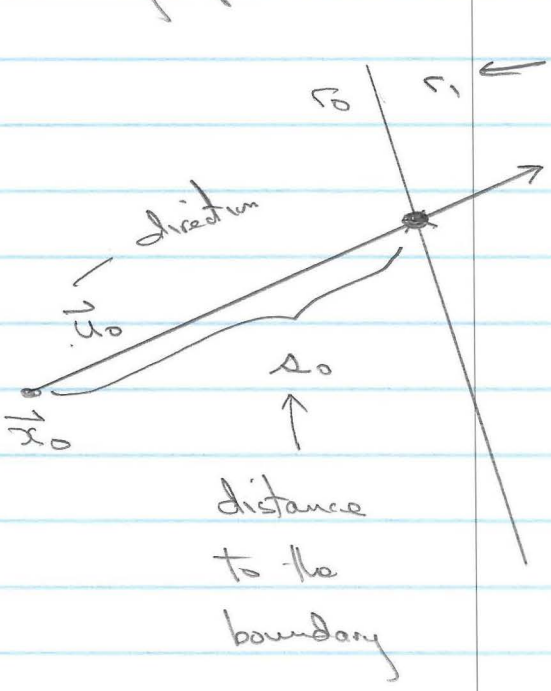
Lecture 6

Geometry

And now, finally it is time to discuss how particles are tracked in geometries of arbitrary specification



Essential knowledge a particle has about itself are

$\{\vec{x}_0, \vec{u}_0, r_0, \ldots\}$ as part of its phase space

It knows how long a step it wants to take based on

$$s = -\Sigma^{-1} \log \xi$$ ← random number

(distance to an interaction)

or some other constraint that arise from physics

Then the geometry answers the following question

- Does the particle encounter (hit) a boundary on its flight shorter than s? (If there are multiple, the answer is chosen to be the minimum)

If this distance is called, say, d_0

transport to the boundary, namely

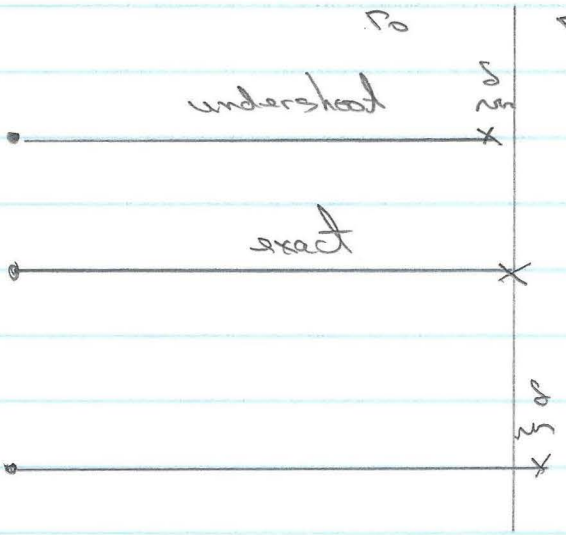
$$\vec{x}_1 = \vec{x}_0 + \vec{v}_0 d_0$$

Then, update any data tables for the new region (e.g. Σ might change), and label the new region number as r_1

10.1 Boundary Crossing

When the transport step takes place, floating-point arithmetic is employed.
The precision of floating point arithmetic
is $\delta \sim 10^{-7}$ (floats or single) 32-bits
or $\delta \sim 2\times10^{-16}$ (doubles) 64-bits
it is never 0!

Consequently, 3 events can occur.
Imagine that transport to a plane $z=0$ has just occured

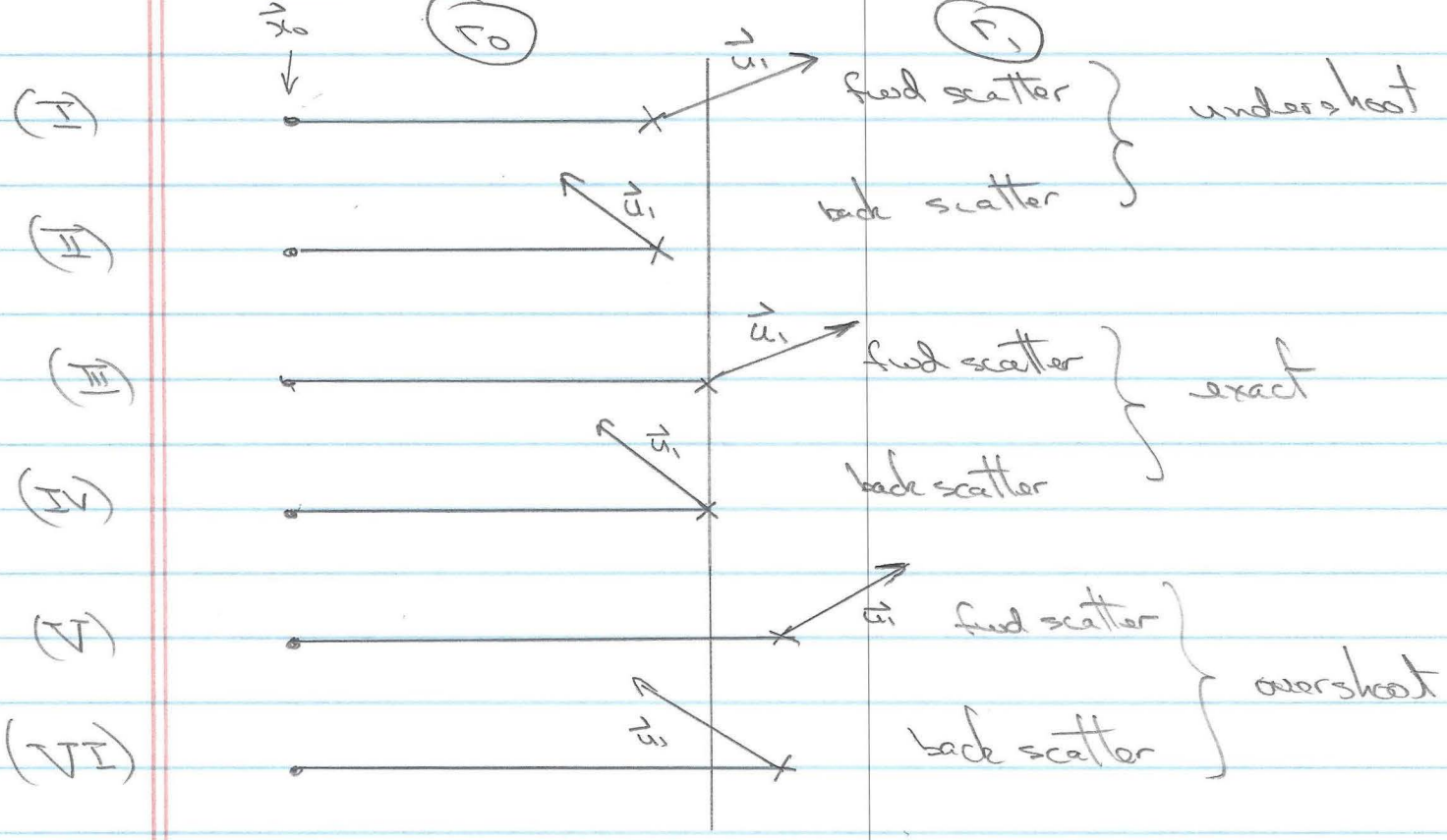


But wait! It gets worse!

Each of these transport steps may have a deflection!

(e.g. – multiple scattering angle
– EM field deflection
– boundary is a mirrored surface (important variance-reduction method))

So, the actual situation is



If one bases one's decision on position, namely

```
if (z_1 < 0) {
    "In region r0" }
else if (z_1 > 0) {
    "In region r_1" }
else {
    "indeterminate"
    if (z · û_1 > 0) {
        "in region r_1" }
    else if (z · û_1 < 0) {
        "in region r0" }
    else {
        "I'm lost" }
}
```

This is how particles get lost in Monte Carlo.

There is a general fix-up!

The fix-up

- Give the particle knowledge of whether it is inside or outside of a surface to start with

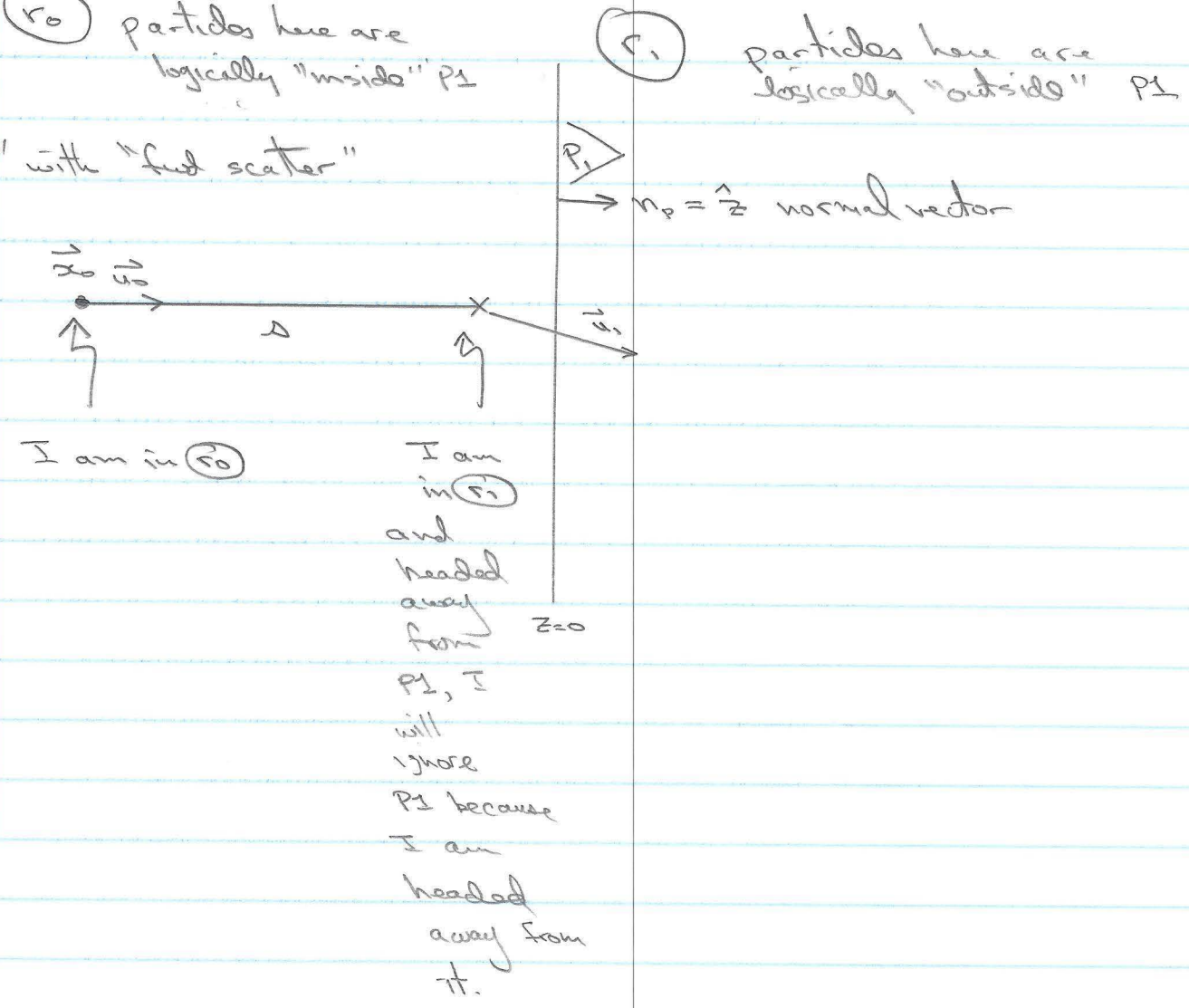
- Do not calculate flight distances to surfaces that it is headed away from based on its direction

- If the flight distance is ≤ 0, set it to 0 and assume that the transport makes a region change, to the previous region

Here is how it works

Case I

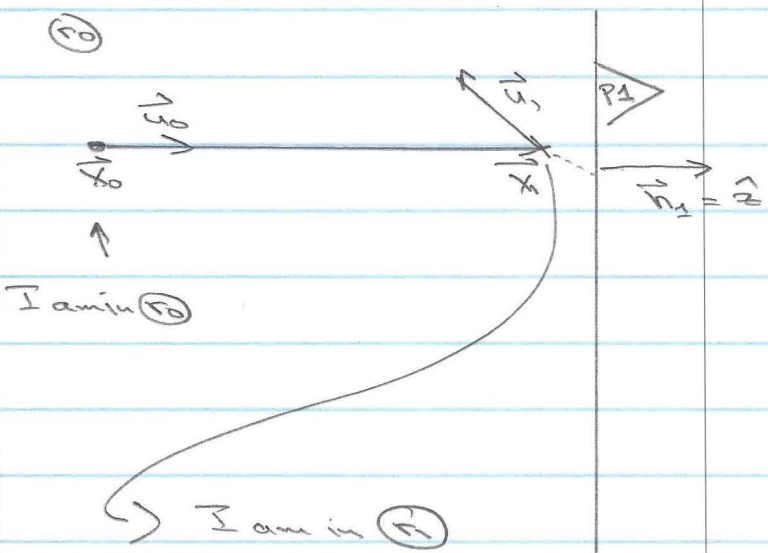
"undershoot" with "fwd scatter"



✓ no problem

Case II

"undershoot" with back scatter



I am in (r1)
I am headed
toward P1
Transport distance
is -ve!
Set transport
distance to 0
and reset the
region # to (r0)
✓ problem fixed

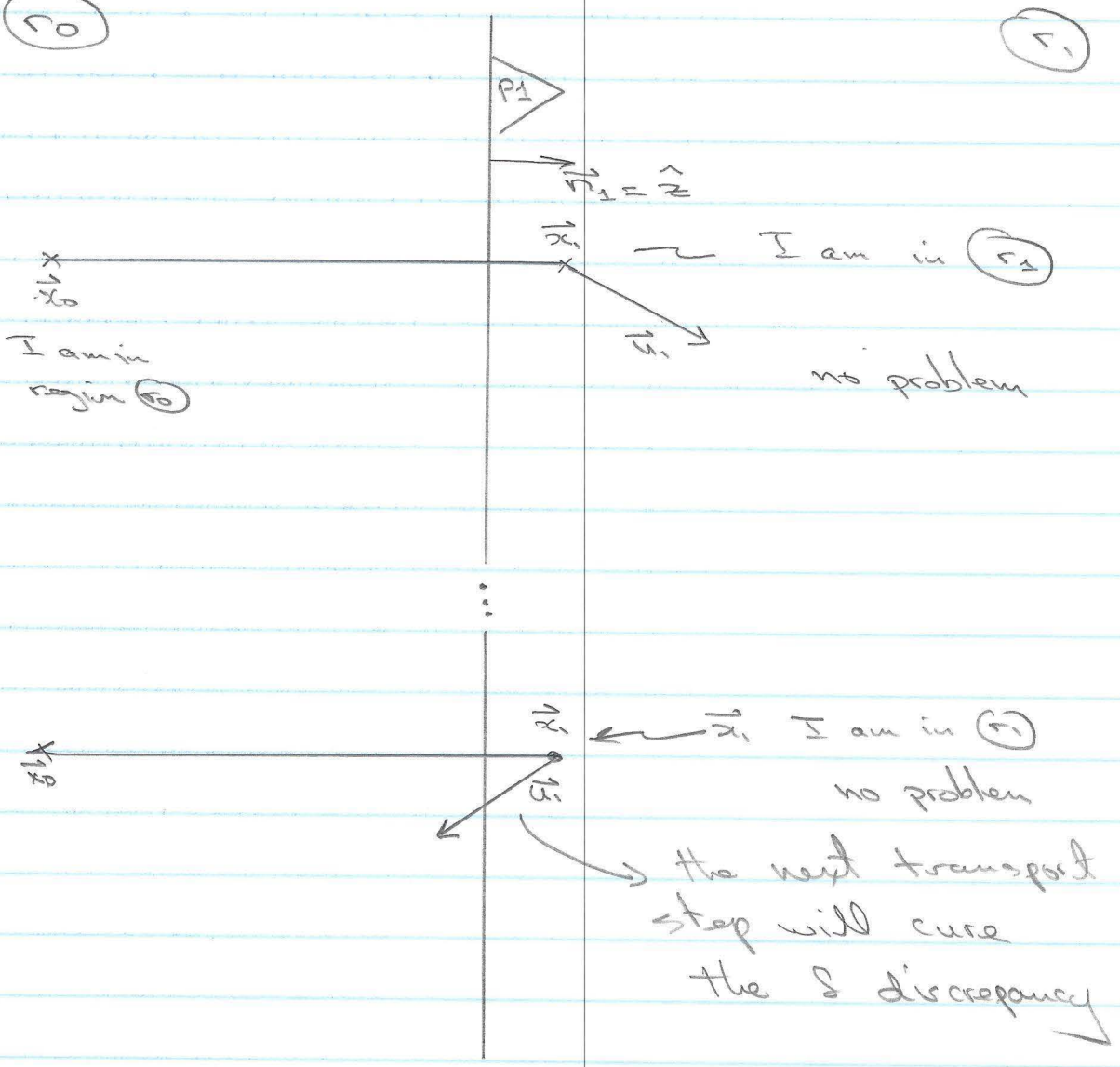
Case III & IV execute exactly as I & II
but with $\delta = 0$

II ✓ no problem

IV ✓ problem fixed

Case V

"overshoot" with fwd scatter



10.2 Solutions for Simple Surfaces

10.2.1

The general equation for a plane of arbitrary orientation is:

(10.2) $\hat{n} \cdot (\vec{x} - \vec{P}) = 0$

where
$\hat{n}$ is a unit normal to the plane (pointed at the "outside")
$\vec{P}$ is any point on the plane

e.g. $z = 0$ is the equation of a plane that intersects the z axis at $z=0$ and contains all (x,y) points

Aside on geometry

For surfaces of the Form

$$F(x,y,z) = F(\vec{x}) = 0$$

the normal to the surface is given by

$$\nabla F \qquad (1)$$

e.g. $F(x,y,z) = z = 0$ (this example)

$$\nabla F = \hat{z}$$

So, given (10.2)

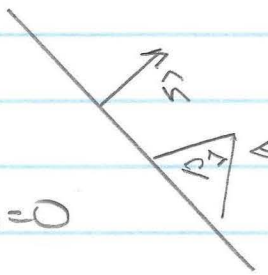
$$F(x,y,z) = n_x x + n_y y + n_z z - \hat{n}\cdot\vec{P} = 0$$

$$\Rightarrow F = \hat{n}$$

By definition

$\hat{n}(x,y,z)$ points to the outside

So, we shall draw the plane as



when constructing geometries, draw them this way.

(10.2) $\hat{n}\cdot(\vec{x}-\vec{P}) = 0$

distance to the plane:

Substitute into (10.2)

$$\vec{x} = \vec{x}_0 + \vec{u}\,\Delta \qquad (10.1)$$

$$\hat{n} \cdot (\vec{x}_0 + s\vec{u} - \vec{p})$$

Solving for

$$\boxed{s = -\frac{\hat{n} \cdot (\vec{x}_0 - \vec{p})}{\hat{n} \cdot \vec{u}}} \quad (10.3)$$

s is the distance to the plane from $\vec{x} = \vec{x}_0$ going in direction $\vec{u}$

Pseudo code

A pseudo code for (10.3) is given in the text pp. 112-113, but a more up to date version is given in the Matlab coding

<plane.m> (demo)

```
function [hit,s] = plane(xp,np,x0,u,inside)

% Calculates the distance of a particle to a planar surface normal to the
% an arbitrary axis

% Inputs
%%%%%%%%
%  double
%     xp:  coordinate that the plane passes through (row vector)
%     np:  normal direction of the plane (row vector)
%     x0:  coordinates of the particle (row vector)
%     u :  direction cosines of the particle (row vector)
%  logical
%     inside: == true   particle thinks it is inside
%           : == false  particle thinks it is outside
%

% Outputs
%%%%%%%%%
%  logical
%     hit: == true   particle would hit  the surface
%        : == false  particle would miss the surface
%
%  double
%     s: distance to the surface (if hit)

npdotu = np*u'; % ' is the transpose operator, and this is the dot product of
                % the direction vector with the normal to the plane
if ((inside & npdotu > 0) | (~inside & npdotu < 0)) % headed towards the surface
   hit = true;
   s = max(0,np*(xp-x0)'/npdotu); % Note: equation was rearranged for efficiency
                                  % The max function corrects for
                                  % delta ambiguities, signaling a region change
   return
else
   % If processing enters here, it is moving away from the plane in question
   hit = false;
   s = inf; % inf is a Matlab keyword for floating positive infinity
   return
end
```

```
function [hit,s] = zplane(zp,z0,w,inside)

% Calculates the distance of a particle to a planar surface normal to the
% z-axis

% Inputs
%%%%%%%%
%  double
%     zp:  z-coordinate that the plane passed through
%     z0:  z-coordinate of the particle
%     w :  z-axis direction cosine of the particle
%  logical
%     inside: == true   particle thinks it is inside
%           : == false  particle thinks it is outside
%

% Outputs
%%%%%%%%%
%  logical
%     hit: == true   particle would hit  the surface
%        : == false  particle would miss the surface
%
%  double
%     s: distance to the surface (if hit)

if ((inside & w > 0) | (~inside & w < 0)) % headed towards the surface
   hit = true;
   s = max(0,(zp-z0)/w);
   return
else
   hit = false;
   s = inf;
   return
end
```

```
function [hit,s] = yplane(yp,y0,v,inside)

% Calculates the distance of a particle to a planar surface normal to the
% y-axis

% Inputs
%%%%%%%%
%  double
%     yp:  y-coordinate that the plane passed through
%     y0:  y-coordinate of the particle
%     v :  y-axis direction cosine of the particle
%  logical
%     inside: == true   particle thinks it is inside
%           : == false  particle thinks it is outside
%

% Outputs
%%%%%%%%%
%  logical
%     hit: == true   particle would hit  the surface
%        : == false  particle would miss the surface
%
%  double
%     s: distance to the surface (if hit)

if ((inside & v > 0) | (~inside & v < 0)) % headed towards the surface
   hit = true;
   s = max(0,(yp-y0)/v);
   return
else
   hit = false;
   s = inf;
   return
end
```

```
function [hit,s] = xplane(xp,x0,u,inside)

% Calculates the distance of a particle to a planar surface normal to the
% x-axis

% Inputs
%%%%%%%%
%  double
%     xp:  x-coordinate that the plane passed through
%     x0:  x-coordinate of the particle
%     u :  x-axis direction cosine of the particle
%  logical
%     inside: == true   particle thinks it is inside
%           : == false  particle thinks it is outside
%

% Outputs
%%%%%%%%%
%  logical
%     hit: == true   particle would hit  the surface
%        : == false  particle would miss the surface
%
%  double
%     s: distance to the surface (if hit)

if ((inside & u > 0) | (~inside & u < 0)) % headed towards the surface
   hit = true;
   s = max(0,(xp-x0)/u);
   return
else
   hit = false;
   s = inf;
   return
end
```

Two simple examples of "simple" "quadrics"

Quadric: A surface open (e.g. a cone) or closed (e.g. a sphere) for which a straight line can intercept it at most twice.
(or once [tangent] or not at all)

The sphere centered at $\vec{x} = \vec{0}$

$\vec{x} \cdot \vec{x} = x^2 + y^2 + z^2 = R^2$ R is the sphere's radius

Substitute $\vec{x} = \vec{x}_0 + \vec{u}s$ into the above

$$(\vec{x}_0 + \vec{u}s) \cdot (\vec{x}_0 + \vec{u}s) = R^2$$
$$\vec{x}_0^2 + 2\vec{u} \cdot \vec{x}_0 s + \underbrace{\vec{u}^2}_{=1} s^2 = R^2$$

Or $s^2 + 2\vec{u} \cdot \vec{x}_0 s + \vec{x}_0^2 = R^2$

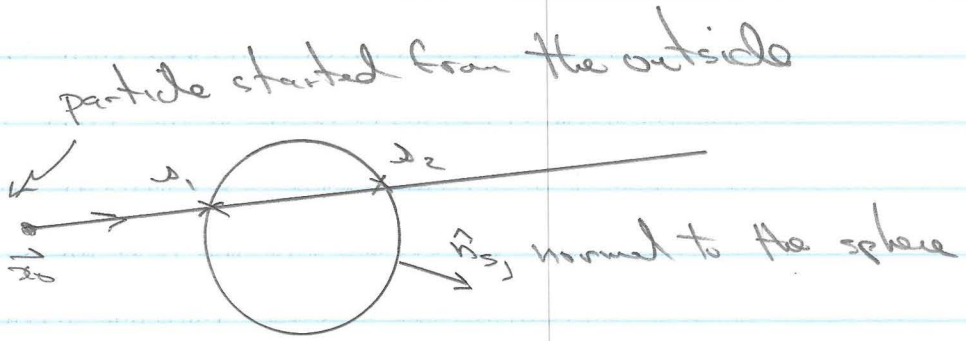
The general solution to this is

$$s = -\vec{u} \cdot \vec{x}_0 \pm \sqrt{(\vec{u} \cdot \vec{x}_0)^2 - (\vec{x}_0^2 - R^2)} = 0$$

The classification of solutions:

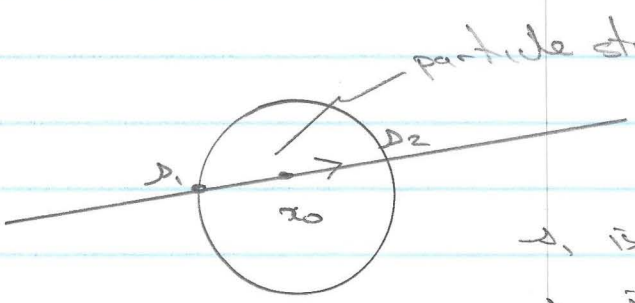
1) Two real solutions

a) Two fwd hits



Use the first s_1.

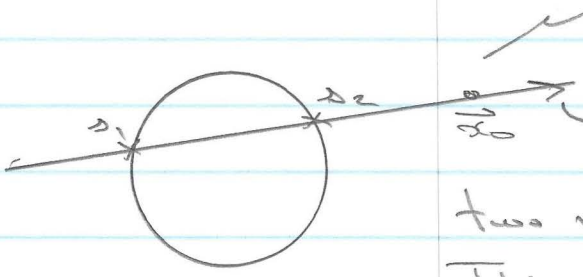
b) One +ve one -ve "hit"



s_1 is -ve, backward direction

s_2 is the only solution in the fwd direction

c) Two -ve hits



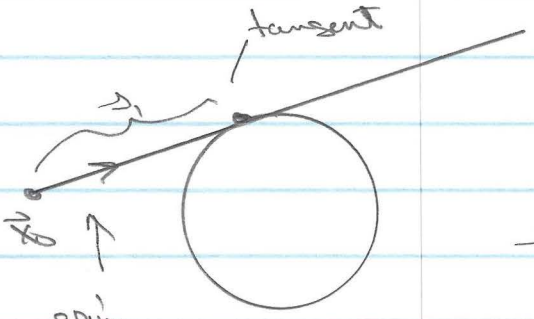
two negative solns

This is a "miss"

2) Radical, the $\sqrt{\ } = 0$

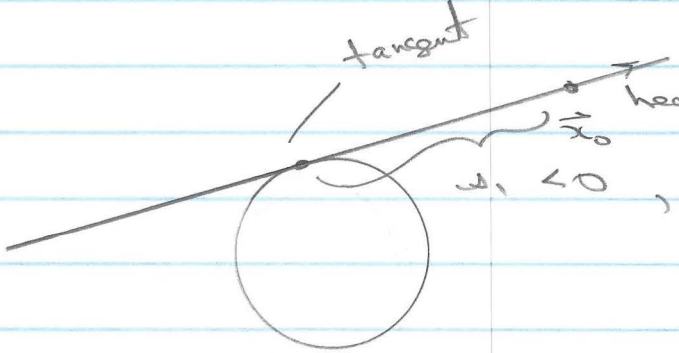
This is a tangent case

a) the solution



$\vec{v}_1$ is the solution in this case

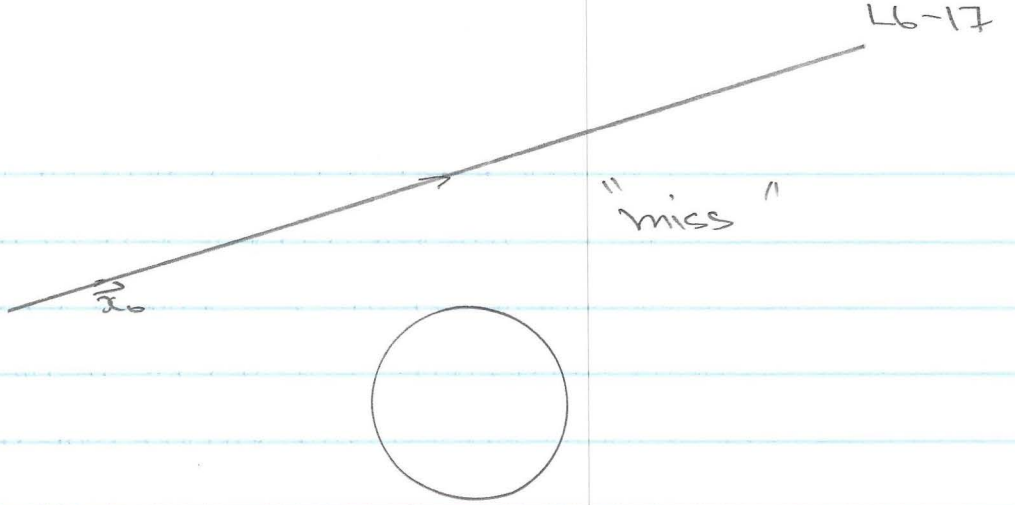
b) -ve solution



3) There are no real solutions

i.e. $\sqrt{(\ )}$ the radical is imaginary

-ve



Special cases.

4) The particle starts off "on" the sphere

recall $s^2 + 2s\vec{u}\cdot\vec{x}_0 + \vec{x}_0^2 - R^2 = 0$

If the particle is on the sphere

$\vec{x}_0^2 - R^2 = 0$

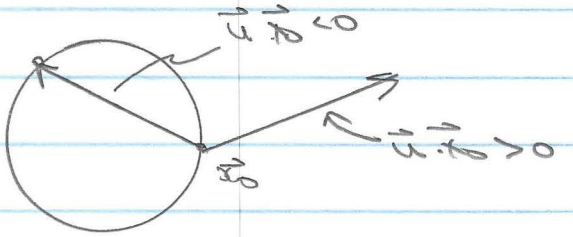
$\therefore \quad s(s + 2\vec{u}\cdot\vec{x}_0) = 0$

$s_1 = 0$ – ignore (for now)

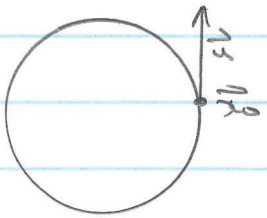
$s_2 = -2\vec{u}\cdot\vec{x}_0$

s_2 is true only if $\vec{u}\cdot\vec{x}_0 < 0$

a)



b) The particle is on the sphere and tangent to it



$$s^2 + 2s\,\vec{v}\cdot\vec{r}_0 + (\vec{r}_0^{\,2} - R^2) = 0$$

$\vec{v}\cdot\vec{r}_0 = 0$: tangent

$\vec{r}_0^{\,2} - R^2 = 0$: "on"

$$s^2 = 0$$

Interpretting this requires that we refine our concept of "inside" & "outside"

In the case of a sphere, in this case, it is always going to the outside

The circular cylinder centered & aligned on the z-axis

In this case, the surface of the cylinder is

$$F(\vec{x}) = 0 = x^2 + y^2 - R^2$$

Substituting $\vec{x} = \vec{x}_0 + \vec{u}s$

$$(x_0 + us)^2 + (y_0 + vs)^2 - R^2 = 0$$

recall $\vec{u} = (u, v, w)$

or $$s^2(u^2+v^2) + 2(x_0u + y_0v)s + x_0^2 + y_0^2 - R^2 = 0 \quad (1)$$

$$s = \frac{-(x_0u + y_0v) \pm \sqrt{(x_0u + y_0v)^2 - (x_0^2 + y_0^2 - R^2)}}{u^2 + v^2}$$

All the same consideration for the sphere apply except for one special case

If the particle is on the cylinder

$$x_0^2 + y_0^2 - R^2 = 0$$

$$0 = s(s + 2(x_0u + y_0v))$$

same conclusion

If the particle is on the cylinder, $x_0^2+y_0^2-R^2=0$ and tangent, we solve

$$s^2(u^2+v^2) + 2s(ux_0+vy_0) + x_0^2+y_0^2-R^2 = 0$$

$ux_0+vy_0 = 0$ "tangent"

$x_0^2+y_0^2-R^2 = 0$ "on"

$$s^2(u^2+v^2) = 0$$

If $u^2+v^2 \neq 0$, same as before

If $u^2+v^2 = 0$, it is travelling up $w=1$, or down $w=-1$ the $\hat{z}$ axis!

What do we do?

Technically, a line that is completely embedded in a surface, is called a ruling of the surface.

Surfaces that have rulings are planes, cylinders, cones

10.3 General solution for an arbitrary quadric surface.

An arbitrary quadric may be expressed as

① $F(x,y,z) = Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fzx + Gx + Hy + Iz + J = 0$

($Ax^2 + By^2 + Cz^2$: 3 x^2 forms; $Dxy + Eyz + Fzx$: 3 cross terms; $Gx + Hy + Iz$: linear terms; J: constant)

A ... I are 10 real constants

The linear + constant term with

$$A = B = C = D = E = F = 0$$

describes the arbitrary plane, already done

Eqn ① above has apparently 10 independent constants.

Actually, there are 9, since ① may be divided by any non-zero constant among the 10 to produce the same surface, since

$$F(x,y,z) = 0$$

So, there are really only 9 independent constants

We will now adopt a more compact notation.

Let $x_i = (1, x, y, z)$

0th component, used $\vec{x}$

Then we may write

$$F(\vec{x}) = F(x,y,z) = \sum_{i,j=0}^{3} a_{ij} x_i x_j \qquad (10.4)$$

a_{ij} is symmetric about the main diagonal,

namely $a_{ij} = a_{ji}$

After rotation and translation, it turns out that there are only 10 independent terms:

can be combined to make geometrical objects of arbitrary complexity and are extremely useful in Monte Carlo modeling of physical objects.

Despite having apparently 10 independent constants, Equation 10.4 represents only 10 independent real surfaces (including the simple plane), unique after a translation and rotation to standard position. The three cross terms (a_{ij} for $i \neq j$ and $i, j \geq 1$) can be eliminated by rotation. The resultant equation then only involves terms like x_i^2 and x_i. In addition, providing that a given variable's quadratic constant is non-zero, the linear terms can be eliminated by a translation. The result is that there are only two generic forms:

$$f(\vec{x}) = \sum_{i=1}^{3} a_i x_i^2 + c = 0, \tag{10.5}$$

and

$$f(\vec{x}) = \sum_{i=1}^{2} a_i x_i^2 + b x_3 = 0. \tag{10.6}$$

Equations 10.5 and 10.6 describe only 10 distinct possibilities with real solutions.

1. **ellipsoids:** $a_1^2 x_1^2 + a_2^2 x_2^2 + a_3^2 x_3^2 - c^2 = 0.$
2. **cones:** $a_1^2 x_1^2 + a_2^2 x_2^2 - a_3^2 x_3^2 = 0.$
3. **cylinders:** $a_1^2 x_1^2 + a_2^2 x_2^2 - c^2 = 0.$
4. **hyperboloids of one sheet:** $a_1^2 x_1^2 + a_2^2 x_2^2 - a_3^2 x_3^2 - c^2 = 0.$
5. **hyperboloids of two sheets:** $a_1^2 x_1^2 + a_2^2 x_2^2 - a_3^2 x_3^2 + c^2 = 0.$
6. **elliptic paraboloids:** $a_1^2 x_1^2 + a_2^2 x_2^2 + a_3 x_3 = 0.$
7. **hyperbolic paraboloids:** $a_1^2 x_1^2 - a_2^2 x_2^2 + a_3 x_3 = 0.$
8. **hyperbolic cylinders:** $a_1^2 x_1^2 - a_2^2 x_2^2 + c^2 = 0.$
9. **parabolic cylinders:** $a_1^2 x_1^2 + a_3 x_3 = 0.$
10. **simple planes:** $a_3 x_3 + c = 0.$

The first nine of these are shown[3] in Figure 10.2. (The magnitude of the above constants were all chosen to be unity for the purposes of display. Consequently, the first six of these surfaces shown exhibit at least one axis of rotational symmetry.)

There are other imaginary surfaces (*e.g.* imaginary ellipsoids $a_1^2 x_1^2 + a_2^2 x_2^2 + a_3^2 x_3^2 + c^2 = 0$) that we will not consider nor will we consider quadrics that can be made up of two

[3]Thanks to Keath Borg (wherever you are!) for generating the data for these figures.

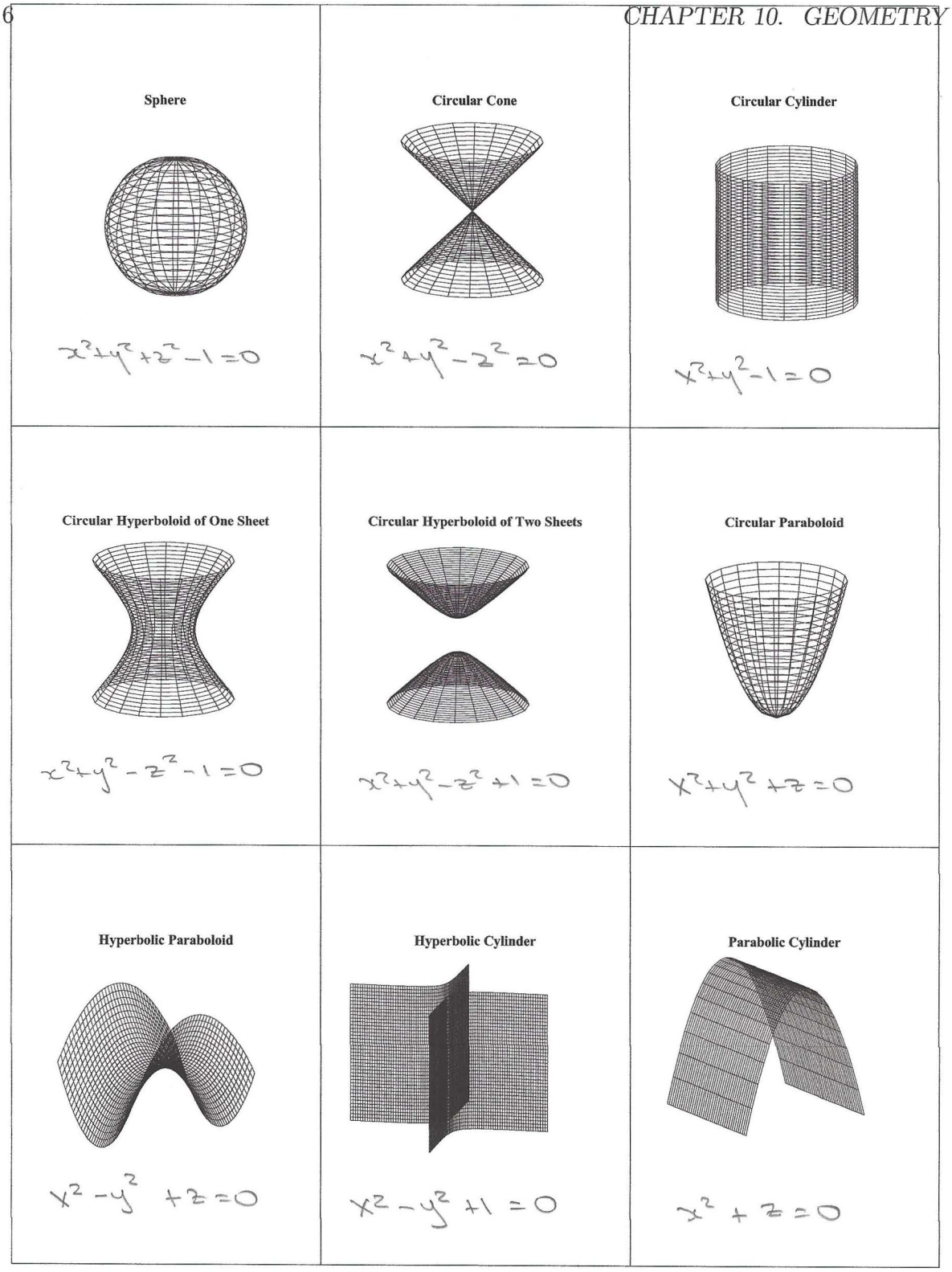


Figure 10.2: The nine real non-planar quadric surfaces.

10.3.1 Intercept to an arbitrary quadric

In our 4-vector notation, a transport step looks like

$$x_i = p_i + u_i \lambda \qquad (10.7)$$

where $x_i = (1, x, y, z)$ as before

$p_i = (1, x_0, y_0, z_0)$ by convention

and $u_i = (0, u, v, w)$ "

$(10.7) \rightarrow (10.3) \Rightarrow$

$$\sum_{ij=0}^{3} a_{ij} (p_i + u_i \lambda)(p_j + u_j \lambda) = 0$$

$$\Rightarrow \lambda^2 \sum_{ij=0}^{3} a_{ij} u_i u_j + \lambda \sum_{ij=0}^{3} a_{ij} p_i u_j$$

$$+ \lambda \sum_{ij} a_{ij} p_j u_i$$

$$+ \sum_{ij=0}^{3} a_{ij} p_i p_j = 0$$

Exploiting the symmetry of a_{ij}

$$\lambda^2 A(\vec{u}) + 2\lambda B(\vec{u}, \vec{p}) + C(\vec{p}) = 0 \qquad (10.8)$$

where $A(\vec{u}) = \sum_{ij} a_{ij} u_i u_j$; $B(\vec{u}, \vec{p}) = \sum_{ij} a_{ij} p_i u_j$

$$C(\vec{p}) = \sum_{ij} a_{ij} p_i p_j$$

Interpretation of the quadratic constants

If $C(\vec{p}) = 0$, the particle is on the surface

since (10.8) becomes

$$s\left(A(\vec{u})s + 2B(\vec{u},\vec{p})\right) = 0$$

$s = 0 \Rightarrow$ the particle is on the surface

When $B(\vec{u},\vec{p}) = \frac{1}{2}\vec{u} \cdot \underbrace{\nabla f(\vec{p})}_{\text{surface normal}}$

If $B(\vec{u},\vec{p}) = 0 \Rightarrow$ that $\vec{u}$ at $\vec{p}$ (on the surface) is tangent to the surface.

If $\underbrace{C(\vec{p}) = 0}_{\text{on surface}}$ and $\underbrace{B(\vec{u},\vec{p}) = 0}_{\vec{u} \text{ is tangent to the normal at } \vec{p}}$

$$s^2 A(\vec{u}) = 0$$

It can be shown that $A(\vec{u})$ is related to the curvature of the surface, along its tangent direction at the point on the surface

where $A(\vec{u})$ and $B(\vec{p},\vec{u})$ and $C(\vec{p})$ are all zero, the particle is travelling along a ruling of the surface.

inside or outside.

$$C(\vec{p} + \vec{u}\varepsilon) = C(p) + 2\varepsilon B(\vec{p},\vec{u})$$

($C(p) = 0$ ← on surface)

If $B > 0$ headed outside, by our convention

If $B < 0$ headed inside

This knowledge is incorporated into the general quadratic solver

```
function [hit,s] = quadric(A,B,C,inside)

% Calculates the first positive distance to an arbitrary quadric surface

% Inputs
%%%%%%%%
%  double
%     A:  Quadratic coefficient A (communicated to quadric)
%     B:  Quadratic coefficient B (communicated to quadric)
%     C:  Quadratic coefficient C (communicated to quadric)
%  logical
%     inside: == true   particle thinks it is inside
%           : == false  particle thinks it is outside

% Outputs
%%%%%%%%%
%  logical
%     hit: == true   particle would hit  the surface
%        : == false  particle would miss the surface
%
%  double
%     s: distance to the surface (if hit)

% Internal variables
%  double
%     Q: quadratic coefficient


Q = B.^2 - A.*C;

if (Q < 0)
    hit = false;
    s = nan;
    return
end

if (inside)           % inside the surface
   if (B <= 0)        % headed away from surface
      if (A > 0)      % but, surface curving up
         hit = true;  % always a hit in this case
         s = (sqrt(Q) - B)./A;
         return
      else            % surface curving away and headed in
         hit = false; % never a hit in this case
         s = inf;
         return
      end
   else               % headed toward the surface
      hit = true;     % always a hit in this case
      s = max(0, -C./(sqrt(Q) + B));
      return
   end
end

% Must be outside the surface

if (B >= 0)          % headed away
    if (A >= 0)      % surface curves away
        hit = false; % never a hit in this case
        s = inf;
        return
    else             % surface curves up
       hit = true;  % always a hit in this case
       s = -(sqrt(Q) + B)./A;
```

```
       return
    end
else                % headed toward the surface
    hit = true;     % always a hit in this case
    s = max(0, C./(sqrt(Q) - B));
    return
end

return
```

```
function [hit,s] = sphere(R,xs,x0,u,inside)

% Calculates the distance to a sphere of radius R centered at (xs,ys,zs)

% Inputs
%%%%%%%%
%  double
%     R :  Radius of the sphere (scalar)
%     xs:  coordinates of the center of the sphere (row vector)
%     x0:  coordinates of the particle (row vector)
%     u :  direction cosines of the particle (row vector)
%  logical
%     inside: == true   particle thinks it is inside
%           : == false  particle thinks it is outside
%

% Outputs
%%%%%%%%%
%  logical
%     hit: == true   particle would hit  the surface
%        : == false  particle would miss the surface
%
%  double
%     s: distance to the surface (if hit)
%
% Internal variables
%  double
%     xr: relative coordinates of the particle wrt the sphere (row vector)
%     A:  Quadratic coefficient A (communicated to quadric) (scalar)
%     B:  Quadratic coefficient B (communicated to quadric) (scalar)
%     C:  Quadratic coefficient C (communicated to quadric) (scalar)

% Compute relative coordinates

xr = x0 - xs;

% Compute the quadratic coefficients
A = 1; % i.e. assuming u**2 + v**2 + w**2 = 1
B = u*xr';
C = xr*xr' - R.^2;

% Call the quadratic solver
[hit,s] = quadric(A,B,C,inside);

return
```

```
function [hit,s] = ellipsoid(xc,a,x0,u,inside)

% Calculates the distance to an ellipsoid centered at (xs,ys,zs)
% with the mathematical form
% (a(x-x0))^2 + (b(y-y0))^2 + (c(z-z0))^2 = 1

% Inputs
%%%%%%%%
%  double
%     xc:  coordinates of the center of the ellipsoid (row vector)
%     a:   axis distortion factors (row vector)
%     x0:  coordinates of the particle (row vector)
%     u :  direction cosines of the particle (row vector)
%  logical
%     inside: == true   particle thinks it is inside
%           : == false  particle thinks it is outside
%

% Outputs
%%%%%%%%%
%  logical
%     hit: == true   particle would hit  the surface
%        : == false  particle would miss the surface
%
%  double
%     s: distance to the surface (if hit)
%
% Internal variables
%  double
%     xr: relative coordinates of the particle wrt the ellipsoid center (row vector)
%     A:  Quadratic coefficient A (communicated to quadric) (scalar)
%     B:  Quadratic coefficient B (communicated to quadric) (scalar)
%     C:  Quadratic coefficient C (communicated to quadric) (scalar)



xr = x0 - xc; % Compute relative coordinates
xr = xr.*a;   % Scale the relative coordinates
u  =  u.*a;   % Scale the direction vector

% Compute the quadratic coefficients
A = u*u'; % i.e. [a(1)*u]^2 + [a(2)*v]^2 + [a(3)*w]^2 = 1
B = u*xr';
C = xr*xr' - 1;

% Call the quadratic solver
[hit,s] = quadric(A,B,C,inside);

return
```

```
function [hit,s] = ccylz(R,x,u,inside)

% Calculates the distance to a cylinder of radius R concentric and aligned
% with the z-axis

% Inputs
%%%%%%%%
%  double
%     R :  Radius of the cylinder (scalar)
%     x :  coordinates of the particle (row vector)
%     u :  direction cosines of the particle (row vector)
%  logical
%     inside: == true   particle thinks it is inside
%           : == false  particle thinks it is outside
%

% Outputs
%%%%%%%%%
%  logical
%     hit: == true   particle would hit  the surface
%        : == false  particle would miss the surface
%
%  double
%     s: distance to the surface (if hit)
%
% Internal variables
%  double
%     A:  Quadratic coefficient A (communicated to quadric) (scalar)
%     B:  Quadratic coefficient B (communicated to quadric) (scalar)
%     C:  Quadratic coefficient C (communicated to quadric) (scalar)


% Compute the quadratic coefficients
A = u(1).^2 + u(2).^2;
B = u(1).*x(1)+ u(2).*x(2);
C = x(1).^2 + x(2).^2 - R.^2;

% Call the quadratic solver
[hit,s] = quadric(A,B,C,inside);

return
```