpendicular distance is 0.4 m, the moment can be calculated from these quantities.

```
» d = 0.40;

» M_L = F(2) * d

M_L =

   62.8000
```

The direction must be determined using the right-hand rule, where the thumb indicates the direction when the fingers are curled around the x axis in the direction of the rotation caused by F_y.

Annotated MATLAB Script Solution

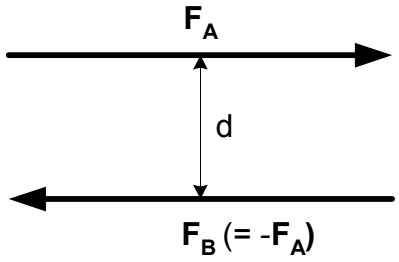
```
%Define the vectors
r = [0 0 0.4];
F = [-40 157 118];

%Take the cross product of r with F to get the moment about point 0 (the
origin);
M_o  = cross(r,F);
M_mag = norm(M_o);
fprintf('M_o = r x F = [ %1.4f %1.4f %1.4f]\n', M_o)
fprintf('M_mag = |M_o| = %1.4f\n', M_mag)
%Note:  This is not the solution to the stated question.
%       The question asks for the moment about the x axes.
%       That will be calculated next

%Declare a unit vector in the x-direction in order to calculate the moment
about the x axis.
u_x = [1 0 0];
```

```
%Calculate the moment about the x axis
M_L = dot(u_x, cross(r,F));
fprintf('Moment about the x axis = %1.4f\n', M_L)

%Check the results using scalar math
d = r(3);
M_L = F(2) * d;
fprintf('Moment about the x axis (with scalar math) = %1.4f\n', M_L)
```

5 Moment of a Couple

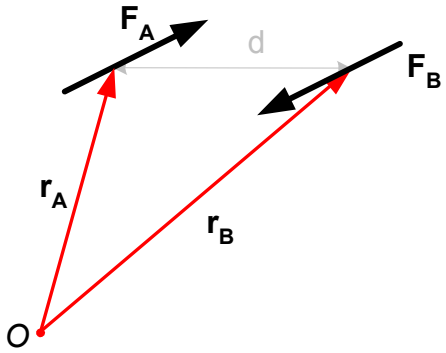
Ref: Hibbeler § 4.6, Bedford & Fowler: Statics § 4.4

A *couple* is a pair of forces, equal in magnitude, oppositely directed, and displaced by perpendicular distance, d.



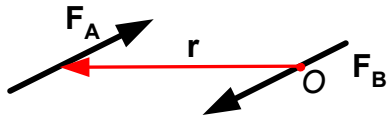
Since the forces are equal and oppositely directed, the resultant force is zero. But the displacement of the force couple (d) does create a *couple moment*.

The moment, **M**, about some arbitrary point *O* can be calculated.



$$\mathbf{M} = \mathbf{r}_A \times \mathbf{F}_A + \mathbf{r}_B \times \mathbf{F}_B$$
$$= \mathbf{r}_A \times \mathbf{F}_A + \mathbf{r}_B \times (-\mathbf{F}_A)$$

If point *O* is placed on the line of action of one of the forces, say $\mathbf{F}_B$, then that force causes no rotation (or tendency toward rotation) and the calculation of the moment is simplified.

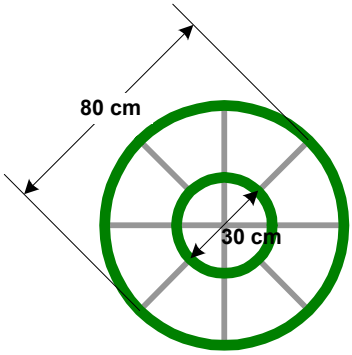


$$\mathbf{M} = \mathbf{r} \times \mathbf{F}_A$$

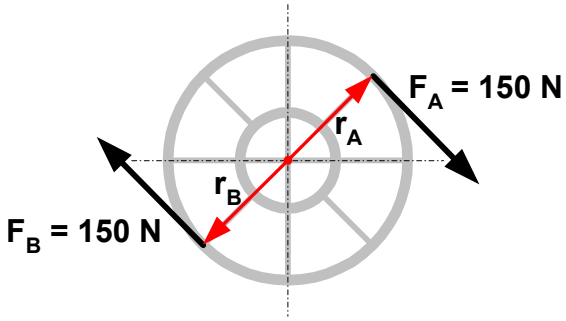
This is a significant result: The couple moment, **M**, depends only on the position vector **r** between forces $\mathbf{F}_A$ and $\mathbf{F}_B$. The couple moment does not have to be determined relative to the location of a point or an axis.

Example A: Moment from a Large Hand Wheel

The stem on a valve has two hand wheels: a small wheel (30 cm diameter) used to spin the valve quickly as it is opened and closed, and a large wheel (80 cm diameter) that may be used to free a stuck valve, or seat the valve tightly when it is fully closed.



If the operator can impose a force of 150 N on each side of the large wheel (a force couple), what moment is imposed on the valve stem?



Solution 1

As drawn, both the force and position vectors have x- and y-components. The vectors may be defined as:

```
» F_A = [ 106.06 -106.06 0];          %Newtons
» r_A = [ 28.284  28.284 0];          %centimeters
» F_B = [-106.06  106.06 0]           %Newtons
» r_B = [-28.284 -28.284 0];          %centimeters
```

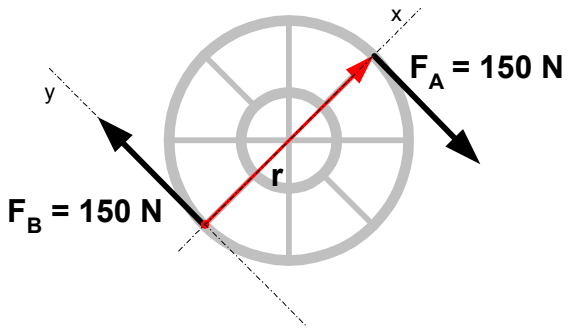
The moment of the couple can be calculated using the cross product operator on the Matrix toolbar.

```
» M = cross(r_A,F_A) + cross(r_B,F_B);     %Newton centimeters
» M = M/100                                %Newton meters
M =
     0     0  -120.0000
```

The result is a couple moment of 120 N·m directed in the –z direction (into the page).

Solution 2

Perhaps a more reasonable positioning of the axes for this problem might look like this:



In this case, the vector definitions become:

```
» F_A = [  0 -150 0];                    %Newtons
» r   = [ 80    0 0];                        %centimeters
```

Since the position vector **r** originates from the line of action of force $\mathbf{F}_B$, $\mathbf{F}_B$ does not contribute to the moment. The moment is then calculated as

```
» M = cross(r,F_A);                      %Newton centimeters
» M = M/100                              %Newton meters
M =
    0     0          -120
```

The result is, of course, the same no matter how the axes are situated.

Solution 3: Using Scalars

Finally, the problem can also be solved using a scalar formulation. The perpendicular distance between the forces is 80 cm. With axes established as in Solution 2, the moment can be calculated as

```
» F = 150;                                   %Newtons
» d = 80;                                    %centimeters
» M= d * F;                                  %Newton centimeters
» M = M/100                                  %Newton meters
M =
  120
```

The direction must be determined using the right-hand rule.

Annotated MATLAB Script Solution

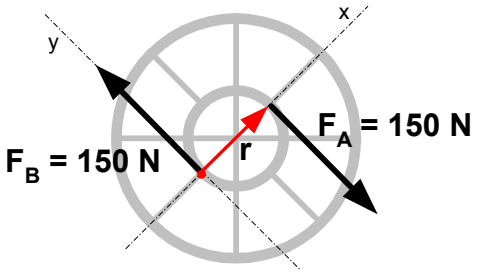
```
%Solution 1
%
%Define the vectors
F_A = [ 106.06 -106.06 0];              %Newtons
r_A = [ 28.284  28.284 0];              %centimeters
F_B = [-106.06  106.06 0];              %Newtons
r_B = [-28.284 -28.284 0];              %centimeters
%Compute the moment using MATLAB's cross product function
M = cross(r_A,F_A) + cross(r_B,F_B);    %Newton centimeters
%Convert to Newton meters
M = M/100;                              %Newton meters
fprintf('The resulting couple moment is = [ %1.2f  %1.2f  %1.2f ] (Nm)\n', M)
```

```
%Solution 2
%
%Define the vectors
F_A = [  0 -150 0];                     %Newtons
r   = [ 80    0 0];                     %centimeters
%Compute the moment
M = cross(r,F_A);                       %Newton centimeters
%Convert to Newton meters
M = M/100;                              %Newton meters
fprintf('The resulting couple moment is = [ %1.2f  %1.2f  %1.2f ] (Nm)\n', M)
```

```
%Solution 3
%
%Define the applied force and the perpendicular distance between the forces
F = 150;                                %Newtons
d = 80;                                 %centimeters
%Compute the moment
M= d * F;                               %Newton centimeters
%Convert to Newton meters
M = M/100;                              %Newton meters
fprintf('The resulting couple moment is = %1.2f (Nm)\n', M)
fprintf('  The direction must be determined using the right-hand rule.\n\n')
```

Example B: Moment from the Small Hand Wheel

The moment resulting from applying the same forces to the smaller hand wheel can be determined using any of the three solution procedures outlined above. Solution 2 is shown here.



```
%Example B: Moment from the Small Hand Wheel
%  The moment resulting from applying the same forces to the
%  smaller hand wheel can be determined using any of the three
%  solution procedures above.  Solution 2 is shown here.
%
%Define the vectors
F_A = [ 0 -150 0];                      %Newtons
r   = [ 30   0 0];                      %centimeters
%Compute the moments
M = cross(r,F_A);                       %Newton centimeters
%Convert to Newton meters
M = M/100;                              %Newton meters
fprintf('The resulting couple moment for the smaller wheel is = [ %1.2f …
         %1.2f %1.2f ] (Nm)\n', M)
```

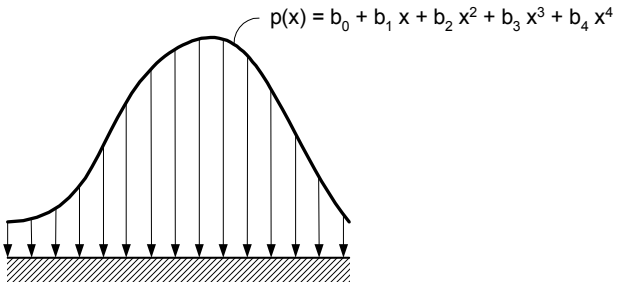
6 Reduction of a Simple Distributed Loading

Ref: Hibbeler § 4.10, Bedford & Fowler: Statics § 7.3

Loads are often not applied to specific points, but are *distributed* across a region. Design calculations can be simplified by finding a single equivalent force acting at a point; this is called *reduction of a distributed loading*. For loads distributed across a single direction (beam loading, for example) we need to find the both the magnitude of the equivalent force, and the position at which the equivalent force acts.

Example 1: Load Distribution Expressed as a Function of Position, x

The load on a beam is distributed as shown below:



where b_0 through b_4 are coefficients obtained by fitting a polynomial to data values (see Example 2). The x values range from 0.05 to 0.85 meters. The pressure is expressed in Pa. The values of the coefficients are tabulated below.

	b x10^{-4}
b_0	0.107
b_1	-1.68
b_2	11.9
b_3	-19.9
b_4	9.59

The data represent the pressure on the floor beneath a pile of 12-foot 2x4's, so y = 12 feet or 3.66 m.

Determine:

a) the two-dimensional loading function, w(x)

b) the magnitude and position of the equivalent force.

Solution 1

The two-dimensional loading function is obtained from the pressure function using the length of the boards, y = 3.66 m.

$$w(x) = p(x)\, y$$
$$= \left(b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4\right)(3.66)$$

The units on w are Pa·m.

The magnitude of the resultant force is obtained by integration.

$$F_R = \int_L w(x)\,dx$$

To perform this integration we first need to create an m-file for the loading function, w(x,y,b), and save it in the MATLAB search path. We then use the MATLAB quad() function to carry out the integration on the new function.

Note: For a full explanation of the quad() function type "help quad" at the command prompt.

```
function w = LoadingFunction(x, y, b)
%Saved as LoadingFunction.m in the MATLAB search path.

%Pressure Function (Pa)
p = b(1) + b(2).*x + b(3).*x.^2 + b(4).*x.^3 + b(5).*x.^4;

%Loading Function (Pa m)
w = p .* y;
```

```
» y = 3.66;                                                    % meters
» b = [ 1.07E3    -1.68E4    1.19E5    -1.99E5    9.59E4 ];
» F_R = quad('LoadingFunction', 0.05, 0.85, [ ], [ ], y, b)    %Newtons
F_R =
  6237
```

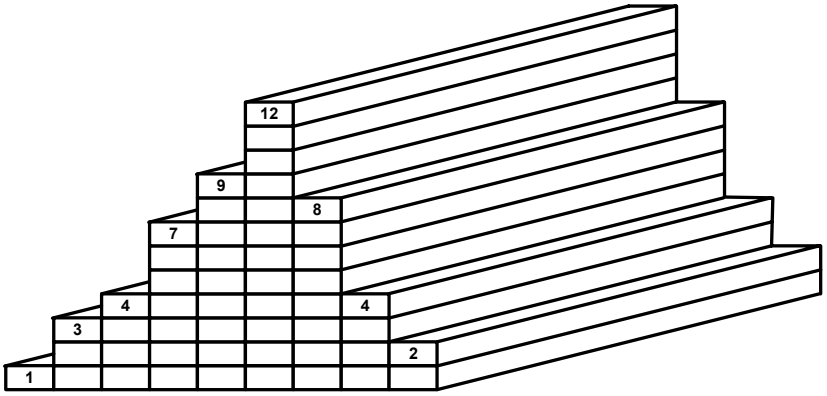
The location at which the resultant force acts is found by calculating the centroid of the area defined by w(x). We again need to create a new function to calculate the 1st moment of w(x,y,b).

```
function FM = FirstMoment(x, y, b)
%Saved as FirstMoment.m in the MATLAB search path.

%Pressure Function (Pa)
p = b(1) + b(2).*x + b(3).*x.^2 + b(4).*x.^3 + b(5).*x.^4;

%Loading Function (Pa m)
w = p .* y;

FM = w .* x;
```

```
» x_loc = quad('MeanForce', 0.05, 0.85, [ ], [ ], y, b)  ./  F_R    %meters
x_loc =
   0.4927
```

Example 2: Tabulated Load Distribution

The function used in Example 1 was obtained by polynomial regression of a set of data values, representing the pressure acting on the floor beneath a pile of 2x4 (inch, or 50x100 mm) lumber.



Using the density of wood (approximately 700 kg/m^3) and the lumber dimensions, the pressure exerted by the wood on the floor at the bottom of each stack of lumber was calculated (see table). The reported x value represents the position of the center of each column of boards.

x (m)	p (Pa)
0.05	343
0.15	1029
0.25	1372
0.35	2401
0.45	3087
0.55	4116
0.65	2744
0.75	1372
0.85	686

Use the tabulated pressure, position data to estimate the magnitude and position of the equivalent force.

Solution 2

w values can be calculated from pressure values, as before:

$$w = p \cdot y$$

```
» x = [ 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85];          %meters
» p = [  343 1029 1372 2401 3087 4116 2744 1372  686];          %Pascals
» w = p .* y                                                    %N/m
w =
  1.0e+004 *
  Columns 1 through 9
    0.1255    0.3766    0.5022    0.8788    1.1298    1.5065    1.0043    0.5022    0.2511
```

The integral for force is approximated as a summation:

$$F_R \approx \sum_{i=1}^{N} w_i \, \Delta x$$

where Δx is the width of a stack (0.10 m) and N is the number of stacks (9).

```
» delta_x = 0.10;                                   %meters
» F_R = sum(w .* delta_x)                           %Newtons
F_R =
  6277
```

Similarly, the centroid is approximated as follows:

```
» x_loc = sum(x .* w .* delta_x) / F_R              %meters
x_loc =
   0.4900
```

Annotated MATLAB Script Solution

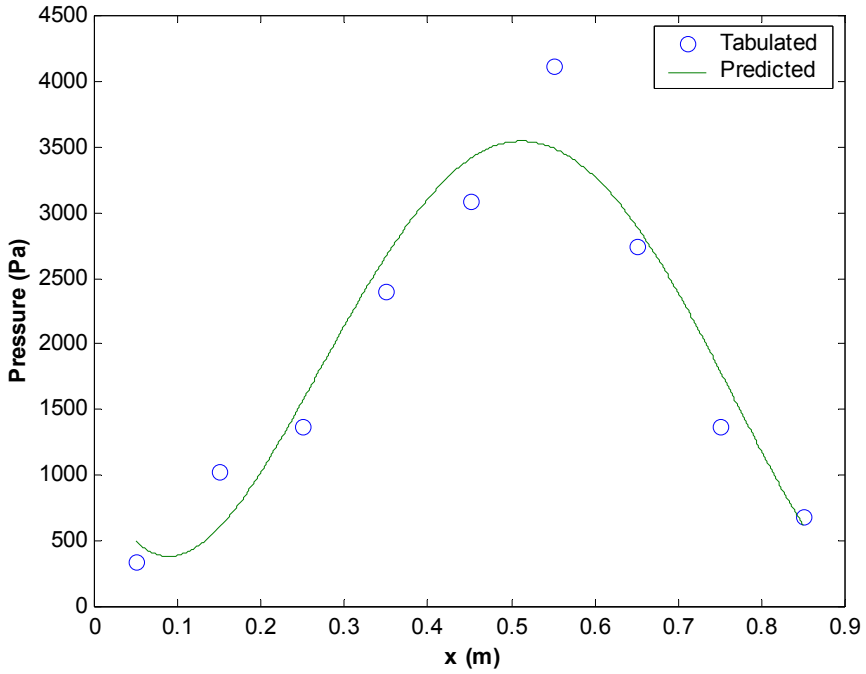
```
function w = LoadingFunction(x, y, b)
%Saved as LoadingFunction.m in the MATLAB search path.

%Pressure Function (Pa)
p = b(1) + b(2).*x + b(3).*x.^2 + b(4).*x.^3 + b(5).*x.^4;

%Loading Function (Pa m)
w = p .* y;
```

```
function mean = FirstMoment(x, y, b)
%Saved as FirstMoment.m in the MATLAB search path.

%Pressure Function (Pa)
p = b(1) + b(2).*x + b(3).*x.^2 + b(4).*x.^3 + b(5).*x.^4;

%Loading Function (Pa m)
w = p .* y;

mean = w .* x;
```

```
%Length of Boards (meters)
y = 3.66;

%Coefficients obtained by fitting a polynomial to data values
b = [ 1.07E3 -1.68E4 1.19E5 -1.99E5 9.59E4 ];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                           Solution 1                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The magnitude of the resultant force is calculated.
F_R = quad('LoadingFunction',0.05,0.85,[],[], y, b);
fprintf('\nThe magnitude of the resultant force is = %1.0f N\n', F_R)

%The location of the resultant force is calculated.
x_loc = quad(' FirstMoment',0.05,0.85,[],[], y, b) ./ F_R;
fprintf('The location of the resultant force is = %1.2f m\n', x_loc)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                  Solution 2 Using Tabulated Data                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x = [ 0.05 0.15 0.25 0.35 0.45 0.55 0.65 0.75 0.85]; %meters
p = [  343 1029 1372 2401 3087 4116 2744 1372  686]; %Pa

%Calculate w values from pressure values using the length of the boards.
w = p .* y;

%Calculate the magnitude of the resultant force (the use
%     of a summation instead of an integration
%     makes this an approximate result).
delta_x = 0.10; % meters
F_R = sum(w .* delta_x);
fprintf('\nThe approximated magnitude of the resultant force is = %1.0f N\n', F_R)

%Calculate the location of the resultant force
x_loc = sum(x .* w .* delta_x) / F_R;
fprintf('The approximated location of the resultant force is = %1.2f m\n', x_loc)
```

A Final Note

Both of these solutions are approximations. The second method approximates the integrals using summations, while the first method approximates the solution because the polynomial regression is a "best fit" but not a perfect fit to the data. In the graph shown here, the circles represent the tabulated values in Example 2, while the pressures predicted by the polynomial are indicated by the line.



7 Equilibrium of a Rigid Body

Ref: Hibbeler § 5.1-5.3, Bedford & Fowler: Statics § 5.1-5.2

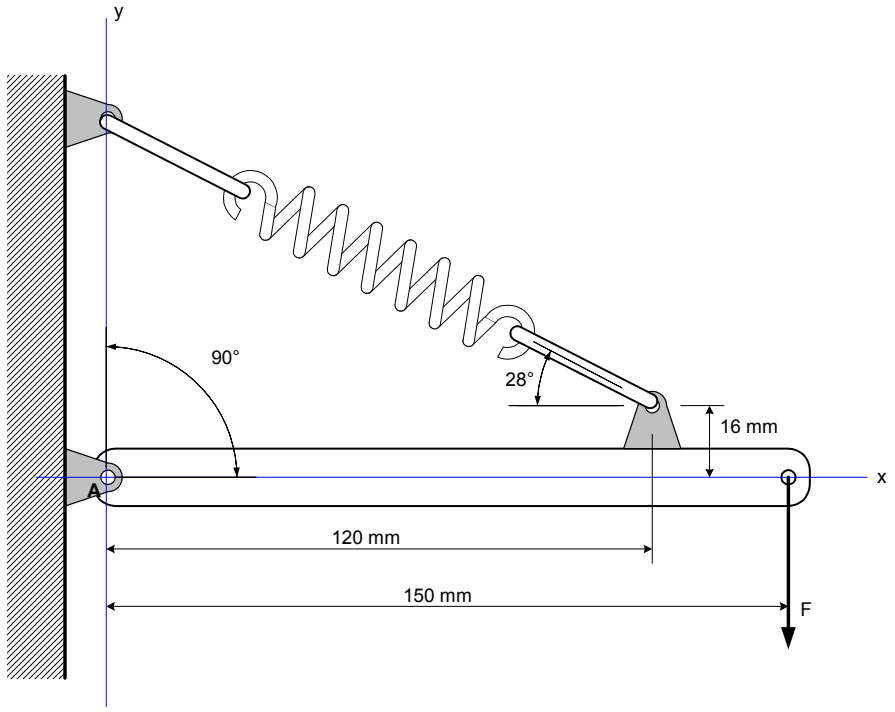
When forces are applied to *rigid bodies*, the conditions of *equilibrium* can be used to determine any unknown forces, and the *reactions* at the supports.

Example 1: Spring Support

A force, F, is applied to the end of a bar to stretch a spring until the bar is horizontal. If the spring constant is k = 600 N/m and the unstretched length of the spring assembly (spring and connecting links) is 75 mm, and ignoring the mass of the bar, determine:

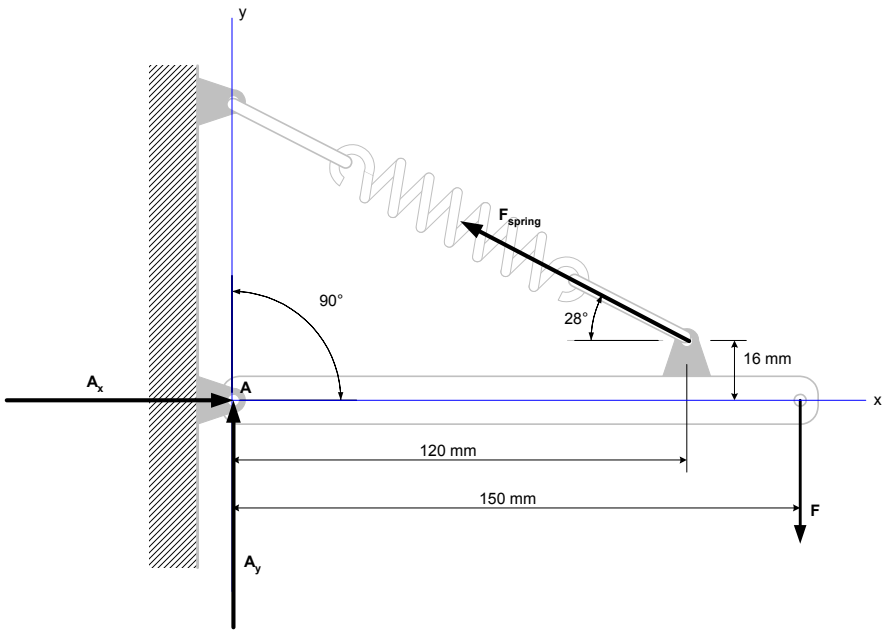
a) the magnitude of the applied force, F, and

b) the reactions at point A.



Solution

First, a free-body diagram is drawn.



We have assumed that the reactions at A will include both x and y components, and will calculate the actual values of A_x and A_y as part of the solution process.

First, a little trigonometry is required to obtain the extended length of the spring assembly.

```
» deg2rad = pi /180;                    %Conversion factor from degrees to radians
» L_ext = 120 / cos( 28 * deg2rad )     %mm
L_ext =
  135.9084
```

Then, we can calculate the actual extension of the spring…

```
» L_unstretched = 75;                   %mm
» L_stretch = L_ext - L_unstretched     %mm
L_stretch =
   60.9084
```

…and the spring force, F_{spring}.

```
» k_spring = 600;                       %N/m
» F_spring = k_spring * L_stretch / 1000    %Newtons
F_spring =
   36.5450
```

Next, we calculate the x and y components of the spring force, using the angle from the positive x axis (not just 28°) so that the direction of the force is accounted for.

```
» alpha = (180 - 28) * deg2rad;         %angle from +x axis
» Fsp_x = F_spring * cos(alpha)         %Newtons
Fsp_x =
  -32.2674
```

```
» Fsp_y = F_spring * sin(alpha)                    %Newtons
Fsp_y =
  17.1569
```

At equilibrium the sum of the moments at A must be zero. We can use this to solve for the applied force, F.

```
» F = -(abs(Fsp_x) * 16 + Fsp_y * 120) / 150      %Newtons
F =
 -17.1673
```

Note: The absolute value function was used on F_{sp_x} since the direction of rotation was accounted for as the equilibrium equation was written. Here, counter-clockwise rotation was assumed positive.

The equilibrium relationships for the x and y components of force can be used to determine the reactions at A. First, x-component equilibrium requires that the sum of the x components of force be zero. This relationship is used to solve for A_x.

```
» A_x = -Fsp_x                                    %Newtons
A_x =
  32.2674
```

Finally, the equilibrium relationship for the sum of the y components of force is used to calculate A_y.

```
» A_y = -Fsp_y – F                                %Newtons
A_y =
   0.0105
```

Annotated MATLAB Script Solution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Spring Support Problem                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Create conversion factor from degrees to radians
deg2rad = pi /180;

%Find the total extended lngth of the spring assembly.
L_ext = 120 / cos( 28 * deg2rad );           %mm
fprintf('\nTotal extended length = %1.0f mm\n', L_ext)

%Calculate the spring's extension.
L_unstretched = 75;                          %mm
L_stretch = L_ext - L_unstretched;           %mm
fprintf('Spring''s extension = %1.0f mm\n\n', L_stretch)

%Calculate spring force.
k_spring = 600;                              %N/m
F_spring = k_spring * L_stretch  / 1000;     %Newtons
fprintf('Spring force = %1.1f N\n\n', F_spring)

%Find x and y components of spring force.
alpha = (180 - 28) * deg2rad;                %angle from +x axis
Fsp_x = F_spring * cos(alpha);               %Newtons
Fsp_y = F_spring * sin(alpha);               %Newtons
fprintf('x-component of spring force = %1.1f N', Fsp_x)
fprintf('\t<= acting in -x direction\n')
fprintf('y-component of spring force = %1.1f N\n\n', Fsp_y)

%Use the sum of the moments about A to find applied force F
%We know => 16 |Fsp_x| + 120 Fsp_y+ 150 F = 0
%Solving for F
F = -(abs(Fsp_x) * 16 + Fsp_y * 120) / 150;  %Newtons
fprintf('Applied force = %1.1f N', F)
fprintf('\t<= acting to cause clockwise rotation\n\n')


%Use x and y component force balences to solve for the reactions at A
%We know => A_x + Fsp_x = 0
%Solving for A_x
A_x = -Fsp_x;                                %Newtons
fprintf('A_x =  %1.1f N', A_x)
fprintf('\t<= acting in +x direction\n')

%We know => A_y + Fsp_y +F  = 0
%Solving for A_y
A_y = -Fsp_y - F;                            %Newtons
fprintf('A_y =  %1.1f N\n', A_y)
```

8 Dry Friction

Ref: Hibbeler § 8.2, Bedford & Fowler: Statics § 9.1

To move a heavy crate, such as the one illustrated in the example below, you have to push on it. If you push hard enough to overcome *friction*, the crate will slide along the floor. But there is the danger that the crate will tip, especially if you push too high on the side of the crate. To better understand the calculations involved with this dry friction problem, and to see how you can check for tipping, we will solve the "sliding a crate" problem for several cases, varying the push height between 0.2 and 1 m, and the magnitude of the pushing force between 100 and 120 N.

Example: Pushing a Crate

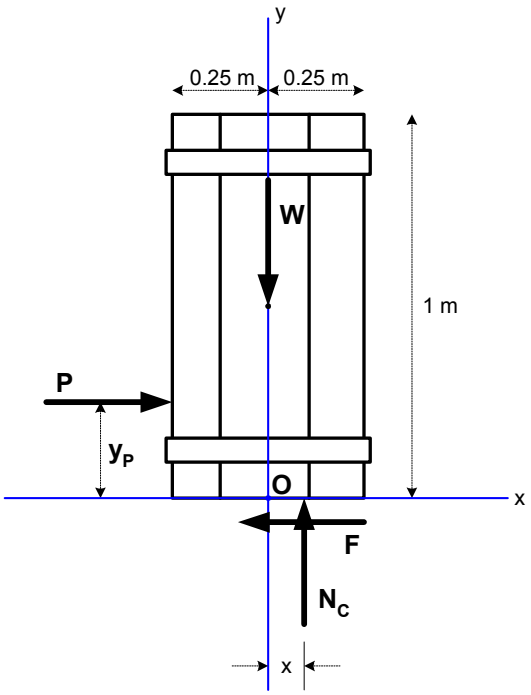
A force, P, is applied to the side of a crate at height y_P from the floor. The crate mass is 35 kg, and is 0.5 m wide and 1.0 m in height. The coefficient of static friction is 0.32. For each of the test cases tabulated below, determine:

a) the magnitude of the friction force, F,

b) the magnitude of the resultant normal force, N_C, and

c) the distance, x, (measured from the origin indicated in the illustration below) at which the resultant normal force acts.

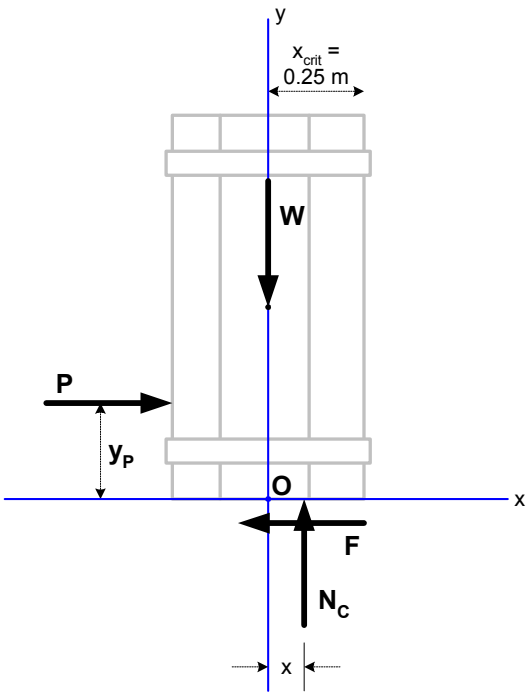
Once these results have been calculated, check for tipping and/or slipping.



Test Cases	P	y_P
1	100 N	0.2 m
2	100 N	0.4 m
3	100 N	0.8 m
4	100 N	1.0 m
5	120 N	0.2 m
6	120 N	0.4 m
7	120 N	0.8 m
8	120 N	1.0 m

Solution, Case 1

First, a free-body diagram is drawn.



To start the solution in MATLAB we begin by assigning the values given in the problem statement to variables.

```
» %Data from the problem statement...
» %      for specific case:
» Case = '1';
» P = 100;                                   %Newtons
» y_p = 0.2;                                 %meters
»
» %      for all cases:
» M = 35;                                    %kg
» x_crit = 0.25;                             %meters
» mu_s = 0.32;
» g = 9.807;                                 %meters/s^2
```

Then we calculate the weight of the crate.

```
» W = -M * g                                 %Newtons
W =
 -343.2450
```

Next, we use the equilibrium condition that the sum of the forces in the x direction must be zero. (This condition holds if the crate is not sliding or tipping.)

```
» % P + F = 0     so...
» F= -P                                          %Newtons
F =
  -100
```

Then, we use the equilibrium relationship for the y-components of force to determine the resultant normal force, N_C.

```
» % W + N_c = 0     so...
» N_c = -W                                       %Newtons
N_c =
  343.2450
```

The last equilibrium relationship, that the sum of the moments about O must be zero, can be used to determine the location at which N_C acts, called x in the illustration above.

```
» % -P * y_p + N_c * x = 0     so...
» x = P * y_p / N_c                              %meters
x =
    0.0583
```

At this point the equilibrium calculations are complete. We can solve for F_{max}, the maximum force that can be applied before overcoming friction. If the frictional force, F, exceeds F_{max}, then the crate would slip. (As it begins to slip, the equilibrium relationships are no longer valid.)

```
» if (abs(F) > F_max)
    fprintf(['Case ', Case, ' is slipping\n'])
  else
    fprintf(['Case ', Case, ' is NOT slipping\n'])
  end

  Case 1 is NOT slipping
```

Note: The absolute value function was used on F since both the magnitude and direction of F (the minus sign) were determined using the equilibrium relationship. Only the magnitude is used to test for slippage.

Finally, we can check for tipping by testing to see if the calculated x is beyond the edge of the crate (x_{crit}).

```
» if (x > x_crit)
    fprintf(['Case ', Case, ' is tipping\n'])
  else
    fprintf(['Case ', Case, ' is NOT tipping\n'])
  end

  Case 1 is NOT tipping
```

Note: The if statements are certainly not required here – the test could be performed by inspection – but it is convenient to let MATLAB do the checking for each of the eight cases.

From these tests we can see that a push of 100 N applied at a height of 0.2 m will not cause the crate to tip, but it will not cause the crate to move (slip), either.

Solution, Case 2

The only change between Case 1 and Case 2 is the height at which the push is applied. In Case 2 y_P = 0.4 m. This value is change at the beginning of the MATLAB script, and the script is simply re-executed to calculate the values for Case 2. The complete, annotated script solution for Case 2 is shown below.

Annotated MATLAB Script Solution – Case 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    Pushing a Crate Problem – Case 2                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Data from the problem statement...
%  for specific case:
Case = '1';
P = 100;                                          %Newtons
y_p = 0.4;                                        %meters

%  for all cases:
M = 35;                                           %kg
x_crit = 0.25;                                    %meters
mu_s = 0.32;
g = 9.807;                                        %meters/s^2

%Calculate the weight of the crate.
W = -M .* g;                                      %Newtons
fprintf('\nW = %1.1f N\t\t', W)

%EQUILIBRUM: The sum of the x components of force is zero.
%P + F = 0 so...
F = -P;                                           %Newtons
fprintf('F     = %1.1f N\t\t', F)

%EQUILIBRUM: The sum of the y components of force is zero.
%W + N_c = 0 so...
N_c = -W;                                         %Newtons
fprintf('N_c = %1.1f N\n', N_c)

%EQUILIBRUM: The sum of the moments about 0 is zero.
%-P * y_p + N_c * x = 0 so...
x = P .* y_p ./ N_c;                              %meters
fprintf('x = %1.3f m\t\t', x)

%Checking the Results...
F_max = mu_s .* N_c;                              %Newtons
fprintf('F_max = %1.1f N\n\n', F_max)

if (abs(F) > F_max)
   fprintf(['Case ', Case, ' is slipping\n'])
```

```
else
   fprintf(['Case ', Case, ' is NOT slipping\n'])
end

if (x > x_crit)
   fprintf(['Case ', Case, ' is tipping\n'])
else
   fprintf(['Case ', Case, ' is NOT tipping\n'])
end
```

Summary of Results for All Eight Cases

By repeatedly changing the assigned values for P and y_P in the MATLAB script, the results for all cases can be determined. Those results are summarized here. The value N_C = 343.2 N is not case dependent in this problem.

Test Cases	P	y_P	F	x	Slip?	Tip?
1	100 N	0.2 m	-100 N	0.058 m	No	No
2	100 N	0.4 m	-100 N	0.117 m	No	No
3	100 N	0.8 m	-100 N	0.233 m	No	No
4	100 N	1.0 m	-100 N	0.291 m*	No	Yes
5	120 N	0.2 m	-120 N	0.070 m	Yes	No
6	120 N	0.4 m	-120 N	0.140 m	Yes	No
7	120 N	0.8 m	-120 N	0.280 m*	Yes	Yes
8	120 N	1.0 m	-120 N	0.350 m*	Yes	Yes

* The calculated result for x has no meaning except to indicate that the crate is tipping if x > x_{crit}.

A faster approach to solving the problem is to define P and y_P as vectors and having MATLAB perform all the calculations simultaneously. By default MATLAB performs matrix calculations when using the *, /, and ^ operators. To force MATLAB to perform element-by-element operations place a period in front of the operator. To display the results we can then take advantage of the MATLAB for() statement to loop through the solution vectors and display the results by Case.

Annotated MATLAB Script Solution – All Cases

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 Pushing a Crate Problem  - All Cases                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Data from the problem statement...
%  for specific case:
Case = [ 1 2 3 4 5 6 7 8 ];
P   = [ 100 100 100 100 120 120 120 120]';   %Newtons
y_p = [ 0.2 0.4 0.8 1.0 0.2 0.4 0.8 1.0]';   %meters

%  for all cases:
M = 35;                                      %kg
x_crit = 0.25;                               %meters
mu_s = 0.32;
g = 9.807;                                   %meters/s^2

%Calculate the weight of the crate.
W = -M .* g;                                 %Newtons
```

```
%EQUILIBRUM: The sum of the x components of force is zero.
%P + F = 0 so...
F= -P;                                          %Newtons

%EQUILIBRUM: The sum of the y components of force is zero.
%W + N_c = 0 so...
N_c = -W;                                       %Newtons

%EQUILIBRUM: The sum of the moments about 0 is zero.
%-P * y_p + N_c * x = 0 so...
x = P .* y_p ./ N_c;                            %meters

%Display and Check the Results...
F_max = mu_s .* N_c;                            %Newtons
fprintf('F_max = %1.1f N\t', F_max)
fprintf('\nW = %1.1f N\t', W)
fprintf('N_c = %1.1f N\n\n', N_c)

%Display results for each case.
for i = 1 : length(Case)
   fprintf('\nCASE %1.0f:\n',Case(i))
   fprintf('P =  %1.1f N\t\t', P(i))
   fprintf('y_p = %1.2f m\n', y_p(i))
   fprintf('F = %1.1f N\t\t', F(i))
   fprintf('x   = %1.3f m\n', x(i))

   if (abs(F(i)) > F_max)
      fprintf(['Case ', num2str(Case(i)), ' is slipping\n'])
   else
      fprintf(['Case ', num2str(Case), ' is NOT slipping\n'])
   end

   if (x > x_crit)
      fprintf(['Case ', num2str(Case(i)), ' is tipping\n'])
   else
      fprintf(['Case ', num2str(Case(i)), ' is NOT tipping\n'])
   end
end
```

9 Finding the Centroid of Volume

Ref: Hibbeler § 9.2, Bedford & Fowler: Statics § 7.4

The *centroid of volume* is the geometric center of a body. If the density is uniform throughout the body, then the *center of mass* and *center of gravity* correspond to the centroid of volume. The definition of the centroid of volume is written in terms of ratios of integrals over the volume of the body.

$$\bar{x} = \frac{\int_V x \, dV}{\int_V dV} \qquad \bar{y} = \frac{\int_V y \, dV}{\int_V dV} \qquad \bar{z} = \frac{\int_V z \, dV}{\int_V dV}$$

Either analytical or numerical integration methods can be used to evaluate these integrals and compute the centroid of volume for the body.

The integrals over volume take slightly different forms depending on the coordinate system you use.

Cartesian Coordinates

$$\int_V dV = \int_{z_1}^{z_2} \int_{y_1}^{y_2} \int_{x_1}^{x_2} dx \, dy \, dz$$

Cylindrical Coordinates

$$\int_V dV = \int_{z_1}^{z_2} \int_{\theta_1}^{\theta_2} \int_{r_1}^{r_2} r \, dr \, d\theta \, dz$$

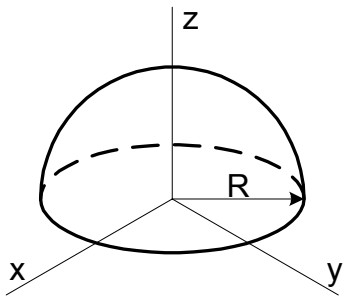
Spherical Coordinates

$$\int_V dV = \int_{\phi_1}^{\phi_2} \int_{\theta_1}^{\theta_2} \int_{r_1}^{r_2} r^2 \sin\phi \, dr \, d\theta \, d\phi$$

These integrals can be evaluated using analytical or numerical integration techniques.

Example: Centroid of a Hemisphere

Find the centroid of volume for a hemisphere of radius R = 7 cm.

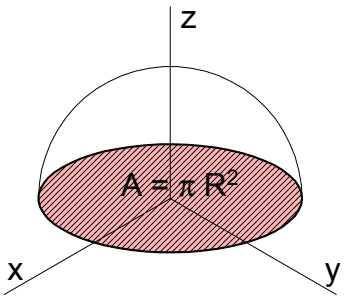


Note: This simple example will allow us to check our results against published values. For example, Hibbeler shows (inside back cover) that the centroid for a hemisphere resting on the plane formed by the x and y axes to be located at x = 0, y = 0, z = 3/8 R.

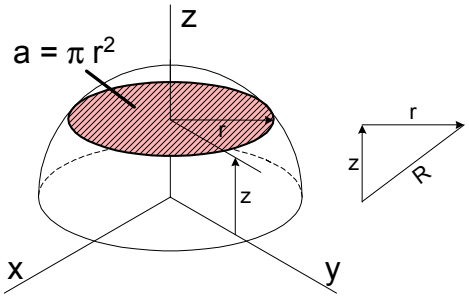
Solution: Numerical Integration

The volume of the hemisphere can be calculated by using the equation for the area of a circle, and integrating in only one direction, z.

To use this approach, first note that the area of the base is easily calculated, as $A = \pi R^2$.



The area of the circle formed by any x-y plane through the hemisphere is calculated as $a = \pi r^2$.



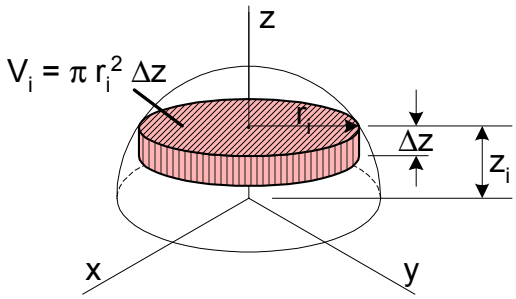
where the value of r depends on z. The relationship between r and z is readily determined, since r and z are two sides of a right triangle in which the hypotenuse is the radius of the hemisphere, R.

$$r = \sqrt{R^2 - z^2}$$

We then integrate the area of the circle from z = 0 to z = R.

$$V = \int_V dV = \int_{z=0}^{R} a\, dz = \int_{z=0}^{R} \left(\pi r^2\right) dz = \int_{z=0}^{R} \pi \left(R^2 - z^2\right) dz$$

To approximate the result using numerical integration methods, rewrite the integral as a sum and the dz as a Δz. Graphically (for the volume calculation in the denominator) this is equivalent to approximating the volume of the hemisphere by a series of stacked disks of thickness Δz. One such disk is shown in the figure below.



Here, the value of r at the current (point "i") value of z has been used to calculate the volume of the disk. Since R = 7 cm in this example, we might try a Δz of 1 cm, and calculate the volumes of seven disks (N = 7). For each disk, the value of z_i can be calculated using

$$z_i = i\,\Delta z$$

The value of the corresponding radius is then

$$r_i = \sqrt{R^2 - z_i^2}$$

The formula for volume can then be written in terms of z, as

$$V_i = \pi \left(R^2 - z_i^2\right)\Delta z$$

The sum of all of the volumes of all seven disks is an approximation of the volume of the hemisphere.

$$V \approx \sum_{i=1}^{N} V_i = \sum_{i=1}^{N} \pi \left[R^2 - (i\,\Delta z)^2\right]\Delta z$$

In MATLAB, this calculation looks like this:

```
» R  = 7;                                %Raidus
» N  = 7;                                %Number of intervals
» deltaZ = R/N;                          %Interval spacing
» %Sum (pi * (R^2 - (i*deltaZ)^2) * deltaZ) for i = 1,2,...,N
» V_predicted = 0;                       %Initialize predicted volume to zero
» for i = 1:N
   V_predicted = V_predicted + pi * (R^2 - (i*deltaZ)^2) * deltaZ;
end
» V_predicted
V_predicted =
  637.7433
```

We can see how good (or bad) the approximation is by calculating the actual volume of the hemisphere.

```
V_actual = 2/3 * pi * R^3
V_actual =
  718.3775
```

The approximation using only seven disks is not too good. If we use more disks, say N = 70, the approximation of the volume is quite a bit better.

```
» R  = 7;                                  %Raidus
» N  = 70;                                 %Number of intervals
» deltaZ = R/N;                           %Interval spacing
» %Sum (pi * (R^2 - (i*deltaZ)^2) * deltaZ) for i = 1,2,...,N
» V_predicted = 0;                        %Initialize predicted volume to zero
» for i = 1:N
   V_predicted = V_predicted + pi * (R^2 - (i*deltaZ)^2) * deltaZ;
end
» V_predicted
V_predicted =
  710.6440
```

To calculate the centroid using numerical methods, simply replace the ratio of integrals by the numerical approximation:

$$\bar{z} = \frac{\int_V z \, dV}{\int_V dV} \approx \frac{\sum_{i=1}^{N} z \, \pi \left[R^2 - (i \, \Delta z)^2\right] \Delta z}{\sum_{i=1}^{N} \pi \left[R^2 - (i \, \Delta z)^2\right] \Delta z}$$

The extra “z” has been included in the numerator as (I * deltaZ). In MATLAB, the calculation of the centroid looks like this:

```
» numerator   = 0;                        %Initialize numerator to zero
» denominator = 0;                        %Initialize denominator to zero
» for i = 1:N                             %Sum both numerator and denominator
   numerator   = numerator + (i*deltaZ) * pi * (R^2 - (i*deltaZ)^2) * deltaZ;
   denominator = denominator + pi * (R^2 - (i*deltaZ)^2) * deltaZ;
end
» z_bar = numerator / denominator
z_bar =
    2.6530
```

We can use the analytical result to check our answer.

```
» z_bar_act = 3/8 *R
z_bar_act =
    2.6250
```

Annotated MATLAB Script Solution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Define the System
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
R  = 7;              %Radius
N  = 70;             %Number of disks
deltaZ = R/N;        %Calculate disk thickness
fprintf('SYSTEM\n')
fprintf('Raidus = %3.1f cm\t\tN = %3.0f\t\tdeltaZ = %3.2f\n\n',R,N,deltaZ)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate the Volume
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Sum (pi * (R^2 - (i*deltaZ)^2) * deltaZ) for i = 1,2,...,N
V_predicted = 0;     %Initialize predicted volume to zero
for i = 1:N
   V_predicted = V_predicted + pi * (R^2 - (i*deltaZ)^2) * deltaZ;
end
fprintf('CALCULATE THE VOLUME\n')
fprintf('Predicted Volume = %3.3f cm^3\n',V_predicted)

V_actual = 2/3 * pi * R^3;
fprintf('Actual Volume = %3.3f cm^3\n\n',V_actual)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate the Centroid
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
numerator   = 0;     %Initialize numerator to zero
denominator = 0;     %Initialize denominator to zero
for i = 1:N          %Sum both numerator and denominator
   numerator   = numerator + (i*deltaZ) * pi * (R^2 - (i*deltaZ)^2) *
deltaZ;
   denominator = denominator + pi * (R^2 - (i*deltaZ)^2) * deltaZ;
end
z_bar = numerator / denominator;
fprintf('CALCULATE THE CENTROID\n')
fprintf('Calculated z_bar = %3.3f cm\n',z_bar)

z_bar_act = 3/8 *R;
fprintf('Actual z_bar = %3.3f cm\n',z_bar_act)
```

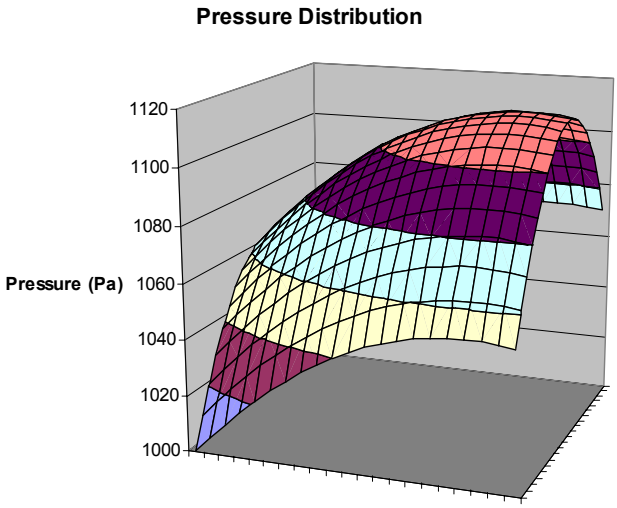
10 Resultant of a Generalized Distributed Loading

Ref: Hibbeler § 9.5, Bedford & Fowler: Statics § 7.4

When a load is continuously distributed across an area it is possible to determine the equivalent resultant force, and the position at which the resultant force acts. This is termed the *resultant of a generalized distributed loading*. This is the generalized case of the simple distributed loading considered earlier (#6).

Example: Load Distribution Expressed as a Function of x and y

The pressure on a surface is distributed as illustrated in the following surface plot:



This plot is described by the following function:

$$p(x, y) = 1000 + 230\,x - 210\,x^2 + 120\,y - 70\,y^2$$

The x and y values range from 0 to 1 meter. The pressure is expressed in Pa.

Determine the magnitude and location of the resultant force.

Solution

The magnitude of the resultant force is obtained by integration.

$$F_R = \int_A p(x, y)\,dA = \int_y \int_x p(x, y)\,dx\,dy$$

To perform this integration we first need to create an m-file for the pressure function, p(x,y), and save it in the MATLAB search path. We then use the MATLAB dblquad() function to carry out the integration on the new function.

```
function p = PressureFunction(x, y)
%Saved as PressureFunction.m in the MATLAB search path.

%Pressure Function (Pa)
p = 1000 + 230 .*x - 210 .*x.^2 + 120 .* y - 70 .* y.^2;
```

```
» F_R = dblquad('PressureFunction', 0, 1, 0, 1)                    %Newtons
F_R =
  1081.7
```

The location at which the resultant force acts is found by calculating the centroid of the volume defined by the distributed loading diagram. We again need to create a new function to calculate the 1st moment of p(x, y). The centroid is then found by dividing the x and y 1st moments by magnitude of the resultant force.

```
function FM = FirstMomentX(x, y)
%Saved as FirstMomentX.m in the MATLAB search path.

%Pressure Function (Pa)
FM = x .* (1000 + 230 .*x - 210 .*x.^2 + 120 .* y - 70 .* y.^2);
```

```
» x_loc = dblquad('FirstMomentX',0,1,0,1) ./ F_R                   %Meters
x_loc =
   0.5015
```

```
function FM = FirstMomentY(x, y)
%Saved as FirstMomentY.m in the MATLAB search path.

%Pressure Function (Pa)
FM = y .* (1000 + 230 .*x - 210 .*x.^2 + 120 .* y - 70 .* y.^2);
```

```
» y_loc = dblquad('FirstMomentY',0,1,0,1) ./ F_R                   %Meters
y_loc =
   0.5039
```

Annotated MATLAB Script Solution

```
function p = PressureFunction(x, y)
%Saved as PressureFunction.m in the MATLAB search path.

%Pressure Function (Pa)
p = 1000 + 230 .*x - 210 .*x.^2 + 120 .* y - 70 .* y.^2;
```

```
function FM = FirstMomentX(x, y)
%Saved as FirstMomentX.m in the MATLAB search path.

%Pressure Function (Pa)
FM = x .* (1000 + 230 .*x - 210 .*x.^2 + 120 .* y - 70 .* y.^2);
```

```
function FM = FirstMomentY(x, y)
%Saved as FirstMomentY.m in the MATLAB search path.

%Pressure Function (Pa)
FM = y .* (1000 + 230 .*x - 210 .*x.^2 + 120 .* y - 70 .* y.^2);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Resultant of a Generalized Distributed Loading                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%The magnitude of the resultant force is calculated.
F_R = dblquad('PressureFunction', 0, 1, 0, 1);%Newtons
fprintf('\nThe magnitude of the resultant force is = %1.0f N\n', F_R)

%The location of the resultant force is calculated.
x_loc = dblquad('FirstMomentX', 0, 1, 0, 1) ./ F_R;%meters
y_loc = dblquad('FirstMomentY', 0, 1, 0, 1) ./ F_R;%meters
fprintf('The location of the resultant force is = (%1.3f, %1.3f)m\n', x_loc, y_loc)
```

11 Calculating Moments of Inertia

Ref: Hibbeler § 10.1-10.2, Bedford & Fowler: Statics § 8.1-8.2

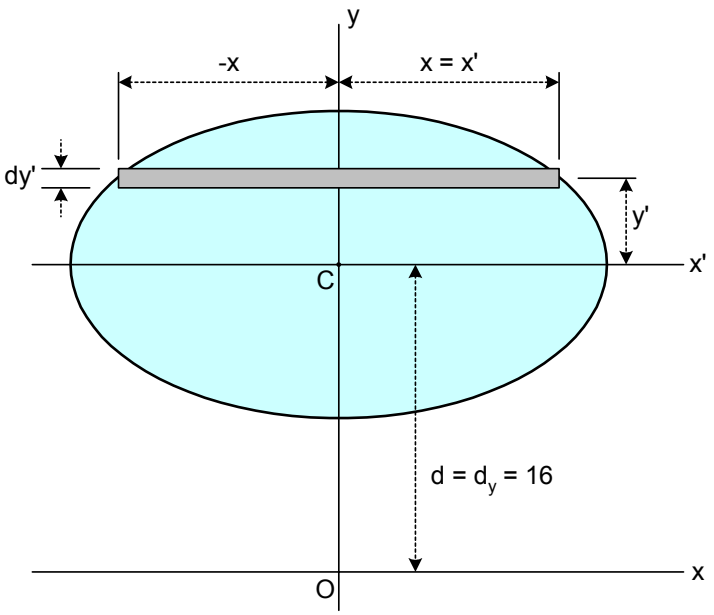
Calculating moments of inertia requires evaluating integrals. This can be accomplished either symbolically, or using numerical approximations. Mathcad's ability to integrate functions to generate numerical results is illustrated here.

Example: Moment of Inertia of an Elliptical Surface

Determine the moment of inertia of the ellipse illustrated below with respect to a) the centroidal x' axis, and b) the x axis.

The equation of the ellipse relative to centroidal axes is

$$\frac{x'^2}{8^2} + \frac{y'^2}{14^2} = 1$$



In this problem, x and y have units of cm.

Solution

The moment of inertia about the centroidal x axis is defined by the equation

$$I_{x'} = \int_A y'^2 \, dA$$

where dA is the area of the differential element indicated in the figure above.

$$dA = 2\,x\,dy'$$

So, the integral for moment of inertia becomes

$$I_{x'} = \int_A y'^2 \; 2\,x \; dy'$$

Furthermore, x (or x') can be related to y' using the equation of the ellipse.

Note: Because of the location of the axes, x = x' in this example.

$$x = x' = \sqrt{8^2\left(1 - \frac{y'^2}{14^2}\right)}$$

The equation for the moment of inertia becomes:

$$I_{x'} = \int_{-8}^{8} y'^2 \; 2\sqrt{8^2\left(1 - \frac{y'^2}{14^2}\right)}\, dy'$$

To perform this integration we need to place the integrand in an m-file function and call MATLAB's quad() function on the m-file.

```
function Ix_integrand = Moment_Of_Inertia_Integrand(y_prime)
%Saved as Moment_Of_Inertia_Integrand.m in the MATLAB
%search path.

x = sqrt(8.^2 .* (1 - y_prime.^2./14^2));
Ix_integrand = y_prime.^2 .* 2 .* x;
```

```
» Ix_prime = quad('Moment_Of_Inertia_Integrand',-8,8)          %cm^4

Ix_prime =

 4890
```

The moment of inertia relative to the original x axis can be found using the *parallel-axis theorem*.

$$I_x = I_{x'} + A\, d_y^{\,2}$$

Where A is the area of the ellipse, and d_y is the displacement of the centroidal y axis from the original y axis.

The required area can be calculated by integration in the same fashion as before.

```
function A = Area_Integrand(y_prime)
%Saved as Area_Integrand.m in the MATLAB search path.

x = sqrt(8.^2 .* (1 - y_prime.^2 ./ 14^2));
A = 2 .* x;
```

```
» A = quad('Area_Integrand',-8,8) %cm^2

A =

 241.2861
```

Then the moment of inertia about x can be determined.

```
» dy = 16;                                                    %cm
```

```
» I_x = Ix_prime + A .* dy.^2                                    %cm^4

I_x =

 66660
```

Annotated MATLAB Script Solution

```
function Ix_integrand = Moment_Of_Inertia_Integrand(y_prime)
%Saved as Moment_Of_Inertia_Integrand.m in the MATLAB search path.

x = sqrt(8.^2 .* (1 - y_prime.^2./14^2));
Ix_integrand = y_prime.^2 .* 2 .* x;
```

```
function A = Area_Integrand(y_prime)
%Saved as Area_Integrand.m in the MATLAB search path.

x = sqrt(8.^2 .* (1 - y_prime.^2 ./ 14^2));
A = 2 .* x;
```

```
%Calculate the moment of inertia relative to the x' axis.
Ix_prime = quad('Moment_Of_Inertia_Integrand',-8,8);%cm^4
fprintf('\nThe moment of inertia relative to the x'' axis is %1.0f cm^4\n',
Ix_prime)

%Calculate the area of the ellipse.
A = quad('Area_Integrand',-8,8);%cm^2
fprintf('The area of the ellipse is %1.2f cm^2\n', A)

%Use the parallel-axis theorm to calculate the moment of inertia relative to
the x axis
dy = 16;%cm
I_x = Ix_prime + A .* dy.^2;%cm^4
fprintf('The moment of inertia relative to the x axis is = %1.0f cm^4\n', I_x)
```

12 Curve-Fitting to Relate s-t, v-t, and a-t Graphs

Ref: Hibbeler § 12.3, Bedford & Fowler: Dynamics § 2.2

The relationships between a particle's position, s, velocity, v, and acceleration, a, over time are all related. When the relationship between a particle's velocity (for example) and time is described by a mathematical equation, that equation can be integrated to obtain the particle's position, and differentiated to determine the particle's acceleration. But what if the velocity vs. time relationship is described by a set of data values or a graph, rather than a mathematical equation? You can still obtain position and acceleration information, either directly from the data values (numerical approximations of integration and differentiation), or by fitting an equation to the data, then using calculus on the resulting equation. The curve-fitting approach will be demonstrated here.

Example: Fitting a Polynomial to Velocity Data

A driver on a straight track accelerates his car from 0 to 75 mph in 14 seconds, and then allows the car to begin to slow. A data recorder connected to the speedometer recorded the data shown below. Since the car is moving along a straight path, the car's speed and velocity are equal.

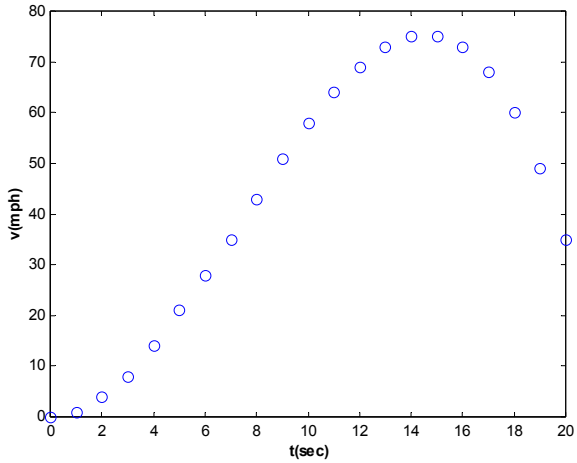
t (sec.)	v (mph)
0	0
1	1
2	4
3	8
4	14
5	21
6	28
7	35
8	43
9	51
10	58
11	64
12	69
13	73
14	75
15	75
16	73
17	68
18	60
19	49
20	35

Fit a curve to the v-t data, and then use the equation of the curve to plot s-t, v-t, and a-t graphs.

Solution – First Attempt

First, let's plot the data to see what we're working with.

```
» t = 0:20;
» v = [0 1 4 8 14 21 28 35 43 51 58 64 69 73 75 75 73 68 60 49 35];
» plot(t,v,'o');
» xlabel('{\bf t(sec)}');
» ylabel('{\bf v(mph)}');
```



The graph shows the increasing velocity for the first 14 seconds then the velocity drops off again. It looks like a smooth curve, and clean (not noisy) data.

A commonly used approach to fitting a curve to a set of data values is polynomial regression. We might try, for example, the following second-order polynomial.[1]

$$v_p = b_1 t^2 + b_2 t + b_3$$

MATLAB's Statistics Toolbox provides a function for fitting linear[2] and nonlinear models such as this polynomial, the nlinfit() function. This function calculates the coefficient values that provide the best fit of the model equation to the data. The regression model is described by either an inline function or an m-file containing the desired function of the independent variable (t, in this case) and coefficients . For our second-order polynomial, the vector could be written as follows:

```
» F = inline('b(1).*t.^2 + b(2).*t  ', 'b', 't')
F =
     Inline function:
     F(b,t) = b(1).*t.^2 + b(2).*t
```

Note: We want the velocity to be zero when t = 0. The way to force this to happen is to leave off the constant term in the F(b,t) function.

Then, nlinfit() uses the t and v values (as vectors), the F(b,t) function shown above, and a vector of initial guesses for each coefficient in F to calculate the b values that produce the "best" fit of the polynomial to the data.

```
» b = nlinfit(t,v,F,[0 0]);
» b
b =
```

[1] A second-order polynomial is not a good choice for this data. A parabola is a second-order equation, and the data shows a lot more curvature than a simple parabola. It will require a higher-order polynomial to adequately fit this data – but the goal of the first attempt is to demonstrate that the "best fit" polynomial may still be a very poor fit to the data.

[2] For curve-fitting, the variables are the model coefficients, the b values. All of the times are known from the data set. This polynomial is linear in the b values, so it is a linear model for curve fitting.

-0.2706

8.2273

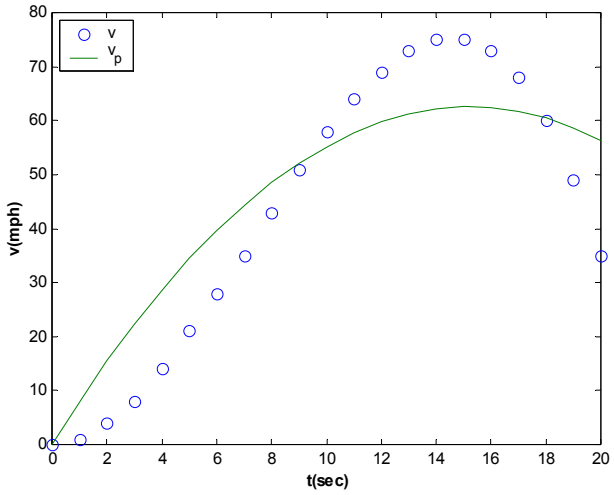
These coefficients tell us that the regression model is

$$v_p = -0.2706\,t^2 + 8.2273\,t + 0$$

The subscript p indicates that the equation is used to calculate predicted values of velocity.

The calculated coefficients provide the *best* possible fit of the model equation to the data, but that does not guarantee that the equation is a *good* fit. You should always calculate predicted values with the regression equation and compare them to the original data to visually determine if the regression equation is actually fitting the data.

```
» v_p = b(1).*t.^2  + b(2).*t;
» plot(t,v,'o',t,v_p);
» xlabel(' t(sec) ');
» ylabel(' v(mph) ');
» legend('v','v_p',2)
```



Clearly, the regression curve (solid line) is not doing a good job of fitting the data.

Solution – Second Attempt

We need a better regression equation. We'll try a third-order polynomial.

$$v_p = b_1 t^3 + b_2 t^2 + b_3 t + b_4$$

The inline function F(b,t) gets another element…

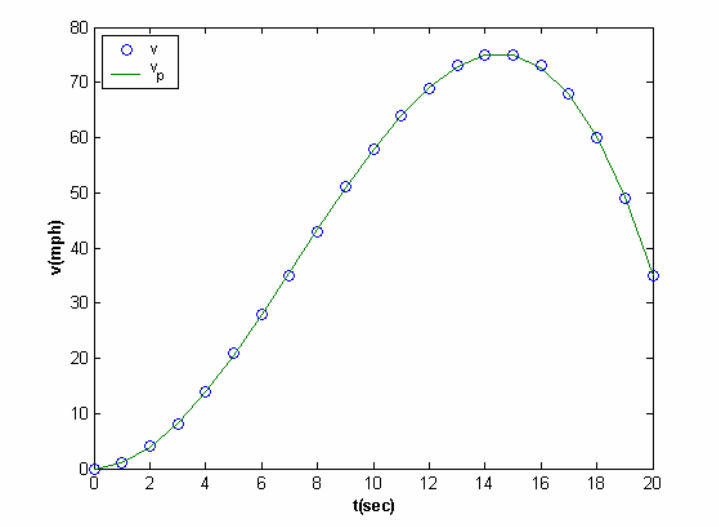
```
» F = inline('b(1).*t.^3 + b(2).*t.^2 + b(3).*t','b','t');
```

…and nlinfit() now finds three b values…

```
» b = nlinfit(t,v,F,[0 0 0]);
» b
b =
  -0.0493
   1.0762
  -0.0403
```

Again, we calculate predicted velocities to check the fit.

```
» v_p = v_p = b(1).*t.^3 + b(2).*t.^2  + b(3).*t;
» plot(t,v,'o',t,v_p);
» xlabel(' t(sec) ');
» ylabel(' v(mph) ');
» legend('v','v_p',2)
```



The new model is doing a good job of fitting the data. The regression equation that fits the data is

$$v_p = 0 - 0.04\,t + 1.076\,t^2 - 0.049\,t^3$$

This equation can be used to generate the s-t and a-t graphs.

MATLAB can integrate the v_p equation to get position, s, and differentiate the equation for acceleration, a, using the polyint() and polydiff() functions respectively. MATLAB represents polynomial's as vectors of coefficients. The first vector element is the coefficient on the highest ordered term and the last vector element is the constant. Therefore, we need to add a zero to the end of our b vector so it is treated as a 3rd order polynomial by MATLAB.

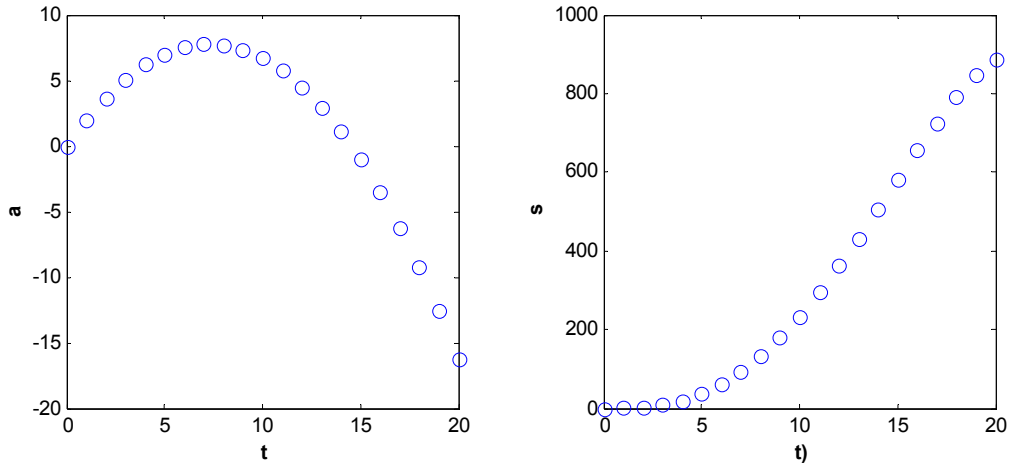
```
» b(4) = 0
b =
  -0.0493
   1.0762
  -0.0403
        0

» b(4) = 0;
» b_a = polyder(b)
b_a =
  -0.1480    2.1524   -0.0403

» b_s = polyint(b)
b_s =
  -0.0123    0.3587   -0.0202         0         0
```

Note: When MATLAB integrates, it does not add the constant of integration. It effectively assumed the car was at s = 0 at t = 0.

```
» a = b_a(1).*t.^2 + b_a(2).*t + b_a(3);
» s = b_s(1).*t.^4 + b_s(2).*t.^3 + b_s(3).*t.^2 + b_s(4)*t;
```

With the equations above, acceleration and position vectors a and s have been calculated by MATLAB. All that remains is to plot the results.

```
» subplot(1,2,1);
» plot(t,a,'o');
» xlabel('t');
» ylabel('a');

» subplot(1,2,2);
» plot(t,s,'o');
» xlabel('t');
» ylabel('s');
```

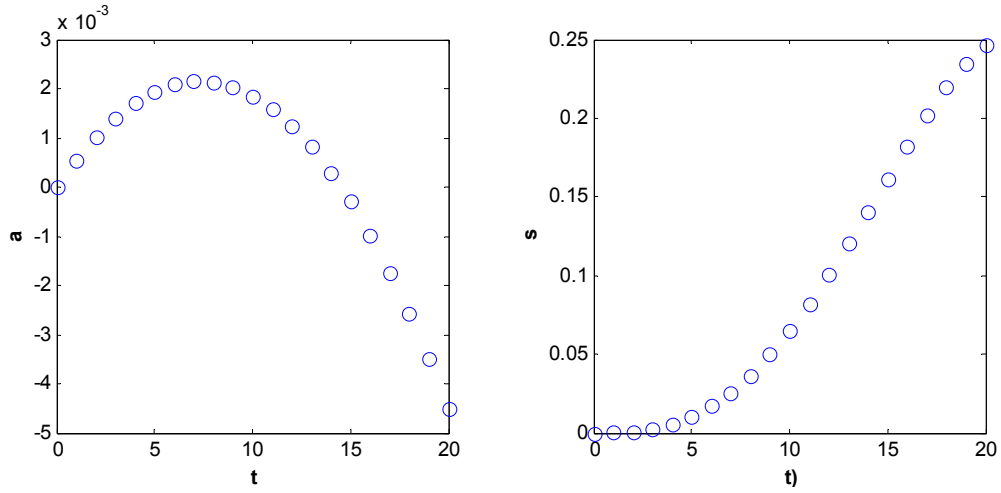


There are some strange units on the values plotted here. The acceleration has units of mi/sec·hr, and the position, s, has units of mi·sec/hr. We can clean this up a bit by building the unit conversions into the calculations for a and s.

```
» a = a ./ 3600;
» s = s ./ 3600;

» subplot(1,2,1);
» plot(t,a,'o');
» xlabel('t');
» ylabel('a');

» subplot(1,2,2);
» plot(t,s,'o');
```

» xlabel('t');

» ylabel('s');



The units are now a [=] mi/sec^2 and s [=] miles.

Annotated MATLAB Script Solution

```
%Define the time and velocity vectors, and plot the v-t graph
t = 0:20;                                          %t = 0,1,2,...,20
v = [0 1 4 8 14 21 28 35 43 51 58 64 69 73 75 75 73 68 60 49 35];
plot(t,v,'o');
xlabel('{\bf t(sec)}');
ylabel('{\bf v(mph)}');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         First Solution Attempt - Second Order Polynomial                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Define the polynomial using the inline function F(b,t).
F = inline('b(1).*t.^2 + b(2).*t','b','t');
%Find the best-fit coefficients using F(b,t).
b = nlinfit(t,v,F,[0 0]);
%Calculate predicted velocity values (at each t value) to see if the
regression equation actually fits the data.
v_p = b(1).*t.^2  + b(2).*t;
subplot(1,2,1);
plot(t,v,'o',t,v_p);
xlabel('{\bf t(sec)}');
ylabel('{\bf v(mph)}');
legend('v','v_p',2)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         Second Solution Attempt - Third Order Polynomial                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Define the polynomial using the inline function F(b,t).
F = inline('b(1).*t.^3 + b(2).*t.^2 + b(3).*t','b','t');
%Find the best-fit coefficients using F(b,t).
b = nlinfit(t,v,F,[0 0 0]);
```

```
%Calculate predicted velocity values (at each t value) to see if the
regression equation actually fits the data.
v_p = b(1).*t.^3 + b(2).*t.^2  + b(3).*t;
subplot(1,2,2);
plot(t,v,'o',t,v_p);
xlabel('{\bf t(sec)}');
ylabel('{\bf v(mph)}');
legend('v','v_p',2);

%Convert b to a MATLAB polynomial vector
b(4) = 0;

%Take the derivative of b to find the equation for a.
b_a = polyder(b);
%Calculate a at each value of t
a = b_a(1).*t.^2  + b_a(2).*t + b_a(3);
%Simplify the units by building in the conversion from hours to seconds.
a = a ./ 3600;

%Integrate b to find the equation for s.
b_s = polyint(b)
%Calculate s at each value of t
s = b_s(1).*t.^4 + b_s(2).*t.^3  + b_s(3).*t.^2 + b_s(4)*t;
%Simplify the units by building in the conversion from hours to seconds.
s = s ./ 3600;

%Plot the results in a new figure.
figure;
subplot(1,2,1);
plot(t,a,'o');
xlabel('{\bf t}');
ylabel('{\bf a}');

subplot(1,2,2);
plot(t,s,'o');
xlabel('{\bf t)}');
ylabel('{\bf s}');
```

13 Curvilinear Motion: Rectangular Components

Ref: Hibbeler § 12.5, Bedford & Fowler: Dynamics § 2.3

A fixed x, y, z frame of reference is commonly used to describe the path of a particle in two situations:

1. The path "fits" the Cartesian coordinate system well (tends to follow straight lines, or simple paths.)
2. The path does not "fit" any commonly used coordinate system well.

The example used here falls into the latter category.

Example: Finding the Velocity and Acceleration of a Particle

The complex path of a particle for times between 0 and 10 seconds has been fit using multivariable regression to the following functions:

$$x(t) = 2\cos(t)$$
$$y(t) = 1.3 + 2.7\,t^2 - 0.031\,t^3$$
$$z(t) = 2.4 + 0.14\,t^3$$

The position of the particle at any point in time for which the functions are valid ($0 \le t \le 10$ sec.) can be determined as

$$\mathbf{r} = x\,\mathbf{i} + y\,\mathbf{j} + z\,\mathbf{k}$$

Given these functions for x, y, and z determine the position, and the magnitudes of the velocity and acceleration of the particle at t = 7 sec.

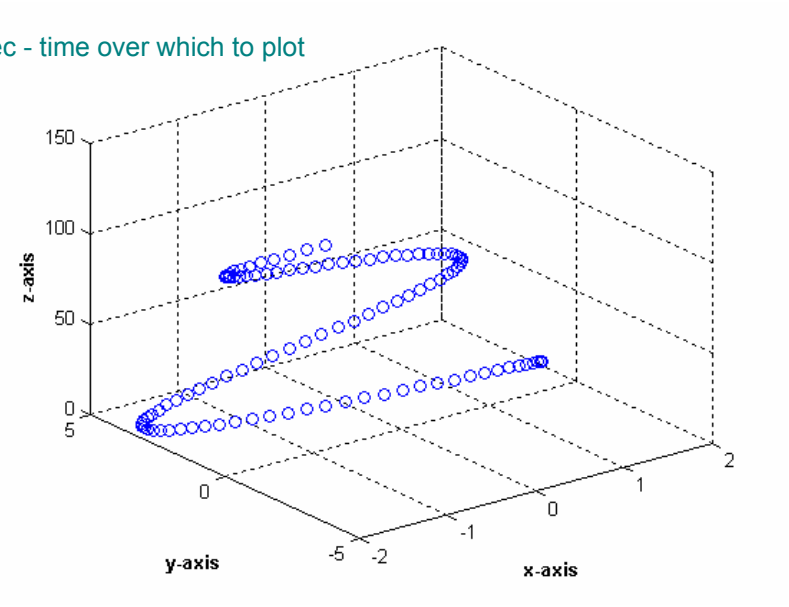
Solution

The first thing we might want to do is graph this particle path, just to see what it looks like. MATLAB can help here. First, declare the three functions of time that describe the particle's path:

```
» x = inline('2 .* cos(t)', 't')
x =
     Inline function:
     x(t) = 2 .* cos(t)

» y = inline('1.3 + 0.27.*t.^2 - 0.031.*t.^3', 't')
y =
     Inline function:
     y(t) = 1.3 + 0.27.*t.^2 - 0.031.*t.^3

» z = inline('2.4 + 0.14.*t.^3', 't')
z =
     Inline function:
     z(t) = 2.4 + 0.14.*t.^3
```

Then ask MATLAB to plot the particle path by placing the three function names into the scatter3() function.

```
» t = 0 : 0.1: 10;          %sec - time over which to plot
» scatter3(x(t),y(t),z(t),'o')
» xlabel('x-axis')
» Ylabel('y-axis')
» zlabel('z-axis')
```



Solving for Position

Declare r(t) as a three-component matrix composed of the three functions. The position is described by function r(t), so finding the position at t = 7 sec. Simply requires evaluating r(t) at this time.

```
» r = inline('[ [2 .* cos(t)]   [1.3 + 0.27. *t.^2 - 0.031 .*t.^3]   [2.4 + 0.14. * t.^3] ]', 't')
r =
     Inline function:
     r(t) = [[2.*cos(t)] [1.3+0.27.*t.^2-0.031.*t.^3] [2.4+0.14.*t.^3]]
» r(7)
ans =
   1.5078    3.8970   50.4200
```

So, at t = 7 seconds, the particle is located at x = 1.5078, y = 3.8970, and z = 50.4200.

Solving for the Magnitude of the Velocity

For velocity we need to take the derivative of the x, y, and z functions. Here, the derivatives are obtained using MATLAB's symbolic math toolbox and cutting-and-pasting into three new inline functions: $v_x(t)$, $v_y(t)$, and $v_z(t)$.

Note: Capital letters are used to represent symbolic variables while lower case letters are assigned a value or to a function.

```
» syms T                    %Define a symbolic variable T
» X = 2 .* cos(T);          %Define X
» diff(X)                   %Differentiate on X
ans =
 -2*sin(T)
```

```
» v_x = inline('-2 .* sin(t)','t');                    %Assign v_x(t)

» Y = 1.3 + 0.27.*T.^2 - 0.031.*T.^3;                  %Define Y
» diff(Y)                                              %Differentiate onY
ans =
 27/50*T-93/1000*T^2
» v_y = inline('27/50 - 93/1000 .*t.^2', 't');         %Assign v_y(t)

» Z = 2.4 + 0.14.*T.^3;                                %Define Z
» diff(Z)                                              %Differentiate onZ
ans =
 21/50*T^2
» v_z = inline('21/50 .* t.^2', 't');                  %Assign v_z(t)
```

The magnitude of the velocity at t = 7 seconds can now be determined.

```
» v = sqrt(v_x(7).^2 + v_y(7).^2 + v_z(7).^2)
v =
   21.0095
```

Solving for the Magnitude of the Acceleration

To find the acceleration, we need to differentiate the v_x, v_y, and v_z functions with respect to time. The derivatives obtained using MATLAB's symbolic math toolbox and cutting-and-pasting into three new inline functions: $a_x(t)$, $a_y(t)$, and $a_z(t)$.

```
» diff(-2*sin(T), T)                                   %Differentiate v_x
ans =
 -2*cos(T)
» a_x = inline('-2.*cos(t)','t')                       %Assign a_x(t)
 a_x =
     Inline function:
     a_x(t) = -2.*cos(t)

» diff(27/50 .* T - 93/1000 .* T.^2, T)                %Differentiate v_y
ans =
 27/50-93/500*T
» a_y = inline('0.54 - 0.186.*t','t')                  %Assign a_y(t)
a_y =
     Inline function:
     a_y(t) = 0.54 - 0.186.*t
```

```
» diff(21/50 .* T.^2, T)                    %Differentiate v_z
ans =
 21/25*T
» a_z = inline('0.84.*t','t')               %Assign a_z(t)
a_z =
     Inline function:
     a_z(t) = 0.84.*t
```

The magnitude of the acceleration at t = 7 seconds is

```
» a = sqrt( a_x(7).^2 + a_y(7).^2 + a_z(7).^2 )
a =
 6.1179
```

Annotated MATLAB Script Solution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         Curvilinear Motion:  Rectangular Components                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Declare the time functions for the particle's path.
x = inline('2 .* cos(t)', 't');
y = inline('1.3 + 0.27.*t.^2 - 0.031.*t.^3', 't');
z = inline('2.4 + 0.14.*t.^3', 't');

%Create a plot to see the particle path
t = 0 : 0.1: 10;%sec - time over which to plot
scatter3(x(t),y(t),z(t),'o')
xlabel('{\bf x-axis}')
Ylabel('{\bf y-axis}')
zlabel('{\bf z-axis}')

%Declare the r(t) function.
r = inline('[[2.*cos(t)] [1.3+0.27.*t.^2-0.031.*t.^3]
[2.4+0.14.*t.^3]]','t');

%Evaluate r(t) at t = 7 to find the position.
position = r(7);
fprintf('\nr(7) = [ %1.3f, %1.3f, %1.3f ]\n', r(7))

%Take the time derivatives of x, y, and z functions.
syms T                                      %Define a symbolic variable T
X = 2 .* cos(T);                            %Define X
diff(X)                                     %Differentiate on X
v_x = inline('-2 .* sin(t)','t');           %Assign v_x(t)

Y = 1.3 + 0.27.*T.^2 - 0.031.*T.^3;         %Define Y
diff(Y)                                     %Differentiate on Y
```

```
v_y = inline('27/50 - 93/1000 .*t.^2', 't'); %Assign v_y(t)

Z = 2.4 + 0.14.*T.^3;                        %Define Z
diff(Z)                                      %Differentiate on Z
v_z = inline('21/50 .* t.^2', 't');          %Assign v_z(t)

%Calculate the magnitude of the velocity at 7 seconds.
v = sqrt(v_x(7).^2 + v_y(7).^2 + v_z(7).^2);
fprintf('Magnitude of the velocity at 7 seconds = %1.3f \n', v)

%Take the time derivatives of the velocity functions.
diff(-2*sin(T), T)                           %Differentiate on v_x
a_x = inline('-2.*cos(t)','t');              %Assign a_x(t)

diff(27/50 .* T - 93/1000 .* T.^2, T);       %Differentiate on v_y
a_y = inline('0.54 - 0.186.*t','t');         %Assign a_y(t)

diff(21/50 .* T.^2, T)                       %Differentiate on v_z
a_z = inline('0.84.*t','t');                 %Assign a_z(t)

%Calculate the magnitude of the acceleration at 7 seconds.
a = sqrt(a_x(7).^2 + a_y(7).^2 + a_z(7).^2);
fprintf('Magnitude of the acceleration at 7 seconds = %1.3f \n\n', a)
```

14 Curvilinear Motion, Motion of a Projectile

Ref: Hibbeler § 12.6, Bedford & Fowler: Dynamics § 2.3

Rectilinear motion refers to motion in a straight line. When a particle follows a non-straight path, it's motion is termed *curvilinear*. Projectile motion is typically curvilinear, although a projectile fired straight up (in the absence of a crosswind), or moving along a straight track would be rectilinear motion.

A projectile's motion can be broken down into three phases: an acceleration phase where the actuator (gun, catapult, golf club, etc.) gets the projectile moving. The second phase of motion is after the projectile leaves the actuator, when the only acceleration acting on it is the acceleration due to gravity.

Note: A common assumption that simplifies the problem considerably, but is not altogether accurate, is that the frictional drag between the projectile and the fluid through which it moves is negligible. This assumption is more reasonable for small, smooth, slow-moving particles through low-viscosity fluid than for large, irregularly shaped particles moving at high speeds through highly viscous fluids.

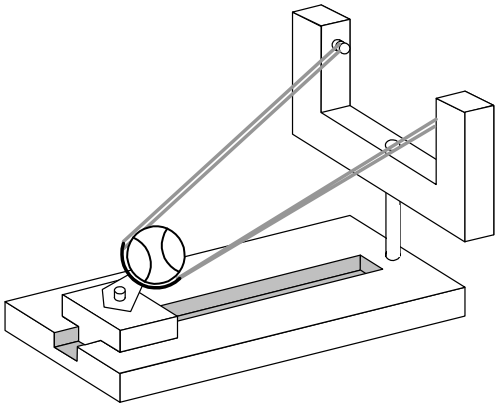
The third phase of a projectile's motion is after impact. The particle may roll, continue moving through a very different medium (water or earth), or break up. Here we will consider only the second phase of the projectile's motion: the projectile has already been accelerated by an actuator, and is flying through the air.

Example: Slingshot Contest

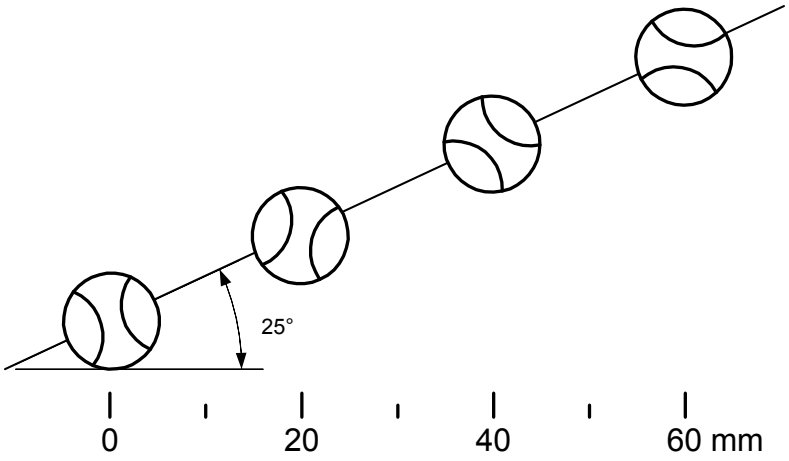
Projectile throwing contests are pretty common at engineering schools. There are several ways to run the contest: highest, farthest, most accurate, etc; and several ways to propel the projectile: slingshot, catapult, etc. This example focuses on a tennis ball slingshot competition.

Tennis Ball Data (approximate)	
Diameter	2.5 inches (6.4 cm)
Weight	2 oz (57 g)

A team of mechanical engineering students has built a slingshot that allows precise control of the pre-release tension and angle. They also connected the release mechanism to a digital camera to take a series of photos in 1millisecond intervals just as the tennis ball is released. The photos are used to calculate the velocity (speed and angle) at which the tennis ball leaves the sling.



A test run with the camera operational gave the following set of photos (superimposed).



Part 1.

Determine:

a. The initial velocity of the tennis ball as it leaves the sling.
b. The predicted time of flight for the ball.
c. The predicted horizontal travel distance for the ball.

Part 2.

The team that gets most balls into a basket set 30 meters from the launch site wins. If they can keep the initial speed constant (at the test run value of 22.1 m/s), what angle should they use to shoot the tennis balls into the basket?

Part 1. Solution

The initial velocity is calculated from the 60 mm horizontal travel distance observed in the photos using the camera's 1 millisecond interval between snapshots.

```
» x_test  = 60 / 1000;              %Horizontal travel distance for all four frames (m)
» theta   = 25 * pi/180;            %Angle of trajectory (radians)
» d_test  = x_test / cos(theta)     %Distance ball traveled in direction of motion  (m)
d_test =
   0.0662
» dt_pic  = 0.001;                  %Interval between snapshots (s)
» dt_test = 3 * dt_pic;             %3 time intervals between the four photos
» v_int   = d_test / dt_test        %Initial velocity of the tennis ball
v_int =
  22.0676
```

The horizontal and vertical components of the initial velocity will be useful for later calculations.

```
» v_yint  = v_int * sin(theta)    %y component of initial velocity
```

```
v_yint =
    9.3262
» v_xint  = v_int * cos(theta)    %x component of initial velocity
v_xint =
    20
» v_x  = v_xint                  %x component of velocity is constant (if air resistance is ignored)
v_x =
    20
```

The predicted time of flight can be calculated using

$$y = y_0 + v_{y\_init}\ t_{flight} + \tfrac{1}{2} a_c\ t_{flight}^{\ 2}$$

where a_c is the constant acceleration. In this problem, the acceleration is due to gravity and acts in the –y direction, so a_c = -g.

If we assume that the flight is over a horizontal surface, then y at the end of the flight is zero. Another common assumption is to assume that y_0 is also zero. This is a reasonable assumption if the vertical position of the tennis ball as it leaves the sling is small compared to the maximum height reached during the flight.

With these assumptions, the equation can be solved for the time of flight

$$t_{flight} = \frac{-2\ v_{y\_init}}{a_c}$$

Using MATLAB, the time of flight is predicted to be 1.9 seconds.

```
» a_c     = -9.8;                %The constant acceleration in this problem is due to gravity
» t_flight= -2 * v_yint / a_c
t_flight =
    1.9033
```

We can calculate the maximum height to see if the assumption that y_0 is negligible is reasonable. The maximum height occurs at ½ the flight time (if $y_0 = y_{final}$, and air resistance is negligible).

```
» y_o     = 0;
» y_final = 0;
» y_max   = y_o + v_yint *  t_flight / 2 + a_c / 2 * ( t_flight/2 )^2
y_max =
    4.4376
```

Looking at the drawing of the slingshot, the ball is released at a height about two or three times the ball diameter, around 15 to 20 cm. 20 cm is nearly 5% of y_{max}. That is probably negligible, but we can use MATLAB's root finder to find t_{flight} including y_0 = 20 cm.

```
» y_o     = 20 / 100;                                  %Initial height (m)
» y_final = 0;                                         %Final height (m)
» coeffs  = [( a_c/2 ) ( v_yint ) ( y_o - y_final )]; %Polynomial
» t_flight= roots(coeffs)                              %Roots
t_flight =
    1.9245
   -0.0212
» t_flight= t_flight(1)                                %Choose positive root
t_flight =
    1.9245
```

Note: See the annotated MATLAB Script Solution for a more complete explanation of MATLAB's ***roots*** *function and how MATLAB represents polynomials.*

Accounting for the initial height increased the predicted flight time from 1.902 to 1.925 seconds (about 1% difference).

Finally, we calculate the predicted horizontal travel distance.

```
» x_o = 0;                                             %Initial horizontal position
» x   = x_o + v_x * t_flight
x =
   38.4901
```

Annotated MATLAB Script Solution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Projectile Motion: Slingshot Contest
%Calculate Initial Velocity from Test Run
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Horizontal travel distance for all four frames (m)
x_test  = 60 / 1000;
%Angle of trajectory (radians)
theta   = 25 * pi/180;
%distance ball traveled in direction of motion (m)
d_test  = x_test / cos(theta);
fprintf('Distance ball traveled in direction of motion = %1.4fm\n',d_test)

%Interval between snapshots (s)
dt_pic  = 0.001;
%3 time intervals between the four photos
dt_test = 3 * dt_pic;
%Initial velocity of the tennis ball
v_int   = d_test / dt_test;
fprintf('Initial velocity = %3.1f m/s\n',v_int)

%y component of initial velocity
v_yint  = v_int * sin(theta);
fprintf('\ty component = %3.1f m/s\n',v_yint)
%x component of initial velocity
v_xint  = v_int * cos(theta);
fprintf('\tx component = %3.1f m/s\n\n',v_xint)

%x component of velocity is constant (if air resistance is ignored)
v_x     = v_xint;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate Time of Flight
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The constant acceleration in this problem is due to gravity
a_c     = -9.8;
t_flight= -2 * v_yint / a_c;
fprintf('Flight time (Ignoring y_o) = %3.3f s\n',t_flight)

y_o     = 0;                            %Initial height
y_final = 0;                            %Final height
y_max   = y_o + v_yint *  t_flight/2 + a_c/2 * ( t_flight/2 )^2;
fprintf('y_max = %3.1f m\n\n',y_max)
```

continues…

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Alternative Solution Method - Include y_o and use MATLAB's root finder
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%The predicted time of flight can be calculated using
%      y_max = y_o + v_yint * t_flight + a_c/2 * t_flight^2
%
%This equation is a second order polynomial in t_flight and can be
%rewritten in the form
%                  0  = a2 * x^2 + a1 * x + a0
%     with
%                           a2 = a_c/2
%                           a1 = v_yint
%                           a0 = y_o - y_final
%
%Polynomials are represented in MATLAB by an array of the
%coefficients in descending order.  Therefore, this polynomial
%would be represented by the following
%               coefficients  = [a2 a1 a0]
%                           or
%        coefficients  = [( a_c/2 ) ( v_yint ) ( y_o - y_final )];

y_o     = 20 / 100;                                      %Initial height (m)
y_final = 0;                                             %Final height (m)
coeffs  = [( a_c/2 ) ( v_yint ) ( y_o - y_final )];      %Polynomial
t_flight= roots(coeffs);                                 %Roots
t_flight= t_flight(1);                                %Choose the positive root
fprintf('Alternative Solution Method - Include y_o and use MATLAB''s root
finder\n')
fprintf('Flight time = %3.3f s\n',t_flight)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate Horizontal Motion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x_o = 0;
x   = x_o + v_x * t_flight;
fprintf('Horizontal Motion = %3.3f m\n',x)
```

Part 2.

With the MATLAB Script, we can simply try some different angles until we calculate a predicted horizontal travel distance of 30 meters.

First, try 20 degrees.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate Inital Velocity Components
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Initial velocity of the tennis ball
v_int   = 22.1;
%Angle of trajectory (radians)
theta   = 20 * pi/180;
fprintf('Initial velocity = %3.1f m/s\t\t',v_int)
fprintf('Angle of trajectory = %3.2f deg\n',theta * 180/pi)

%y component of initial velocity
v_yint  = v_int * sin(theta);
fprintf('\ty component = %3.1f m/s\n',v_yint)
%x component of initial velocity
v_xint  = v_int * cos(theta);
fprintf('\tx component = %3.1f m/s\n\n',v_xint)

%x component of velocity is constant (if air resistance is ignored)
v_x     = v_xint;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate Time of Flight
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The constant acceleration in this problem is due to gravity
a_c     = -9.8;
t_flight= -2 * v_yint / a_c;
fprintf('Flight time (Ignoring y_o) = %3.3f s\n',t_flight)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Calculate Horizontal Motion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x_o = 0;
x   = x_o + v_x * t_flight;
fprintf('Horizontal Motion = %3.3f m\n',x)
```

MATLAB Output

```
Initial velocity = 22.1 m/s                Angle of trajectory = 20.00 deg
        y component = 7.6 m/s
        x component = 20.8 m/s


Flight time (Ignoring y_o) = 1.543 s
Horizontal Motion = 32.035 m
```

With a little trial and error, the predicted angle should be 18.5 degrees.

15 Curvilinear Motion: Normal and Tangential Components

Ref: Hibbeler § 12.7, Bedford & Fowler: Dynamics § 2.3

When the path of a particle is known, an n-t coordinate system with an origin at the location of the particle (at an instant in time) can be helpful in describing the motion of the particle. Hibbeler gives a concise procedure for analysis in section 12.7, which we will apply to the following example.

Example: A skateboarder in a quarter pipe.

A skateboarder has dropped into a 3 meter (radius) quarter pipe. When she's dropped a distance of 1 meter her speed is 3.2 m/s and is increasing by 7.8 m/s^2.

Determine the normal and tangential components of the skateboarder's acceleration.

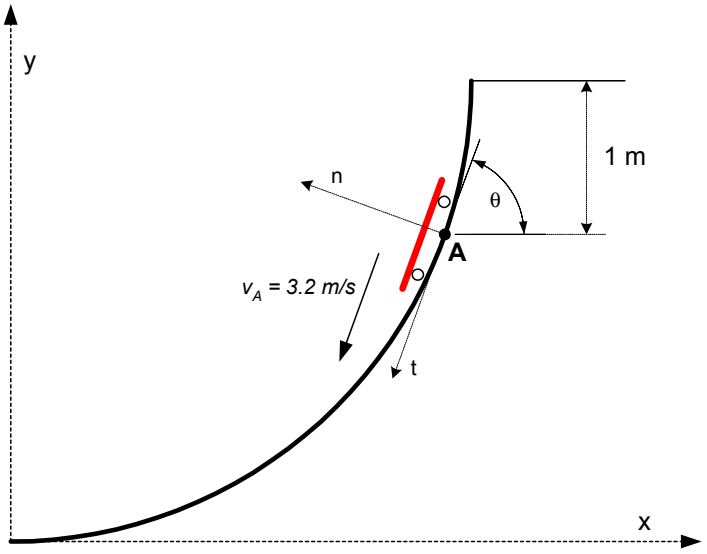
Solution

The equation of the skateboarder's path is that of a quarter circle:

$$y = R - \sqrt{R^2 - x^2}$$

where R is the radius, 3 m.

First, we need to determine the actual location of the skateboarder when she has dropped 1 m.



She is obviously R – 1 = 2 m off the ground, and her x-position can be determined from the path equation using either MATLAB's iterative root finder (fzero), or symbolic math capability. Here, we first create a function (an m-file) whose root we wish to find and then call the fzero iterative solver on the newly created function.

```
%Define the function on which to use fzero.
function F = FindMinimum(x)

R=3;  %3 meters radius
Y=2;  %2 meters off the ground

%The equation describing the scateboarders path is that of
%  and quarter circle => x^2 + y^2  = R^2
%  or similarly  => y = R - sqrt(x^2 + y^2)
%
%  Then setting the equation equal to some arbitrary variable
%  and solving for the root
F = R - sqrt(R^2 - x^2) - Y;
```

```
>> X = fzero('FindMinimum',2)    %Guess: 2

X =

   2.8284
```

So, at the instant of interest, the skateboarder is located at X = 2.828 m, Y = 2 m.

Next, the equation of the slope of the path at this point can be determined using MATLAB's symbolic derivative operator as follows:

```
>> syms  x   y   r                    %Define the x, y , and R as symbolic variables
                                       %Find symbolic result to the first derivative
>> y = r - sqrt( r^2 - x^2 );          %Define symbolic equation
>> dydx = diff(y,x)                    %Find first derivative
 dydx =
 1/(r^2-x^2)^(1/2)*x
```

Note: lowercase variables are the symbolic variables while uppercase variables are assigned a numeric value.

By assigning the derivative to an inline function and X and R to their respective values, the numeric value of the derivative can be calculated.

```
>> R = 3;
>> X = 2.8284;
>> DYDX = inline('1/(r^2-x^2)^(1/2)*x','x','r')
DYDX =
     Inline function:
     DYDX(x,r) = 1/(r^2-x^2)^(1/2)*x
>> DYDX(X,R)
ans =
   2.8282
```

The angle in degrees at point **A** can then be determined as:

```
>> atan(2.8282) * 180/pi
ans =
  70.5273
```

The skateboarder's acceleration can be written in terms of normal and tangential velocities, as

$$\mathbf{a} = \dot{v}\,\mathbf{u}_t + \frac{v^2}{\rho}\mathbf{u}_n$$

$$= 7.8\,\frac{m}{s^2}\,\mathbf{u}_t + \frac{\left(3.2\,\frac{m}{s}\right)^2}{\rho}\mathbf{u}_n$$

The radius of curvature, ρ, of the path at the point of interest (X=2.828 m, Y = 2 m) can be calculated as:

$$\rho = \frac{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{\frac{3}{2}}}{\left|\frac{d^2y}{dx^2}\right|}$$

In MATLAB, this equation is evaluated as follows:

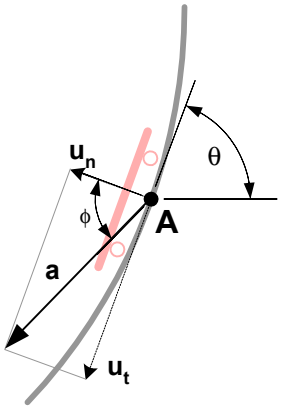
```
>> y = r - sqrt( r^2 - x^2 );                          %Define Symbolic Equation
>> dydx   = diff(y,x)                                  %First Derivative
dydx =
1/(r^2-x^2)^(1/2)*x
>> DYDX = inline('1/(R^2-x^2)^(1/2)*x','x','R');       %Create inline function DYDX
>> dy2dx2 = diff(diff(y,x),x)                          %Second Derivative
dy2dx2 =
1/(r^2-x^2)^(3/2)*x^2+1/(r^2-x^2)^(1/2)
                                                       %Create inline function DY2DX2
>> DY2DX2 = inline('1/(r^2-x^2)^(3/2)*x^2+1/(r^2-x^2)^(1/2)','x','r');
>> RHO = (1 + DYDX(X,R)^2)^(3/2) / abs(DY2DX2(X,R))
RHO =
   3.0000
```

The calculated radius of curvature is 3 m – which should come as no surprise since the path is a circle of radius 3 m.

The equation for the acceleration of the skateboarder in terms of tangential and normal components is now:

$$\mathbf{a} = \dot{v}\,\mathbf{u}_t + \frac{v^2}{\rho}\mathbf{u}_n$$

$$= 7.8\,\frac{m}{s^2}\,\mathbf{u}_t + \frac{\left(3.2\,\frac{m}{s}\right)^2}{3\ m}\mathbf{u}_n$$



The magnitude of the acceleration is found with the following calculation:

```
>> a = sqrt(7.8^2 + ( 3.2^2 / 3 )^2)    %meters per second squared
a =
   8.5142
```

While the angle (in degrees), ϕ, is found using the atan() function and multiplying by the conversion factor $180^{o}/\pi$.

```
>> PHI = atan(v_dot / ( v^2/rho )) *180/pi
PHI =
  66.3655
```

The angle of the acceleration in degrees from the positive x axis is:

```
>> ANGLE = THETA + 90 + PHI
ANGLE =
  226.8941
```

Annotated MATLAB Script Solution

```
%Define the function on which to use fzero.
function F = FindMinimum(x)

R=3;  %3 meters radius
Y=2;  %2 meters off the ground

%The equation describing the scateboarders path is that of
%  and quarter circle => x^2 + y^2  = R^2
%  or similarly  => y = R - sqrt(x^2 + y^2)
%
%  Then setting the equation equal to some arbitrary variable
%  and solving for the root
F = R - sqrt(R^2 - x^2) - Y;
```

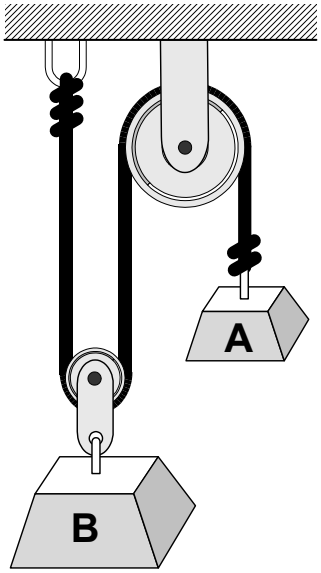
```
%Call fzero on the FindMinimum function with an initial guess of 2
X = fzero('FindMinimum',2);                  %Guess: 2
fprintf('X = %1.4f m .\n', X)

%Define all other variables using uppercase letters to indicate the
variable has been assigned a value.
V_dot = 7.8;
V = 3.2;
R = 3;
Y = 2;

%Define the x, y , and r as symbolic variables.
%Note:  Lowercase variables are all symbolic variables while
%       uppercase variables are assigned a numeric value.
syms x y r;

%Find symbolic result to first and second derivatives
y = r - sqrt( r^2 - x^2 );                   %Define Symbolic Equation
dydx   = diff(y,x)                           %First Derivative
```

```
DYDX = inline('1/(R^2-x^2)^(1/2)*x','x','R');
dy2dx2 = diff(diff(y,x),x)                    %Second Derivative
DY2DX2 = inline('1/(r^2-x^2)^(3/2)*x^2+1/(r^2-x^2)^(1/2)','x','r');

%Calculate the angle at point A.
THETA = atan(X) *180/pi;
fprintf('The angle at point a is %1.4f degrees.\n', THETA)

%Calculate the radius of curvature at X = 2.828 m and y = 2 m.
RHO = (1 + DYDX(X,R)^2)^(3/2) / abs(DY2DX2(X,R));
fprintf('The radius of curvature at X = 2.828 m and y = 2 m is %1.4f
m.\n', RHO)

%Calculate the magnitude of acceleration.
A = sqrt(V_dot^2 + (V^2/RHO)^2);
fprintf('The magnitude of acceleration is %1.4f m/s^2.\n', A)

%Calculate the algle of acceleration relative to the t,n coordinate
system.
PHI = atan(V_dot / ( V^2/RHO ))  *180/pi;

%Calculate the angle of acceleration relative to the positive x-axis.
ANGLE = THETA + 90 + PHI;
fprintf('The angle of acceleration relative to the positive x-axis is
%1.4f degrees.\n', ANGLE)
```

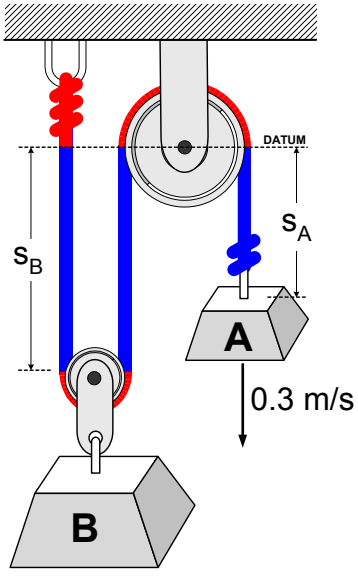
16 Dependent Motion of Two Particles

Ref: Hibbeler § 12.9, Bedford & Fowler: Dynamics, chapter 2 (no directly analogous section)

Sometimes the motion of one particle depends on the motion of another – this is called *dependent motion*. One situation that creates dependent motion is when two particles are connected by a cord around a pulley.



If weight A is pulled downward, weight B will be raised, but the total length of the cord, L_{total}, (assumed inextensible) is constant. The total length of the cord can be described in terms of cord lengths that change, s_A and s_B, and unchanging lengths, combined and called L_{const} (shown in red in the next figure.)



$$2\,s_B + s_A + L_{const} = L_{total}$$

Differentiating this equation with respect to time yields the time rates of change of position, or velocities of each weight.

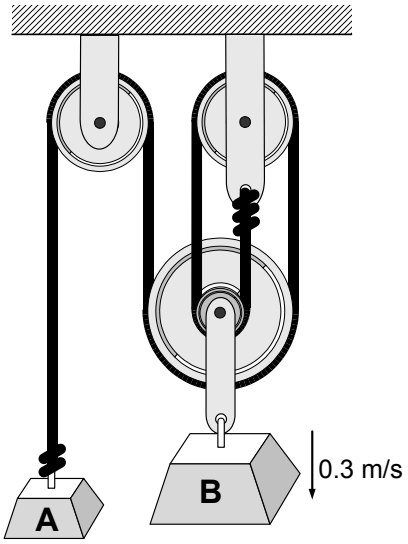
$$2\frac{ds_B}{dt}+\frac{ds_A}{dt}+0=0$$

$$2v_B=-v_A$$

So, if weight A moves down at 0.3 m/s, weight B will move up at 0.15 m/s.

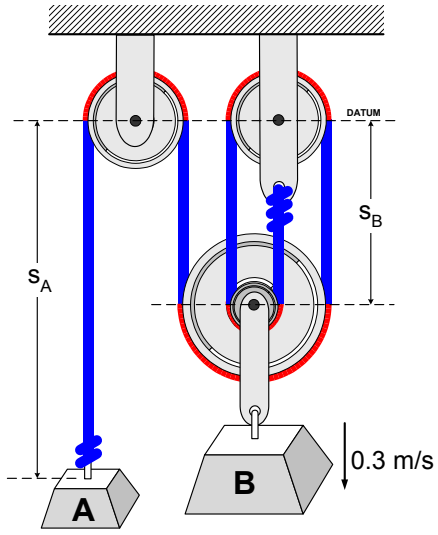
Example: Four Pulley System

At what rate, and in which direction must weight A move if weight B is to fall at a rate of 0.3 m/s?



Solution

There is a single cord between weights A and B, so the motion of one of the weights is dependent upon one the other. While the physical system is somewhat more complex than the original, 2-pulley example, the solution is not very different. The total length of the cord is made up of 9 segments, as shown in the figure below. There are 4 equivalent-length sections that lengthen as weight B is lowered. The length of each of these sections is labeled S_B.



The length of the cord segment connected to weight A is labeled S_A. There are four cord segments around the four pulleys that do not change as the weights move. These segments are shown in red in the figure, and their lengths are combined and called L_{const}. The total cord length, then, is

$$4\,s_B + s_A + L_{const} = L_{total}$$

Differentiating this equation with respect to time yields the time rates of change of position, or velocities of each weight.

$$4\,\frac{ds_B}{dt} + \frac{ds_A}{dt} + 0 = 0$$

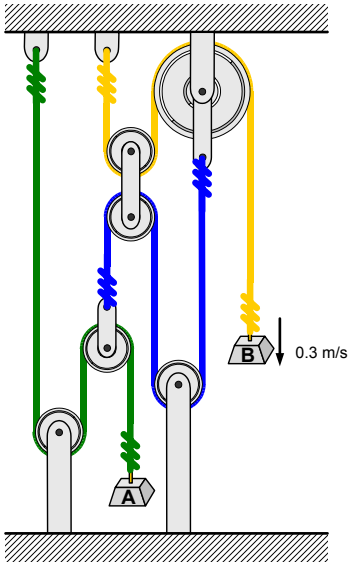
$$4\,v_B = -v_A$$

So, if weight B moves down at 0.3 m/s, weight A will move up at 1.2 m/s.

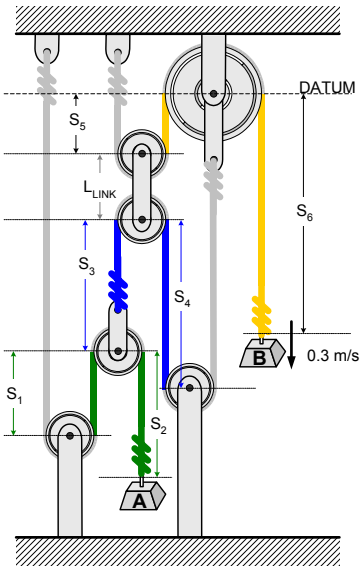
So far, these problems require only simple calculus, and there is little need for MATLAB's calculational abilities. However, when there are several cords involved, MATLAB's matrix math capabilities can come into play.

Example: Three Cord System

In the example shown below, three cords are connected around a complicated pulley system. The length of each of the cords is constant, so the sum of the segment lengths can be written for each cord.



To simplify things a bit, we first determine which segments are of constant length, shown in gray in the following figure. Then the segments that are changing length are labeled (S_1 through S_6, as shown in the figure.)



The following relationships involving the changing segment lengths can be written:

1) Segments S_1 and S_2, plus a fixed length, F_G, sum to the total length of the green cord. L_G.
2) Segments S_3 and S_4, plus a fixed length, F_B, sum to the total length of the blue cord. L_B.
3) Segments S_5 and S_6, plus a fixed length, F_Y, sum to the total length of the yellow cord. L_Y.

Two additional relationships can be written:

4) Segments S_1, S_3, S_5 and the length of the floating link, L_{LINK}, sum to an unknown, but constant (fixed) length, F_1.
5) Segments S_4, S_5 and the length of the floating link, L_{LINK}, sum to an unknown, but constant (fixed) length, F_2.

In summary, the following five equations are written:

$$\begin{aligned} S_1 + S_2 + F_G &= L_G \\ S_3 + S_4 + F_B &= L_B \\ S_5 + S_6 + F_Y &= L_Y \\ S_1 + S_3 + S_5 + L_{LINK} &= F_1 \\ S_4 + S_5 + L_{LINK} &= F_2 \end{aligned}$$

Taking time derivatives, the segment lengths become velocities, and the constant terms go to zero.

$$\begin{aligned} v_1 + v_2 + 0 &= 0 \\ v_3 + v_4 + 0 &= 0 \\ v_5 + v_6 + 0 &= 0 \\ v_1 + v_3 + v_5 + 0 &= 0 \\ v_4 + v_5 + 0 &= 0 \end{aligned}$$

Since v_6 is known to be 0.3 m/s (downward), these 5 equations in 5 unknowns can be solved. Including the known value, the equations become…

$$
\begin{aligned}
v_1 + v_2 &= 0 \\
v_3 + v_4 &= 0 \\
v_5 &= -0.3 \\
v_1 + v_3 + v_5 &= 0 \\
v_4 + v_5 &= 0
\end{aligned}
$$

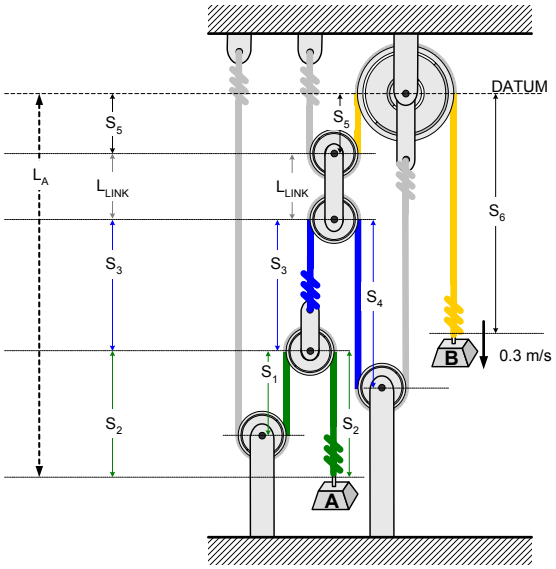
To solve these simultaneous linear equations using matrix methods in MATLAB, we rewrite the equations as a coefficient matrix, **C**, and a right-hand-side vector, **r**. Note that the apostrophe on the **r** vector transposes **r** from a row vector into a column vector.

```
>> C = [ 1   1   0   0   0;
         0   0   1   1   0;
         0   0   0   0   1;
         1   0   1   0   1;
         0   0   0   1   1; ];

>> r = [  0    0    -0.3    0    0 ]';                    %Units => m/s
```

We then invert the coefficient matrix, and multiply the inverted coefficient matrix and the **r** vector to obtain the unknown velocities.

```
>> C_inv  =  inv(C)

C_inv =

    0   -1   -2    1    1
    1    1    2   -1   -1
    0    1    1    0   -1
    0    0   -1    0    1
    0    0    1    0    0


>> v  =  C_inv * r                                      %Units => m/s

v =

   0.6000
  -0.6000
  -0.3000
   0.3000
  -0.3000
```

These velocities represent the rate at which segment lengths S_1 through S_5 are changing with time, and are not (generally) velocities relative to the datum. In order to determine the velocity of point A relative to the datum, the distance of point A relative to the datum (called L_A in the figure below) must be written in terms of segment lengths.



$$L_A = S_5 + L_{LINK} + S_3 + S_2$$

The velocity of point A relative to the datum is the time derivative of L_A.

$$\begin{aligned} v_A &= \frac{dL_A}{dt} \\ &= \frac{dS_5}{dt} + 0 + \frac{dS_3}{dt} + \frac{dS_2}{dt} \\ &= v_5 + 0 + v_3 + v_2 \end{aligned}$$

Velocities v_2, v_3, and v_5 are known, so v_A can be determined.

```
>> v_a  =  v(5) + 0 + v(3) + v(2)

v_a =
   -1.2000
```

Since the velocity at point B was 0.3 m/s downward, this result indicates that point A is moving at 1.2 m/s upward.

Annotated MATLAB Script Solution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Four Pulley System
% Calculate segment velocities and velocity of point A
% relative to the datum
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Declare the coefficient matrix and right-hand-side vector
C = [ 1 1 0 0 0;
      0 0 1 1 0;
      0 0 0 0 1;
      1 0 1 0 1;
      0 0 0 1 1;];

r = [ 0 0 -0.3 0 0]';  %Units => m/s
```

Annotated MATLAB Script Solution Continued

```
%Invert the coefficient matrix.
C_inv = inv(C);

%Multiply the inverted coefficient matrix and the right-hand-side
%vector to calculate the segment velocities.
v = C_inv * r; %Units => m/s
fprintf('\nThe Segment Velocitys are as follows:\n')
fprintf('\tv(1) = %3.1f m/s \n', v(1))
fprintf('\tv(2) = %3.1f m/s \n', v(2))
fprintf('\tv(3) = %3.1f m/s \n', v(3))
fprintf('\tv(4) = %3.1f m/s \n', v(4))
fprintf('\tv(5) = %3.1f m/s \n', v(5))

%Use the segment velocities to calculate the velocity of point A
%relative to the datum.
v_a  =  v(5) + 0 + v(3) + v(2);
fprintf('\nThe velocity of point A relative to the datum is %3.1f m/s \n',
v_a)
```

17 Kinetics of a Particle: Force and Acceleration

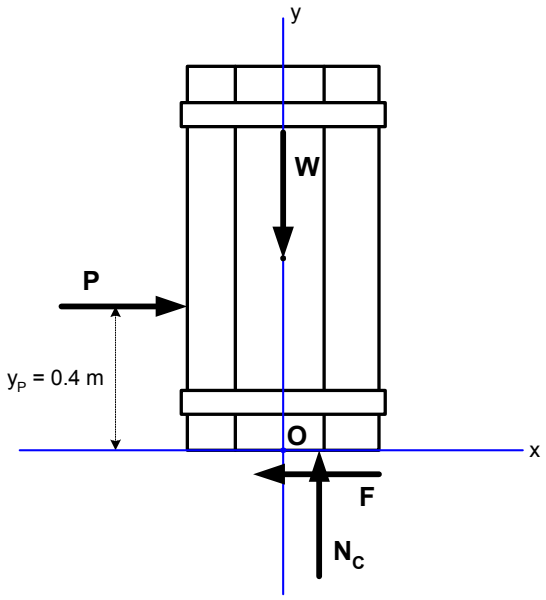
Ref: Hibbeler § 13.4, Bedford & Fowler: Dynamics § 3.1-3.4

In an earlier example (#8) we looked at “pushing a crate” and investigated the effect of changing the height of the push and the magnitude of the push on the rigid body. Specifically, we tested to see if the crate would tip or slip. In this example, we will determine the initial acceleration of the crate and the velocity after a short time at that acceleration.

Example: Pushing a Crate

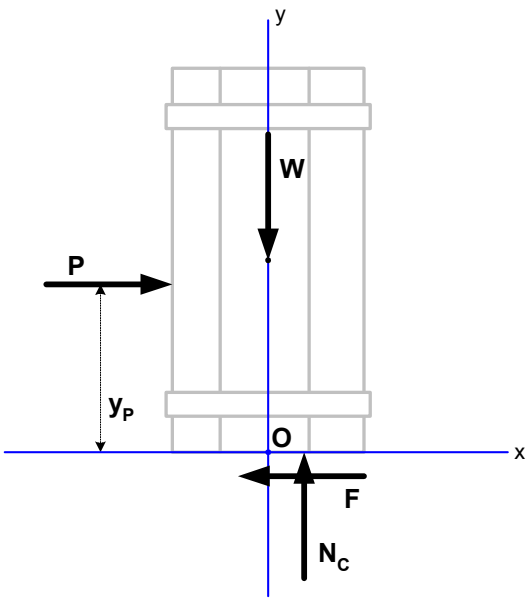
A crate weighs 35 kg, and is 0.5 m wide and 1.0 m in height. A force, P, is applied to the side of the crate at height y_P = 0.4 m from the floor. From the earlier example (#8) we know that this force is sufficient to overcome friction, and when applied at this height will not tip the crate. The coefficient of kinetic friction is 0.24.

Determine the initial acceleration and, assuming the acceleration remains constant, the expected velocity of the crate after 3 seconds.



Solution

First, a free-body diagram is drawn.



We start the solution by assigning values given in the problem statement to variables:

```
» M = 35;                                   %kg
» P = 120;                                  %Newtons
» y_p = 0.4;                                %meters
» g = 9.807;                                %meters/s^2
» mu_k = 0.24;
```

Then we calculate the weight of the crate.

```
» W = -M .* g                               %Newtons
W =
 -343.2450
```

We can use the equation of motion for the y-components of force to determine the resultant normal force, N_C (but the result is not much of a surprise.)

```
» %EQ of MOTION: y components of force.
» % W + N_c = 0          so...
» N_c = -W                                  %Newtons - acts in the +x direction
N_c =
  343.2450
```

Knowing N_C, the friction force can be determined.

```
» F= mu_k .* N_c;          %Newtons - this equation does not account for direction
» F = -F                   %Newtons - sign changed since F acts in -x direction
F =
  -82.3788
```

Next, we use the equations of motions with the information on the free-body diagram to solve for the acceleration, a_x.

```
» %EQ of MOTION: x components of force.
» % P + F = M * a_x      so...
» a_x = (P + F) ./ M                                    %m / s^2
a_x =
   1.0749
```

Now the velocity after 3 seconds can be determined.

```
» %Calculate the velocity after three seconds.
» v_o = 0;                                             %m / s
» delta_t = 3;                                         %s
» a = a_x;                                             %m / s^2
» v = v_o + a .* delta_t                               %m / s
v =
   3.2247
```

Annotated MATLAB Script Solution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Pushing a Crate - Initial Acceleration                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Data from the problem statement...
M = 35;                                                 %kg
P = 120;                                                %Newtons
y_p = 0.4;                                              %meters
g = 9.807;                                              %meters/s^2
mu_k = 0.24;

%Calculate the weight of the crate.
W = -M .* g;                                            %Newtons
fprintf('\nW = %1.1f N\t\t', W)

%EQ of MOTION: y components of force.
%W + N_c = 0 so...
N_c = -W;                                               %Newtons
fprintf('N_c = %1.1f N\n', N_c)

%Calculate teh friction force, F.
F= mu_k .* N_c;   %Newtons - this equation does not account for direction
F = -F;           %Newtons - Sign changed since F acts in -x direction
fprintf('F = %1.1f N\t\t', F)

%EQ of MOTION: x components of force.
%P + F = M * a_x so...
a_x = (P + F) ./ M;                                     %m/s^2
fprintf('a_x = %1.3f m/s^2\n\n', a_x)
```

```
%Calculate the velocity after three seconds.
v_o = 0;                                                %m / s
delta_t = 3;                                            %s
a = a_x;                                                %m/s^2
v = v_o + a .* delta_t;                                 %m/s
fprintf('v = %1.3f m/s\n\n', v)
```

18 Equations of Motion: Normal and Tangential Components

Ref: Hibbeler § 13.5, Bedford & Fowler: Dynamics § 3.4

In an earlier example (#14) we looked at the motion of a skateboarder along a curved path. Now, we apply the equations of motion to that problem to investigate the significance of friction in the problem.

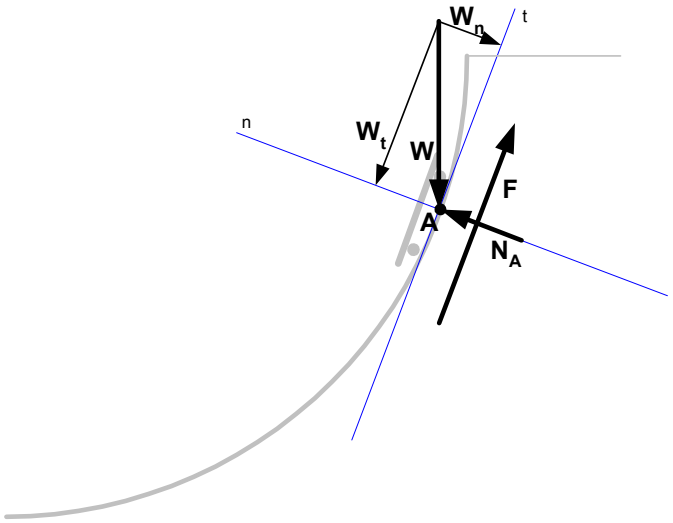
Example: A skateboarder in a quarter pipe.

A skateboarder with a mass of 55 kg has dropped into a 3 meter (radius) quarter pipe. When she's dropped a distance of 1 meter her speed is 3.2 m/s and is increasing by 7.8 m/s^2. At this location the surface is at an angle of 19.4° from vertical.

Draw a free body diagram of this system including a friction force. Neglect the size of the skateboarder and her board. Solve the equations of motion for the normal force, N_A, and the friction force, F. Then, calculate the coefficient of kinetic friction required to generate the observed friction force.

Solution

A free-body diagram for this system looks like the following:



First, information from the problem statement is entered into MATLAB.

```
» v = 3.2;                      %m / s
» a_t = 7.8;                    %m / s^2
» M = 55;                       %kg
» g = 9.807;                    %m / s^2
» alpha = 19.4 * pi /180;       %radians - angle from vertical
» rho = 3;                      %meters - circular path, constant rho
```

The tangential acceleration at this location was given in the problem statement, as $a_t = 7.8$ m/s^2. The normal acceleration can be calculated from the stated velocity (3.2 m/s) and the radius of curvature (3 m).

```
» a_n = v.^2 ./ rho                          %m / s^2
a_n =
    3.4133
```

The normal component of the equation of motion can be used to find N_A, but first we need the normal component of the skateboarder's weight.

```
» W = M .* g                                 %Newtons
W =
  539.3850

» W_n = -W .* sin(alpha)                     %Newtons - minus sign accounts for direction
W_n =
 -179.1627

» W_t =  W .* cos(alpha)                     %Newtons
W_t =
  508.7602
```

Then find the normal force, N_A.

```
» % N_A + W_n = M * a_n          so...
» N_A = M .* a_n - W_n                       %Newtons
N_A =
  366.8961
```

Next, use the tangential component of the equation of motion to determine the friction force.

```
» % W_t + F = M * a_t              so...
» F = M .* a_t - W_t                         %Newtons
F =
  -79.7602
```

The friction force, F, and the normal force, NA, can be used together to calculate the apparent coefficient of kinetic friction for this problem.

```
» mu_k = abs(F) ./ N_A                       %need only the magnitude of F here
mu_k =
    0.2174
```

Annotated MATLAB Script Solution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                Friction Force on a Skateboarder                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Set problem parameters as stated in the problem.
v = 3.2;                        %m/s
a_t = 7.8;                      %m/s^2
M = 55;                         %kg
g = 9.807;                      %m/s^2
alpha = 19.4 * pi /180;         %radians - angle from vertical
rho = 3;                        %meters - circular path, constant rho

%Solve for the normal acceleration.
a_n = v.^2 ./ rho;              %m/s^2
fprintf('\nNormal Acceleration = %1.1f M/s^2 \n', a_n)

%Find the normal and tangential components of the skateboarder's weight
W = M .* g;                     %Newtons
W_n = -W .* sin(alpha);         %Newtons - minus sign accounts for direction
W_t =  W .* cos(alpha);%Newtons
fprintf('Skateboarder''s Weight = %1.0f N\n', W)
fprintf('\tNormal component     = %1.0f N\n', W_n)
fprintf('\tTangential component =  %1.0f N\n', W_t)

%EQ of MOTION: normal component
%N_A + W_n = M * a_n so...
N_A = M .* a_n - W_n;           %Newtons
fprintf('Normal Force = %1.2f N\n', N_A)

%EQ of MOTION: tangential component
%W_t + F = M * a_t so...
F = M .* a_t - W_t;             %Newtons
fprintf('Friction Force = %1.1f N\n', F)

                                %Calculate the coefficient of kinetic friction
mu_k = abs(F) ./ N_A;           %need only the magnitude of F here
fprintf('Coefficient of Kinetic Friction = %1.2f\n\n', mu_k)
```

19 Principle of Work and Energy

Ref: Hibbeler § 14.3, Bedford & Fowler: Dynamics § 8.1-8.2

The *principle of work and energy* states that the work done by all of the external forces and couples as a rigid body moves between positions 1 and 2 is equal to the change in the body's potential energy. Hibbeler writes the resulting equation as

$$T_1 + \sum U_{1-2} = T_2 \qquad \text{Hibbeler (14-7)}$$

The same equation is written as

$$U_{12} = T_2 - T_1 \qquad \text{Bedford and Fowler: Dynamics (8.4)}$$

by Bedford and Fowler.

The difference between these two equations is simply nomenclature. Both $\sum U_{1-2}$ and U_{12} represent the sum of work done by all external forces and couples on the body. We can use the principle of work and energy to solve problems involving force, displacement, and velocity.

Example: Skid-to-Stop Braking Distance

The driver of a 1600 kg passenger car hits the brakes and skids to a stop to try to avoid hitting a deer. When the deer suddenly appeared in the headlights [at an estimated distance of 300 ft (91 m)] the car was moving at 75 mph (120 km/hr).

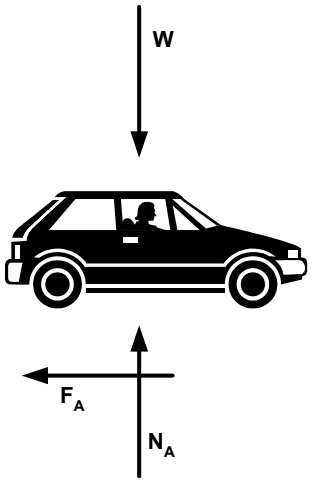
After the incident long skid marks were visible on the pavement, and the deer was seen running over a nearby hill. Did the driver get the car stopped in time, or did the deer manage to get out of the way?

Assumptions:

- Assume a 1.5 second reaction time – that is, it took 1.5 seconds from the time the driver saw the deer until he or she pressed the brake pedal.
- Assume a dry, flat road with a coefficient of friction, $\mu_k = 0.80$.

Solution

A free body diagram of this system shows the various forces acting on the car.



Because the road was assumed to be flat, the normal force, N_A, is equal to the vehicle's weight.

```
» M = 1600;                                  %kg
» g = 9.807;                                 %m / s^2
» W = M .* g;                                %Newtons
» N_A = W                                    %Newtons
N_A =
  15691
```

The friction force during the skid (wheels locked) is calculated using the normal force and the coefficient of kinetic friction, μ_k.

```
» mu_k = 0.80;
» F_A = mu_k .* N_A                          %Newtons
F_A =
  12553
```

The length of the skid can be determined using the principle of work and energy.

$$T_1 + \sum U_{1-2} = T_2$$

$$\tfrac{1}{2} m\, v_1^{\,2} + \sum U_{1-2} = \tfrac{1}{2} m\, v_2^{\,2}$$

The final velocity, v_2, is zero (skid-to-stop), and the only force acting on the car during the deceleration is the friction force, F_A, acting through the skid distance, s.

$$\tfrac{1}{2} m\, v_1^{\,2} + \left(-F_A\ s\right) = 0$$

This equation can be solved for the skid distance.

```
» v(1) = 75;                                 %mi / hr
» v(1) = v(1) .* 1609 ./ 3600;               %m / s
» s = 0.5 .* M .* v(1).^2 / F_A              %meters
s =
  71.6100
```

That looks good; it is possible to stop in less than 91 meters, the initial distance between the car and the deer. But, we also need to account for the distance the car traveled in the 1.5 seconds between the time the driver saw the deer and the moment the brakes were applied.

```
» t_reaction = 1.5;                          %sec
» s_reaction = v(1) .* t_reaction            %meters
s_reaction =
  50.2813
```

The total distance traveled between the moment the deer was spotted and the vehicle coming to a complete stop is 50.2813 + 71.6100 = 121.8912m.

```
» s_total = s + s_reaction                   %meters
s_total =
  121.8912
```

The driver did not get stopped within 91 m. The deer had to get out of the way in order to escape injury.

Annotated MATLAB Script Solution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Skid-to-Stop Calculations                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Calculate the magnitude of the normal force.
M = 1600;                                           %kg
g = 9.807;                                          %m/s^2
W = M .* g;                                         %Newtons
N_A = W                                             %Newtons
fprintf('Magnitude of the normal force = %1.0f N\n', N_A)

%Calculate the friction force.
mu_k = 0.80;
F_A = mu_k .* N_A;                                  %Newtons
fprintf('Friction force = %1.0f N\n', F_A)

%Calculate the skid distance
v(1) = 75;                                          %mi/hr
v(1) = v(1) .* 1609 ./ 3600;                        %m/s
s = 0.5 .* M .* v(1).^2 / F_A ;                     %meters
fprintf('Skid distance = %1.0f m\n', s)

%Calculate the distance traveled during the driver's reaction time.
t_reaction = 1.5;                                   %sec
s_reaction = v(1) .* t_reaction;                    %meters
fprintf('Distance traveled during the drivers reaction ')
fprintf('time = %1.0f m\n', s_reaction)

%Calculate the total distance traveled after spotting the deer.
s_total = s + s_reaction;                           %meters
fprintf('Total distance traveled after spotting the ')
fprintf('deer %1.0f m\n', s_total)
```

20 Rotation About a Fixed Axis

Ref: Hibbeler § 16.3, Bedford & Fowler: Dynamics § 9.1

Because drive motors are routinely used, solving problems dealing with rotation about fixed axes is commonplace. The example used here looks at a very old-fashioned drive motor – a water wheel.

Example: Water Wheel

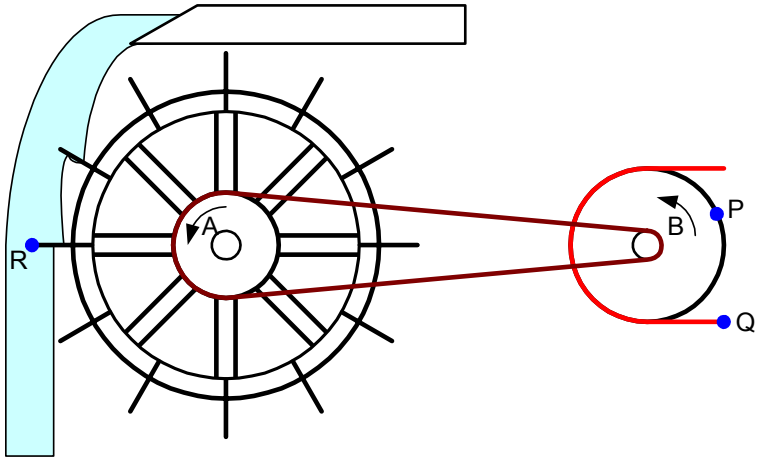
Long ago, a water wheel was used to drive a mill. The point labeled "R" on the outermost edge of the water wheel was 1.25 m from the center of the wheel. A drive wheel, labeled "A", was directly attached to the water wheel. A belt connected the drive wheel to the shaft on wheel "B". The dimensions of the various wheels and shafts are listed below.

Shortly after the water gate was opened to allow water to flow over the wheel, then water wheel had an angular acceleration of 0.1 rad/s^2. Assume the belts did not slip.

Determine:

a) The magnitudes of the velocities and accelerations at points R and P after the water wheel has made two complete revolutions.

b) The velocity and acceleration of point Q on the belt leaving wheel B.



Dimensions	Diameter
Wheel A	0.7 m
Wheel B	1 m
Shaft B	0.19 m

Solution, part a (point R)

Movement through a complete circle is equivalent to moving a point through 2π radians, so two complete revolutions of point R would move that point through 4π radians, or 4 x 3.1416 = 12.57 radians. So,

$$\theta_R = 12.57 \text{ rad}$$

The water wheel is accelerating at constant rate, $\alpha_A = 0.1 \text{ rad/s}$, so the angular velocity of point R is

$$\omega_R{}^2 = \omega_0{}^2 + 2\alpha_A(\theta_R - \theta_0)$$
$$= 0 + 2\left(0.1 \text{ rad/s}^2\right)(12.57 \text{ rad} - 0)$$
$$= 2.51 \text{ rad}^2/\text{s}^2$$
$$\omega_R = 1.59 \text{ rad/s}$$

In MATLAB, this calculation can be performed as follows:

```
» alpha_A = 0.1;                                          %radians / s^2
» omega_0 = 0;                                            %radians / s
» theta_0 = 0;                                            %radians
» theta_R = 4 * pi;                                       %radians
» omega_R = sqrt(omega_0.^2 + 2 .* alpha_A .* (theta_R - theta_0))   %radians / s
omega_R =
   1.5853
```

The velocity at point R can now be determined.

```
» r_R = 1.25;                                             %meters
» v_R = omega_R *r_R                                      %m / s
v_R =
   1.9817
```

Point R has tangential and normal components of acceleration. They are calculated as follows.

```
» a_Rt = alpha_A .* r_R                                   %m / s^2
a_Rt =
   0.1250


» a_Rn = omega_R.^2 .* r_R                                %m / s^2
a_Rn =
   3.1416
```

Then the magnitude of the acceleration at point R can be determined.

```
» a_R = sqrt(a_Rt.^2 + a_Rn.^2)                           %m / s^2
a_R =
   3.1441
```

Solution, part a (point P)

The connector between drive wheel A and wheel B is the belt between the two wheels. If the belt does not slip, then

$$s = \theta_A r_A = \theta_B r_{B\_shaft}$$

Notice that the radius of the shaft on wheel B is used in this calculation, because the belt from wheel A goes around the shaft on wheel B, not wheel B itself. However, the angular displacement of the shaft on wheel B is the same as the angular displacement of wheel B – they are directly connected.

```
» theta_A = theta_R                                  %radians
theta_A =
   12.5664
» r_A = 0.35;                                        %meters
» r_B_shaft = 0.19 ./ 2;                             %meters
» theta_B = theta_A .* r_A ./ r_B_shaft              %radians
theta_B =
   46.2972
```

Since the belt connecting wheel A and shaft B has the same speed and tangential component of acceleration, we can write:

$$v = \omega_A r_A = \omega_B r_{B\_shaft}$$

$$a_t = \alpha_A r_A = \alpha_B r_{B\_shaft}$$

These relationships can be used to find the angular velocity and tangential component of acceleration of shaft B, which are also the angular velocity and tangential component of acceleration of point P on wheel B.

```
» omega_A = omega_R;                                 %radians / s
» omega_B = omega_A .* r_A ./ r_B_shaft              %radians / s
omega_B =
   5.8407
» alpha_B = alpha_A .* r_A ./ r_B_shaft              %radians / s^2
alpha_B =
   0.3684
```

Now it is possible to calculate the magnitudes of the velocity and acceleration at point P.

```
» r_P = 0.5;                                         %meters
» v_P = omega_B .* r_P                               %m / s
v_P =
   2.9203
» a_Pt = alpha_B .* r_P                              %m / s^2
a_Pt =
   0.1842
» a_Pn = omega_B.^2 .* r_P                           %m / s^2
a_Pn =
   17.0568
» a_P = sqrt(a_Pt.^2 + a_Pn.^2)                      %m / s^2
a_P =
   17.0578
```

Solution, part b (point Q)

Point Q has the same velocity and tangential component of acceleration as point P.

```
» v_Q = v_P                                                    %m / s
v_Q =
   2.9203
» a_Q = a_Pt                                                   %m / s^2
a_Q =
   0.1842
```

Annotated MATLAB Script Solution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          Water Wheel Problem                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Calculate the angular velocity of the water wheel (and point R).
alpha_A = 0.1;                                                 %radians/s^2
omega_0 = 0;                                                   %radians/s
theta_0 = 0;                                                   %radians
theta_R = 4 * pi;                                              %radians
omega_R = sqrt(omega_0.^2 + 2 .* alpha_A .* (theta_R-theta_0)); %radians/s
fprintf('\nAngular Velocity of the Water Wheel = %1.2f rad/s\n', omega_R)

%Find the velocity at piont R.
r_R = 1.25;                                                    %meters
v_R = omega_R *r_R;                                            %m/s
fprintf('Velocity at Piont R = %1.2f m/s\n', v_R)

%Find the tangential and angular components of acceleration at point R.
a_Rt = alpha_A .* r_R;                                         %m/s^2
a_Rn = omega_R.^2 .* r_R;                                      %m/s^2
fprintf('Components of acceleration at point\n')
fprintf('\tTangential component = %1.3f m/s^2\n', a_Rt)
fprintf('\tAngular component    = %1.3f m/s^2\n', a_Rn)

%Determine the magnitude of the acceleration at point R.
a_R = sqrt(a_Rt.^2 + a_Rn.^2);                                 %m/s^2
fprintf('Magnitude of the Acceleration at point R = %1.3f m/s^2\n', a_R)

%Determine the angular displacement of wheel B.
theta_A = theta_R;                                             %radians
r_A = 0.35;                                                    %meters
r_B_shaft = 0.19 ./ 2;                                         %meters
theta_B = theta_A .* r_A ./ r_B_shaft;                         %radians
fprintf('Angular Displacement of Wheel B = %1.1f rad\n', theta_B)

%Determine the angular velocity and tangential component of acceleration
of shaft B (and point P).
omega_A = omega_R;                                             %radians/s
omega_B = omega_A .* r_A ./ r_B_shaft;                         %radians/s
alpha_B = alpha_A .* r_A ./ r_B_shaft;                         %radians/s^2
fprintf('Angular Velocity of Shaft B = %1.2f rad/s\n', omega_B)
```

```
fprintf('Tangential Component of Acceleration of ')
fprintf('Shaft B = %1.2f rad/s^2\n', alpha_B)

%Find the velocity and acceleration at point P.
r_P = 0.5;%meters
v_P = omega_B .* r_P;                                          %m/s
a_Pt = alpha_B .* r_P;                                         %m/s^2
a_Pn = omega_B.^2 .* r_P;                                      %m/s^2
a_P = sqrt(a_Pt.^2 + a_Pn.^2);                                 %m/s^2
fprintf('\nVelocity at point P     = %1.2f m/s\n', v_P)
fprintf('Acceleration at point P = %1.1f m/s^2\n', a_P)

%Find the velocity and acceleration of point Q.
v_Q = v_P;                                                     %m/s
a_Q = a_Pt;                                                    %m/s^2
fprintf('\nVelocity at point Q     = %1.2f m/s\n', v_Q)
fprintf('Acceleration at point Q = %1.1f m/s^2\n', a_Q)
```