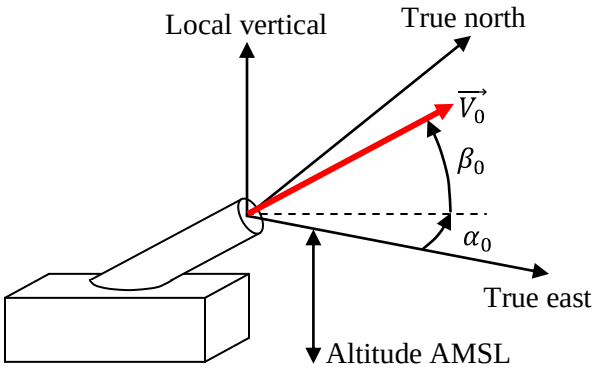
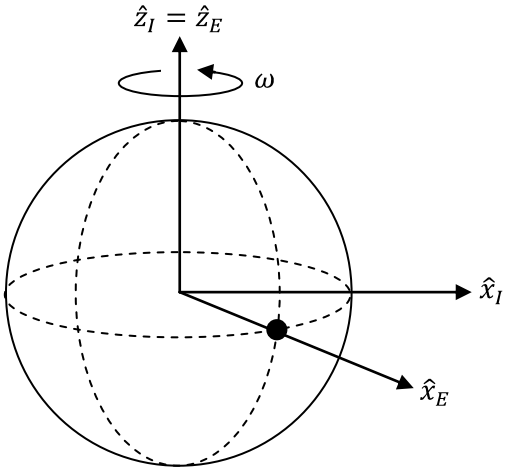
Exterior ballistics of a supersonic sphere

In this paper, I will look at the trajectory of a sphere launched at high speed from somewhere near the Earth's surface. I will assume that the sphere does not spin during flight. The situation in the vicinity of the gun is shown in the following figure.

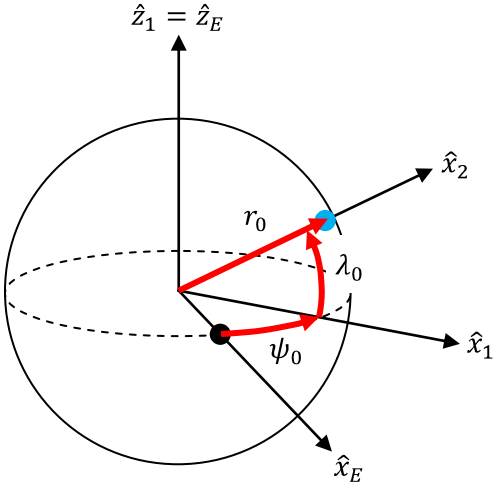


The red arrow shows the velocity of the cannonball as it leaves the muzzle. Its speed is V_0. Its launch direction is specified by two spherical angles. The initial track is angle α_0 north of true east and angle β_0 above the local horizontal plane. It will also be necessary to know the altitude of the muzzle. I will reference all altitudes to mean sea level.



I am going to introduce a couple of Earth-centered frames of reference. The $\hat{I}$ frame of reference shown in the figure at the left is an inertial frame of reference, assumed to be fixed with respect to the faraway stars. The $\hat{E}$ frame of reference is fixed to the Earth. Its $\hat{x}_E$-axis pierces the Earth's surface at the Equator at the longitude of Greenwich, which point of intersection is marked by the black dot. With respect to the $\hat{I}$ frame of reference, the Earth spins towards the east around the $\hat{z}_I = \hat{z}_E$-axis with angular velocity ω. Both of these frames of reference have $\hat{y}$-axes but I have not drawn them in the figure.

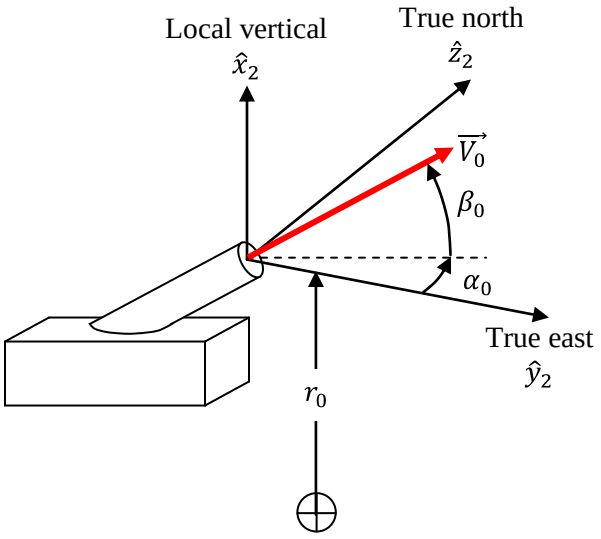
I am going to locate the muzzle of the gun in the $\hat{E}$ frame of reference using spherical co-ordinates. The muzzle is at longitude ψ_0, latitude λ_0 and radial distance r_0 from the center of the Earth. I will give effect to the three variables in the same order as I have just mentioned them. As a convention, I will assume that longitude east of Greenwich is positive and latitude north of the equator is positive.



The blue dot in the figure is the location of the muzzle. The sequence of transformations is this. The $\hat{1}$ frame of reference is derived from the $\hat{E}$ frame by a positive rotation by angle ψ_0 around the $\hat{z}_1 = \hat{z}_E$-axis. The $\hat{2}$ frame of reference is derived from the $\hat{1}$ frame by a negative rotation by angle λ_0 around the $\hat{y}_2 = \hat{y}_1$-axis. The muzzle is located a distance r_0 from the origin along the $\hat{x}_2$-axis.

Because of the way the $\hat{2}$ frame of reference has been derived, it happens that the $\hat{x}_2$-axis points directly upwards from the center of the muzzle. The $\hat{y}_2$-axis points due east and the $\hat{z}_2$-axis points due north. The following figure

shows (once again) the velocity of the sphere as it leaves the muzzle.



In the figure, the distance r_0 is the distance from the center of the Earth to the $\hat{y}_2$-$\hat{z}_2$ plane. I will be calculating this as the sum of: (i) the mean radius of the Earth, (ii) the altitude of the launch site above mean sea level and (iii) the height of the gun from its base to the center of the muzzle. One of the things that we need to do in preparation for the analysis below is to resolve the initial velocity of the sphere into its three components in this $\hat{2}$ frame of reference. Here, and below, Cartesian co-ordinates are listed in (x, y, z) order.

$$\overrightarrow{V_0}_2 = V_0 \begin{bmatrix} \sin\beta_0 \\ \cos\beta_0 \cos\alpha_0 \\ \cos\beta_0 \sin\alpha_0 \end{bmatrix} \quad (1)$$

The instantaneous location of the muzzle as seen in the inertial frame of reference $\hat{I}$

We can write down by inspection the location of the muzzle in the $\hat{2}$ frame of reference. It is:

$$\vec{M}_2 = \begin{bmatrix} r_0 \\ 0 \\ 0 \end{bmatrix} \quad (2)$$

It is not that difficult to express the location of the muzzle in the inertial frame of reference. It is a matter of multiplying standard rotation matrices in the correct order. Let's go through the rotations one-by-one. In the $\hat{1}$ frame of reference, the location of the muzzle is:

$$\vec{M}_1 = \begin{bmatrix} \cos\lambda_0 & 0 & -\sin\lambda_0 \\ 0 & 1 & 0 \\ \sin\lambda_0 & 0 & \cos\lambda_0 \end{bmatrix} \begin{bmatrix} r_0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} r_0\cos\lambda_0 \\ 0 \\ r_0\sin\lambda_0 \end{bmatrix} \quad (3)$$

In the equatorial Earth-centered Earth-fixed $\hat{E}$ frame of reference, it is:

$$\vec{M}_E = \begin{bmatrix} \cos\psi_0 & -\sin\psi_0 & 0 \\ \sin\psi_0 & \cos\psi_0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_0\cos\lambda_0 \\ 0 \\ r_0\sin\lambda_0 \end{bmatrix} = \begin{bmatrix} r_0\cos\lambda_0\cos\psi_0 \\ r_0\cos\lambda_0\sin\psi_0 \\ r_0\sin\lambda_0 \end{bmatrix} \quad (4)$$

Lastly, in the $\hat{I}$ frame of reference, starting at some arbitrary time $t = 0$ when the $\hat{E}$ and $\hat{I}$ frames are coincident, the location of the muzzle is:

$$\begin{aligned} \vec{M}_I &= \begin{bmatrix} \cos\omega t & -\sin\omega t & 0 \\ \sin\omega t & \cos\omega t & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_0\cos\lambda_0\cos\psi_0 \\ r_0\cos\lambda_0\sin\psi_0 \\ r_0\sin\lambda_0 \end{bmatrix} \\ &= r_0 \begin{bmatrix} \cos\lambda_0\cos(\psi_0+\omega t) \\ \cos\lambda_0\sin(\psi_0+\omega t) \\ \sin\lambda_0 \end{bmatrix} \end{aligned} \quad (5)$$

The instantaneous location of the sphere as seen in the inertial frame of reference $\hat{I}$

In the previous section, I used a sequence of rotation matrices to transform the location of the gun's muzzle, which we knew in the $\hat{2}$ frame of reference, into the inertial frame of reference. The procedure I used is certainly not restricted to the muzzle. It is very general. Let's introduce another Earth-centered frame of reference, $\hat{3}$, whose $\hat{x}_3$-axis always passes through the center of the sphere. The $\hat{3}$ frame of reference is similar to the $\hat{2}$-frame, so I will use the symbols ψ and λ, respectively, for the sphere's instantaneous longitude and latitude, instead of ψ_0 and λ_0, which will be reserved for the muzzle. In the $\hat{3}$ frame of reference, the sphere's location is specified simply by its location along the $\hat{x}_3$-axis, say, distance r from the center of the Earth.

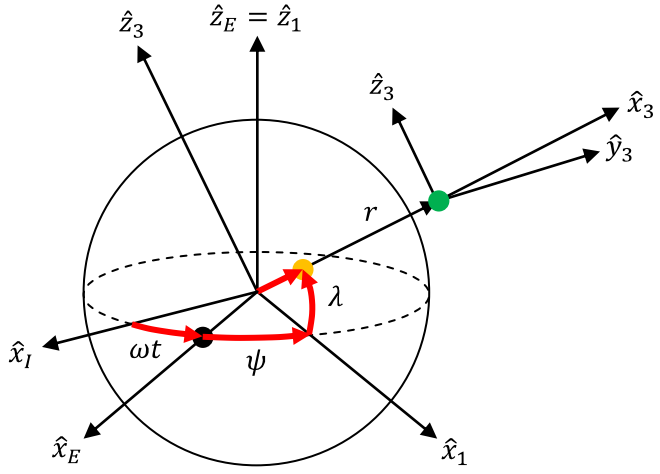
Let's use the symbol $\vec{S}$ for the vector from the center of the Earth to the center of the sphere at any particular moment during flight. The components of $\vec{S}$ can be written in the $\hat{3}$ frame of reference or, using successive rotation matrices for the latitude, longitude and Earth-rotation just like we did in steps (2) through (5), in the inertial frame of reference, as follows:

$$\vec{S}_3 = \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} \quad \rightarrow \quad \vec{S}_I = r \begin{bmatrix} \cos\lambda \cos(\psi + \omega t) \\ \cos\lambda \sin(\psi + \omega t) \\ \sin\lambda \end{bmatrix} \qquad (6)$$

In our simulations, we are going to base the simulation time ($t = 0$) on the instant when the sphere passes through the center of the muzzle. The initial conditions will therefore be:

$$\left.\begin{array}{l} r(t=0) = r_0 \\ \psi(t=0) = \psi_0 \\ \lambda(t=0) = \lambda_0 \end{array}\right\} \qquad (7)$$

The following figure shows how the radius r, longitude ψ and latitude λ correspond to the Cartesian $\hat{x}_3$, $\hat{y}_3$ and $\hat{z}_3$-axes in the $\hat{3}$-frame.



The green dot is the instantaneous location of the sphere. The orange dot is the projection of the sphere straight downwards onto the surface of the Earth. From the point of view of the sphere,

$\hat{x}_3$ points straight up (vertical)
$\hat{y}_3$ points due east
$\hat{z}_3$ points due north

From the figure, it can be seen that the longitude ψ is the main (but not only) determinant of the $\hat{y}_3$-axis and that the latitude λ is the main (but not only) determinant of the $\hat{z}_3$-axis. For this reason, I will list the location variables in the order (r, ψ, λ), which most-closely parallels the Cartesian co-ordinates (x, y, z).

The instantaneous location of the sphere as seen from the muzzle

It is sometimes useful to know the instantaneous location of the sphere as straight-line distances from the center of the muzzle. This is a vector, whose tail is the muzzle and whose head is the sphere. Like all vectors, its components can be found by subtraction, so long as the two vectors to be compared are expressed in the same frame of reference. Since the longitudes and latitudes of the muzzle and the sphere are defined with respect to the Earth-fixed $\hat{E}$ frame of reference, that frame could be the easiest one to use for subtraction. The vector from the muzzle to the sphere, in the $\hat{E}$-frame, is:

$$\begin{aligned}
\overrightarrow{MS}_E &= \vec{S}_E - \vec{M}_E \\
&= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\lambda & 0 & -\sin\lambda \\ 0 & 1 & 0 \\ \sin\lambda & 0 & \cos\psi \end{bmatrix} \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} r_0 \cos\lambda_0 \cos\psi_0 \\ r_0 \cos\lambda_0 \sin\psi_0 \\ r_0 \sin\lambda_0 \end{bmatrix} \\
&= \begin{bmatrix} r\cos\lambda\cos\psi \\ r\cos\lambda\sin\psi \\ r\sin\lambda \end{bmatrix} - \begin{bmatrix} r_0 \cos\lambda_0 \cos\psi_0 \\ r_0 \cos\lambda_0 \sin\psi_0 \\ r_0 \sin\lambda_0 \end{bmatrix} \\
&= \begin{bmatrix} r\cos\lambda\cos\psi - r_0 \cos\lambda_0 \cos\psi_0 \\ r\cos\lambda\sin\psi - r_0 \cos\lambda_0 \sin\psi_0 \\ r\sin\lambda - r_0 \sin\lambda_0 \end{bmatrix}
\end{aligned} \tag{8}$$

This vector will be more useful if we transform it into the muzzle's frame of reference, which is the $\hat{2}$-frame, as follows:

$$\begin{aligned}
\overrightarrow{MS}_2 &= \begin{bmatrix} \cos\lambda_0 & 0 & \sin\lambda_0 \\ 0 & 1 & 0 \\ -\sin\lambda_0 & 0 & \cos\lambda_0 \end{bmatrix} \begin{bmatrix} \cos\psi_0 & \sin\psi_0 & 0 \\ -\sin\psi_0 & \cos\psi_0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \overrightarrow{MS}_E \\
&= \begin{bmatrix} \cos\lambda_0 & 0 & \sin\lambda_0 \\ 0 & 1 & 0 \\ -\sin\lambda_0 & 0 & \cos\lambda_0 \end{bmatrix} \begin{bmatrix} \cos\psi_0 & \sin\psi_0 & 0 \\ -\sin\psi_0 & \cos\psi_0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r\cos\lambda\cos\psi - r_0 \cos\lambda_0 \cos\psi_0 \\ r\cos\lambda\sin\psi - r_0 \cos\lambda_0 \sin\psi_0 \\ r\sin\lambda - r_0 \sin\lambda_0 \end{bmatrix} \\
&= \begin{bmatrix} \cos\lambda_0 & 0 & \sin\lambda_0 \\ 0 & 1 & 0 \\ -\sin\lambda_0 & 0 & \cos\lambda_0 \end{bmatrix} \begin{bmatrix} r\cos\lambda\cos\psi\cos\psi_0 - r_0\cos\lambda_0 + r\cos\lambda\sin\psi\sin\psi_0 \\ -r\cos\lambda\cos\psi\sin\psi_0 + r\cos\lambda\sin\psi\cos\psi_0 \\ r\sin\lambda - r_0\sin\lambda_0 \end{bmatrix} \\
&= \begin{bmatrix} \cos\lambda_0 & 0 & \sin\lambda_0 \\ 0 & 1 & 0 \\ -\sin\lambda_0 & 0 & \cos\lambda_0 \end{bmatrix} \begin{bmatrix} r\cos\lambda\cos(\psi-\psi_0) - r_0\cos\lambda_0 \\ r\cos\lambda\sin(\psi-\psi_0) \\ r\sin\lambda - r_0\sin\lambda_0 \end{bmatrix} \\
&= \begin{bmatrix} -r_0 + r\cos\lambda\cos(\psi-\psi_0)\cos\lambda_0 + r\sin\lambda\sin\lambda_0 \\ r\cos\lambda\sin(\psi-\psi_0) \\ -r\cos\lambda\cos(\psi-\psi_0)\sin\lambda_0 + r\sin\lambda\cos\lambda_0 \end{bmatrix}
\end{aligned} \tag{9}$$

This is the vector from the muzzle to the sphere, as it would be seen by an observer standing next to the gun, with the three components being upwards (vertical), to the east and to the north, respectively. This vector is easy to imagine for short-range, flat-Earth, trajectories. Once the sphere drops below the horizon on a long-range trajectory, though, the vector will not be quite so intuitive.

Setting up the initial conditions for speed

Let's think about the moment when the simulation starts, at time $t = 0$. Substituting the initial radius, longitude and latitude into Equation (9) shows that $\overrightarrow{MS_2}(t = 0) = 0$. That is to be expected; at the starting instant, the sphere is located at the center of the muzzle.

The initial velocity, or speed, of the sphere is a different matter. I have chosen to specify the sphere's initial velocity as a given speed in a given direction. A conversion is necessary to put this velocity into terms the numerical simulation can understand.

The numerical simulation is going to be based on the sphere's three spherical variables r, ψ and λ, and their time-derivatives. It is not going to be based on the rates-of-change of the Cartesian co-ordinates. Therefore, the initial linear speeds need to be converted into rates-of-change of radius, longitude and latitude.

Equation (9) is the instantaneous location of the sphere with respect to the muzzle, in Cartesian co-ordinates. If we take the time-derivative of Equation (9), we will get the instantaneous velocity of the sphere with respect to the muzzle. We get:

$$\frac{d\overrightarrow{MS_2}}{dt} = \begin{bmatrix} \left\{ \begin{matrix} \dot{r}\cos\lambda\cos(\psi-\psi_0)\cos\lambda_0 + \dot{r}\sin\lambda\sin\lambda_0 + \cdots \\ \cdots - r\dot{\lambda}\sin\lambda\cos(\psi-\psi_0)\cos\lambda_0 + r\dot{\lambda}\cos\lambda\sin\lambda_0 + \cdots \\ \cdots - r\dot{\psi}\cos\lambda\sin(\psi-\psi_0)\cos\lambda_0 \end{matrix} \right\} \\ \left\{ \begin{matrix} \dot{r}\cos\lambda\sin(\psi-\psi_0) + \cdots \\ \cdots - r\dot{\lambda}\sin\lambda\sin(\psi-\psi_0) + \cdots \\ \cdots + r\dot{\psi}\cos\lambda\cos(\psi-\psi_0) \end{matrix} \right\} \\ \left\{ \begin{matrix} -\dot{r}\cos\lambda\cos(\psi-\psi_0)\sin\lambda_0 + \dot{r}\sin\lambda\cos\lambda_0 + \cdots \\ \cdots + r\dot{\lambda}\sin\lambda\cos(\psi-\psi_0)\sin\lambda_0 + r\dot{\lambda}\cos\lambda\cos\lambda_0 + \cdots \\ \cdots + r\dot{\psi}\cos\lambda\sin(\psi-\psi_0)\sin\lambda_0 \end{matrix} \right\} \end{bmatrix} \quad (10)$$

This is the relative speed at any time during flight. We only need to evaluate it at the starting instant, $t = 0$, when $r = r_0$, $\psi = \psi_0$ and $\lambda = \lambda_0$. At the starting instant, Equation (10) is greatly simplified, to:

$$\left.\frac{d\overrightarrow{MS_2}}{dt}\right|_0 = \begin{bmatrix} \dot{r}|_0\cos^2\lambda_0 + \dot{r}|_0\sin^2\lambda_0 - r_0\dot{\lambda}\big|_0\sin\lambda_0\cos\lambda_0 + r_0\dot{\lambda}\big|_0\cos\lambda_0\sin\lambda_0 \\ r_0\dot{\psi}\big|_0\cos\lambda_0 \\ -\dot{r}|_0\cos\lambda_0\sin\lambda_0 + \dot{r}|_0\sin\lambda_0\cos\lambda_0 + r_0\dot{\lambda}\big|_0\sin^2\lambda_0 + r_0\dot{\lambda}\big|_0\cos^2\lambda_0 \end{bmatrix}$$

$$= \begin{bmatrix} \dot{r}|_0 \\ r_0\dot{\psi}\big|_0\cos\lambda_0 \\ r_0\dot{\lambda}\big|_0 \end{bmatrix} \quad (11)$$

In Equation (11), $\dot{r}|_0$, $\dot{\psi}\big|_0$ and $\dot{\lambda}\big|_0$ are the initial rates-of-change of the sphere's radius, longitude and latitude, respectively. Now, we already have a second expression for this velocity. It is Equation (1), in which the sphere's initial velocity is expressed in terms of parameters of the gun. Setting the two formulations equal gives:

$$V_0 \begin{bmatrix} \sin\beta_0 \\ \cos\beta_0 \cos\alpha_0 \\ \cos\beta_0 \sin\alpha_0 \end{bmatrix} = \begin{bmatrix} \dot{r}\big|_0 \\ r_0 \dot{\psi}\big|_0 \cos\lambda_0 \\ r_0 \dot{\lambda}\big|_0 \end{bmatrix} \quad (12)$$

This constitutes three equations in the three unknowns $\dot{r}\big|_0$, $\dot{\psi}\big|_0$ and $\dot{\lambda}\big|_0$, which can be solved to give:

$$\left.\begin{aligned} \dot{r}\big|_0 &= V_0 \sin\beta_0 \\ \dot{\psi}\big|_0 &= \frac{V_0 \cos\beta_0 \cos\alpha_0}{r_0 \cos\lambda_0} \\ \dot{\lambda}\big|_0 &= \frac{V_0 \cos\beta_0 \sin\alpha_0}{r_0} \end{aligned}\right\} \quad (13)$$

The instantaneous speed and velocity of the sphere in the inertial frame of reference

Suppose the sphere is moving. The sphere's velocity is the first derivative with respect to time of its location. Let's start with $\vec{S}$ in the inertial frame of reference, as given by Equation (6). We can use the product rule to expand the derivative as:

$$\frac{d\vec{S}_I}{dt} = \dot{r} \begin{bmatrix} \cos\lambda \cos(\psi + \omega t) \\ \cos\lambda \sin(\psi + \omega t) \\ \sin\lambda \end{bmatrix} + r \begin{bmatrix} -\dot{\lambda} \sin\lambda \cos(\psi + \omega t) - (\dot{\psi} + \omega) \cos\lambda \sin(\psi + \omega t) \\ -\dot{\lambda} \sin\lambda \sin(\psi + \omega t) + (\dot{\psi} + \omega) \cos\lambda \cos(\psi + \omega t) \\ \dot{\lambda} \cos\lambda \end{bmatrix} \quad (14)$$

This is the velocity of the sphere, expressed in the inertial frame of reference. The speed is the square root of the sum of the squares of the three components. If V is the instantaneous speed, then:

$$V^2 = \left\{ \begin{array}{c} \left[\dot{r} \cos\lambda \cos(\psi + \omega t) - r\dot{\lambda} \sin\lambda \cos(\psi + \omega t) - r(\dot{\psi} + \omega) \cos\lambda \sin(\psi + \omega t)\right]^2 + \cdots \\ \cdots + \left[\dot{r} \cos\lambda \sin(\psi + \omega t) - r\dot{\lambda} \sin\lambda \sin(\psi + \omega t) + r(\dot{\psi} + \omega) \cos\lambda \cos(\psi + \omega t)\right]^2 + \cdots \\ \cdots + \left[\dot{r} \sin\lambda + r\dot{\lambda} \cos\lambda\right]^2 \end{array} \right\} \quad (15)$$

I have shown the algebra to reduce this expression in Appendix "A". The speed simplifies to:

$$V = \sqrt{\dot{r}^2 + r^2\dot{\lambda}^2 + r^2(\dot{\psi} + \omega)^2 \cos^2\lambda} \quad (16)$$

The instantaneous speed and velocity of the sphere with respect to the undisturbed air

There is nothing that requires that we calculate the speed of the sphere in the inertial frame of reference. We can calculate the speed in any frame of reference we wish. One of the things we are going to want to know is the velocity of the sphere with respect to the undisturbed air. The aerodynamic drag will be directly opposed to the sphere's velocity with respect to the air.

On a calm day, the speed of the air is zero, but only in some of the various frames of reference. It is zero in the muzzle's frame of reference (the $\hat{2}$-frame) and in the Earth-fixed frame of reference (the $\hat{E}$-frame). It is not zero in the sphere's frame of reference (the $\hat{3}$-frame) or in the inertial frame of reference (the $\hat{I}$-frame).

In Equation (9) above, we expressed the instantaneous location of the sphere in the muzzle's frame of reference. Since the air is stationary in that frame of reference, taking the derivative of that location vector will give the velocity of the sphere with respect to the air. But, we already took that derivative, in Equation (10), for the purpose of converting the initial conditions. If we take the sum of the squares of the three components, we will have the square of the sphere's speed with respect to the air.

$$V_{rel,2}^2 = \left\{ \begin{array}{c} \left[\begin{array}{c} \dot{r}\cos\lambda\cos(\psi-\psi_0)\cos\lambda_0 + \dot{r}\sin\lambda\sin\lambda_0 + \cdots \\ \cdots - r\dot{\lambda}\sin\lambda\cos(\psi-\psi_0)\cos\lambda_0 + r\dot{\lambda}\cos\lambda\sin\lambda_0 + \cdots \\ \cdots - r\dot{\psi}\cos\lambda\sin(\psi-\psi_0)\cos\lambda_0 \end{array} \right]^2 + \cdots \\ \cdots + \left[\begin{array}{c} \dot{r}\cos\lambda\sin(\psi-\psi_0) + \cdots \\ \cdots - r\dot{\lambda}\sin\lambda\sin(\psi-\psi_0) + \cdots \\ \cdots + r\dot{\psi}\cos\lambda\cos(\psi-\psi_0) \end{array} \right]^2 + \cdots \\ \cdots + \left[\begin{array}{c} -\dot{r}\cos\lambda\cos(\psi-\psi_0)\sin\lambda_0 + \dot{r}\sin\lambda\cos\lambda_0 + \cdots \\ \cdots + r\dot{\lambda}\sin\lambda\cos(\psi-\psi_0)\sin\lambda_0 + r\dot{\lambda}\cos\lambda\cos\lambda_0 + \cdots \\ \cdots + r\dot{\psi}\cos\lambda\sin(\psi-\psi_0)\sin\lambda_0 \end{array} \right]^2 \end{array} \right\} \quad (17)$$

Quite a bit of algebra is needed to reduce this expression. I have relegated the details to Appendix "B". Here, the sphere's speed relative to the air simplifies to:

$$V_{rel,2} = \sqrt{\dot{r}^2 + r^2\dot{\lambda}^2 + r^2\dot{\psi}^2\cos^2\lambda} \quad (18)$$

The instantaneous speed and velocity of the sphere with respect to the undisturbed air, once more

Equation (18) for the speed of the sphere relative to the undisturbed air is similar to Equation (16) for the speed of the sphere relative to the fixed stars. It took a lot of algebra in Appendices "A" and "B" to arrive at these two results. That is puzzling. Since the expressions for the speed are so simple, one would expect there to be some other, simpler, way to reach the same conclusion. I am going to work through that other way in this section. Let's start with Equation (14). $d\vec{S}_I/dt$ is the velocity of the sphere, expressed in the inertial frame of reference. It shows how the rates-of-change $\dot{r}$, $\dot{\lambda}$ and $\dot{\psi}$ of the sphere's three location variables r, λ and ψ give rise to rates-of-change along the three Cartesian axes of the inertial frame of reference.

Let' s consider the undisturbed air in the immediate vicinity of the sphere. This air has the same three location variables r, λ and ψ as the sphere, but all three of their rates-of-change are zero. Although these rates-of-change are zero, the velocity of the undisturbed air in the inertial frame of reference is not zero. As seen from the faraway stars, the undisturbed air is carried around the Earth's rotational axis as the Earth rotates. If we set $\dot{r} = \dot{\lambda} = \dot{\psi} = 0$ in Equation (14), we will have the velocity of the undistribed air in the $\hat{I}$-frame. Let's call this velocity $d\vec{A}_I/dt$. We get:

$$\frac{d\vec{A}_I}{dt} = r\omega\cos\lambda \begin{bmatrix} -\sin(\psi+\omega t) \\ \cos(\psi+\omega t) \\ 0 \end{bmatrix} \quad (19)$$

To find the velocity of the sphere with respect to the undisturbed air, we can take the difference between their velocity vectors in any frame of reference where we have expressions for them. Since we now have expressions for both velocities in the inertial frame, we can do the subtraction right now. The relative speed is:

$$\begin{aligned}\vec{V}_{rel,I} &= \frac{d\vec{S}_I}{dt} - \frac{d\vec{A}_I}{dt} \\ &= \begin{bmatrix} \dot{r}\cos\lambda\cos(\psi+\omega t) - r\dot{\lambda}\sin\lambda\cos(\psi+\omega t) - r\dot{\psi}\cos\lambda\sin(\psi+\omega t) \\ \dot{r}\cos\lambda\sin(\psi+\omega t) - r\dot{\lambda}\sin\lambda\sin(\psi+\omega t) + r\dot{\psi}\cos\lambda\cos(\psi+\omega t) \\ \dot{r}\sin\lambda + r\dot{\lambda}\cos\lambda \end{bmatrix} \\ &= \begin{bmatrix} \left(\dot{r}\cos\lambda - r\dot{\lambda}\sin\lambda\right)\cos(\psi+\omega t) - r\dot{\psi}\cos\lambda\sin(\psi+\omega t) \\ \left(\dot{r}\cos\lambda - r\dot{\lambda}\sin\lambda\right)\sin(\psi+\omega t) + r\dot{\psi}\cos\lambda\cos(\psi+\omega t) \\ \dot{r}\sin\lambda + r\dot{\lambda}\cos\lambda \end{bmatrix} \end{aligned} \tag{20}$$

Finding the relative speed from this equation is straightforward. It is easy to see which terms will add constructively and destructively when the first two components are squared and added together. We can write down by inspection:

$$\begin{aligned} V_{rel,I}^2 &= \left(\dot{r}\cos\lambda - r\dot{\lambda}\sin\lambda\right)^2 + \left(r\dot{\psi}\cos\lambda\right)^2 + \left(\dot{r}\sin\lambda + r\dot{\lambda}\cos\lambda\right)^2 \\ &= \begin{bmatrix} \dot{r}^2\cos^2\lambda - 2\dot{r}r\dot{\lambda}\sin\lambda\cos\lambda + r^2\dot{\lambda}^2\sin^2\lambda + \cdots \\ \cdots + r^2\dot{\psi}^2\cos^2\lambda + \cdots \\ \cdots + \dot{r}^2\sin^2\lambda + 2\dot{r}r\dot{\lambda}\sin\lambda\cos\lambda + r^2\dot{\lambda}^2\cos^2\lambda \end{bmatrix} \\ &= \dot{r}^2 + r^2\dot{\lambda}^2 + r^2\dot{\psi}^2\cos^2\lambda \end{aligned} \tag{21}$$

.

This is the same as Equation (18).

The instantaneous acceleration of the sphere

So far, we have talked only about the location and velocity of the sphere. Let's move on to the acceleration. I will start with Equation (14), which is the instantaneous velocity of the sphere in the inertial frame of reference. Taking another time derivative gives the acceleration.

$$\frac{d^2\vec{S}_I}{dt^2} = \left\{ \begin{array}{c} \ddot{r}\begin{bmatrix}\cos\lambda\cos(\psi+\omega t)\\ \cos\lambda\sin(\psi+\omega t)\\ \sin\lambda\end{bmatrix} + 2\dot{r}\begin{bmatrix}-\dot{\lambda}\sin\lambda\cos(\psi+\omega t) - \left(\dot{\psi}+\omega\right)\cos\lambda\sin(\psi+\omega t)\\ -\dot{\lambda}\sin\lambda\sin(\psi+\omega t) + \left(\dot{\psi}+\omega\right)\cos\lambda\cos(\psi+\omega t)\\ \dot{\lambda}\cos\lambda\end{bmatrix} + \cdots \\ \cdots + r\begin{bmatrix}-\ddot{\lambda}\sin\lambda\cos(\psi+\omega t) - \dot{\lambda}^2\cos\lambda\cos(\psi+\omega t) + \dot{\lambda}\left(\dot{\psi}+\omega\right)\sin\lambda\sin(\psi+\omega t)\\ -\ddot{\lambda}\sin\lambda\sin(\psi+\omega t) - \dot{\lambda}^2\cos\lambda\sin(\psi+\omega t) - \dot{\lambda}\left(\dot{\psi}+\omega\right)\sin\lambda\cos(\psi+\omega t)\\ \ddot{\lambda}\cos\lambda - \dot{\lambda}^2\sin\lambda\end{bmatrix} + \cdots \\ \cdots + r\begin{bmatrix}-\left(\ddot{\psi}+\dot{\omega}\right)\cos\lambda\sin(\psi+\omega t) + \dot{\lambda}\left(\dot{\psi}+\omega\right)\sin\lambda\sin(\psi+\omega t) - \left(\dot{\psi}+\omega\right)^2\cos\lambda\cos(\psi+\omega t)\\ \left(\ddot{\psi}+\dot{\omega}\right)\cos\lambda\cos(\psi+\omega t) - \dot{\lambda}\left(\dot{\psi}+\omega\right)\sin\lambda\cos(\psi+\omega t) - \left(\dot{\psi}+\omega\right)^2\cos\lambda\sin(\psi+\omega t)\\ 0\end{bmatrix} \end{array} \right\} \tag{22}$$

In many problems, including this one, the Earth's rotational speed ω can be assumed to be constant. The derivative $\dot{\omega}$ is therefore zero, and two of the terms in the last row of Equation (22) vanish.

There is an advantage to our having written the acceleration of the sphere in an inertial frame of reference. Only in an inertial frame of reference, which is not accelerating with respect to the faraway stars, can Newton's Third Law be used in its simplest $F = ma$ form. In contrast, in any frame of reference which is itself accelerating, Newton's Third Law can only be used if additional fictional forces are added to counteract the acceleration of the frame.

If point $\vec{S}_I$ is the instantaneous location of the sphere (and if the sphere can be treated as a rigid body with point mass m), then Newton's Third Law can be written as:

$$m\frac{d^2\vec{S}_I}{dt^2} = Net\ force\ acting\ on\ mass\ m \quad (23)$$

I am going to treat the cannonball as a point mass. It will be subject to two forces. There will be a gravitational force acting towards the center of the Earth. And, there will be an aerodynamic drag force acting in the direction opposite to the sphere's velocity with respect to the undisturbed air. Since I have assumed that the cannonball does not spin while in flight, there will not be a Magnus-type aerodynamic force arising from the spin.

For the moment, I am not going to quantify these forces. I am going to assume that we know them. Furthermore, I am going to assume that we know them, and can resolve them into their components, in the inertial frame of reference $\hat{I}$. For the time being, I am going to assume that we can write the total net force acting on the sphere as follows:

$$\vec{F}_I = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} \quad (24)$$

Substituting this into Equation (23) gives:

$$\frac{1}{m}\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \left\{RHS_{(22)}\right\} \quad (25)$$

where the contents of the curly brackets are the contents of the curly brackets on the right-hand side of Equation (22). (It is not worth writing all of it down again.) Note that Equation (22) is linear in the second derivatives of all three location variables $\ddot{r}$, $\ddot{\lambda}$ and $\ddot{\psi}$. We can therefore define 12 functions $a_{11} = a_{12}(r, \lambda, \psi, \dot{r}, \dot{\lambda}, \dot{\psi}, t)$, $a_{12} = a_{12}(r, \lambda, \psi, \dot{r}, \dot{\lambda}, \dot{\psi}, t)$ and so on all the way up to $a_{34} = a_{34}(r, \lambda, \psi, \dot{r}, \dot{\lambda}, \dot{\psi}, t)$ in such a way that Equation (25) can be written as:

$$\frac{1}{m}\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} a_{11}\ddot{r} + a_{12}\ddot{\lambda} + a_{13}\ddot{\psi} + a_{14} \\ a_{21}\ddot{r} + a_{22}\ddot{\lambda} + a_{23}\ddot{\psi} + a_{24} \\ a_{31}\ddot{r} + a_{32}\ddot{\lambda} + a_{33}\ddot{\psi} + a_{34} \end{bmatrix} \quad (26)$$

The twelve coefficient functions depend on the three location variables r, λ and ψ, their derivatives and time t. Just so there is no confusion, I have shown the individual functions in Appendix "C". Equation (26) can be re-arranged and re-stated as:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} \ddot{r} \\ \ddot{\lambda} \\ \ddot{\psi} \end{bmatrix} = \frac{1}{m}\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} - \begin{bmatrix} a_{14} \\ a_{24} \\ a_{34} \end{bmatrix} \quad (27)$$

Equation (27) is well-suited for a numerical integration. When we arrive at the beginning of any particular time step, we will know the values of three location variables r, λ and ψ and their derivatives

$\dot{r}$, $\dot{\lambda}$ and $\dot{\psi}$. They would have been calculated at the end of the preceding time step, which is coincident in time with the start of the present time step. We will also know the current time t, of course. We can compute all twelve coefficients $a_{11}, a_{12}, a_{13} \cdots a_{34}$. Assuming that we also know the components of the net force at this time, then we can solve Equation (27) for the three second derivatives, $\ddot{r}$, $\ddot{\lambda}$ and $\ddot{\psi}$. If the time step has duration ΔT, and if ΔT is suitably short, then we can assume that the three second derivatives remain virtually constant through the time step. If so, then the values of the three first derivatives at the end of this time step can be found through a simple linear integration, thus:

$$\left.\begin{aligned} \dot{r}_{end} &= \dot{r}_{start} + \ddot{r}\Delta T \\ \dot{\lambda}_{end} &= \dot{\lambda}_{start} + \ddot{\lambda}\Delta T \\ \dot{\psi}_{end} &= \dot{\psi}_{start} + \ddot{\psi}\Delta T \end{aligned}\right\} \quad (28)$$

A second integration gives the values of the three location variables at the end of this time step:

$$\left.\begin{aligned} r_{end} &= r_{start} + \dot{r}_{start}\Delta T + \tfrac{1}{2}\ddot{r}\Delta T^2 \\ \lambda_{end} &= \lambda_{start} + \dot{\lambda}_{start}\Delta T + \tfrac{1}{2}\ddot{\lambda}\Delta T^2 \\ \psi_{end} &= \psi_{start} + \dot{\psi}_{start}\Delta T + \tfrac{1}{2}\ddot{\psi}\Delta T^2 \end{aligned}\right\} \quad (29)$$

Equations (28) and (29) produce the six values which are needed to begin the next time step. In these equations, the subscripts "$start$" and "end" simply identify the whether the particular value is the value at the beginning or end of the time step being processed.

The code which implements the numerical procedure, and which is listed in Appendix "D", does not invert the 3-by-3 coefficient matrix *per se*. Instead, I have solved the three equations using Eulerian substitution, as described in Appendix "C".

The force of gravity

The magnitude of the gravitational force is simple to write down. Since the sphere will not necessarily be close to the Earth's surface, I will use the Newton's more general expression for the force of gravity:

$$F_g = G\frac{M_E m}{r^2} \quad (30)$$

In this expression, G is the universal gravitational constant $G = 6.673 \times 10^{-11}\ Nm^2/kg^2$, M_E is the mass of the Earth $M_E = 5.97219 \times 10^{24}\ kg$, m is the mass of the cannonball and r is the distance from the center of the Earth to the center of the cannonball. This force always acts in the direction from the sphere to the center of the Earth.

This is a good time to talk about the Earth's radius. For many ballistic problems (satellites, for example), neither the Earth's radius nor its gravitational field can be considered to be constant. The Earth's oblate shape has a small, but cumulatively adverse, impact on ballistic predictions. The Earth's radius varies from 6,353 kilometers at the poles to 6,384 kilometers at the Equator. The mean radius, which is arguably most appropriate for mid-latitudes, is sometimes given as $R_E = 6{,}356{,}766$ meters. I am going to use this value as the radial distance to mean sea level. I will measure all local altitudes above mean sea level. If H is the instantaneous geometric altitude of the sphere, then Equation (30) can be written as:

$$F_g = G\frac{M_E m}{(R_E + H)^2} \quad (31)$$

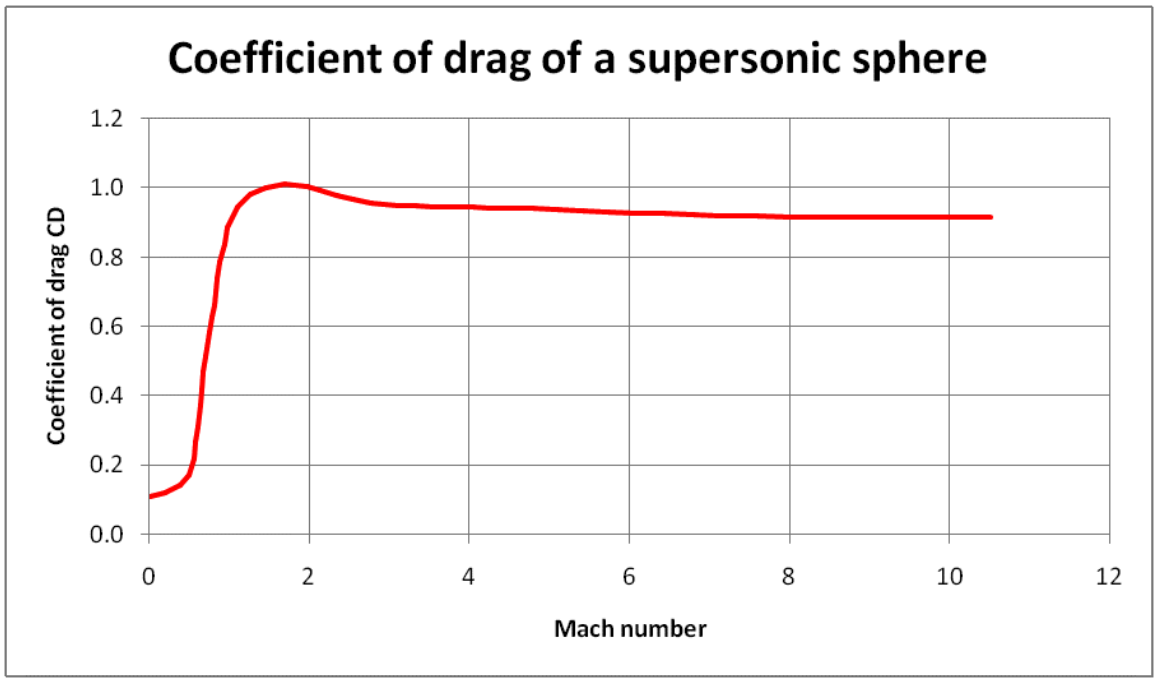
In the sphere's own frame of reference, the $\hat{S}$ frame, the direction of gravity is easily stated. It points directly down, in the direction of the negative $\hat{x}_3$-axis. We can use Equation (6) to transform it into the inertial frame of reference.

$$\vec{F}_{g,S} = \begin{bmatrix} -G\dfrac{M_E m}{(R_E+H)^2} \\ 0 \\ 0 \end{bmatrix} \quad \rightarrow \quad \vec{F}_{g,I} = -G\frac{M_E m}{(R_E+H)^2}\begin{bmatrix} \cos\lambda\cos(\psi+\omega t) \\ \cos\lambda\sin(\psi+\omega t) \\ \sin\lambda \end{bmatrix} \qquad (32)$$

At the beginning of any particular time step in the numerical integration, we will know the sphere's local angles λ_{start} and ψ_{start}. We will also know its then-altitude, H_{start} say, and can use Equation (32) to express the force of gravity in the inertial frame of reference.

The aerodynamic drag

I am going to base my calculations of aerodynamic drag on a study published by NASA in August 1993. The study was done by M.L. Spearman and D.O. Braswell and is titled *Aerodynamics of a sphere and an oblate spheroid for Mach numbers from 0.6 to 10.5 including some effects of test conditions*. They tested spheres of various sizes up to 12 inches in diameter in various wind tunnels. The principal result I am going to take over from their report is a graph which plots the coefficient of drag against a range of Mach numbers. I digitized their graph and obtained a set of $(C_D, Mach)$ co-ordinate pairs. The following graph was drawn using the digitized co-ordinate pairs.



If one knows the sphere's Mach number, the coefficient of drag is easily extracted from the graph. In the numerical simulation, I use linear interpolation between adjacent co-ordinates pairs to do that. Finding the Mach number is a little more of a challenge. I handled this task by coding the U.S. Standard Atmosphere in a way that calculates the speed of sound, and other state variables of the air, for any given geometric altitude in the U.S. Standard Atmosphere. The details of this process are described in a separate paper, titled *Formulae and code for the U.S. Standard Atmosphere (1976)*, the computer code from which I have taken for this application over without any change.

Suppose we introduce the following variables:

- H (as above) is the instantaneous altitude of the sphere above mean sea level, measured in meters,
- ρ_{air} is the density of the air in the vicinity of the sphere, measured in kilograms per cubic meter,
- c_{air} is the speed of sound in the vicinity of the sphere, measured in meters per second,
- ν_{air} is the kinematic viscosity of the air in the vicinity of the sphere, in meters squared per second,
- V_{rel} is the instantaneous speed of the sphere, relative to the local air, measured in meters per second,
- $Mach$ is the Mach number of the sphere's speed,
- C_D is the instantaneous coefficient of drag,
- R_{sphere} is the radius of the sphere, measured in meters,
- F_a is the instantaneous aerodynamic drag (force) acting on the sphere, measured in Newtons, and
- Re is the instantaneous Reynolds number.

The procedure is as follows. The only input value required to invoke the standard atmosphere is the sphere's geometric altitude H. The subroutine which models the atmosphere converts this geometric altitude into its gravity-corrected geopotential altitude, calculates its way up through the layers in the standard atmosphere, and calculates the temperature, pressure, density, dynamic and kinematic viscosities, and speed of sound for the given altitude. We only need three of these state variables, the local density ρ_{air}, the local speed of sound c_{air} and the local kinematic viscosity ν_{air}.

Next, we compute the sphere's Mach number. This is the sphere's speed with respect to the undisturbed air, divided by the local speed of sound. We will have to make sure that we use the right frame of reference to determine the sphere's relative speed V_{rel} but, once we have it, the Mach number is:

$$Mach = \frac{V_{rel}}{c_{air}} \quad (33)$$

With this Mach number, we can look up the coefficient of drag C_D from the graph above. The magnitude of the aerodynamic drag force is then calculated using the standard drag equation:

$$F_a = C_D \times ½\rho_{air} V_{rel}^2 \times \pi R_{sphere}^2 \quad (34)$$

The factor $½\rho_{air} V_{rel}^2$ is the core term in the drag (and lift) equations; it captures the force's dependence on density and speed. That factor is adjusted for the frontal area of the object, πR_{sphere}^2 in the case of our sphere, and by the coefficient of drag C_D.

As a reality check on what is going on, I will also calculate the Reynolds number, which will give us some idea about the nature of the flow. It should be consistent with supersonic speeds. The definition of the Reynolds number for an object is:

$$Re = \frac{V_{rel} \times 2R_{sphere}}{\nu_{air}} \quad (35)$$

where the factor $2R_{sphere}$ is the characteristic length of the object, in our case being the diameter of the sphere.

This drag force F_a will be directed in opposition to the sphere's velocity through the undisturbed air. Equation (20) above gives this relative velocity. Handily, this relative velocity is already expressed in the inertial frame of reference. The speed is given Equation (21). This is the speed which is substituted into Equation (33) to calculate the sphere's Mach number. With the Mach number at hand, the

coefficient of drag can be extracted from the graph and then the magnitude of the aerodynamic drag F_a calculated using Equation (34). The aerodynamic drag must now be broken down into its components. The components will have the same proportions as the velocity in Equation (20), but it will have the opposite direction.

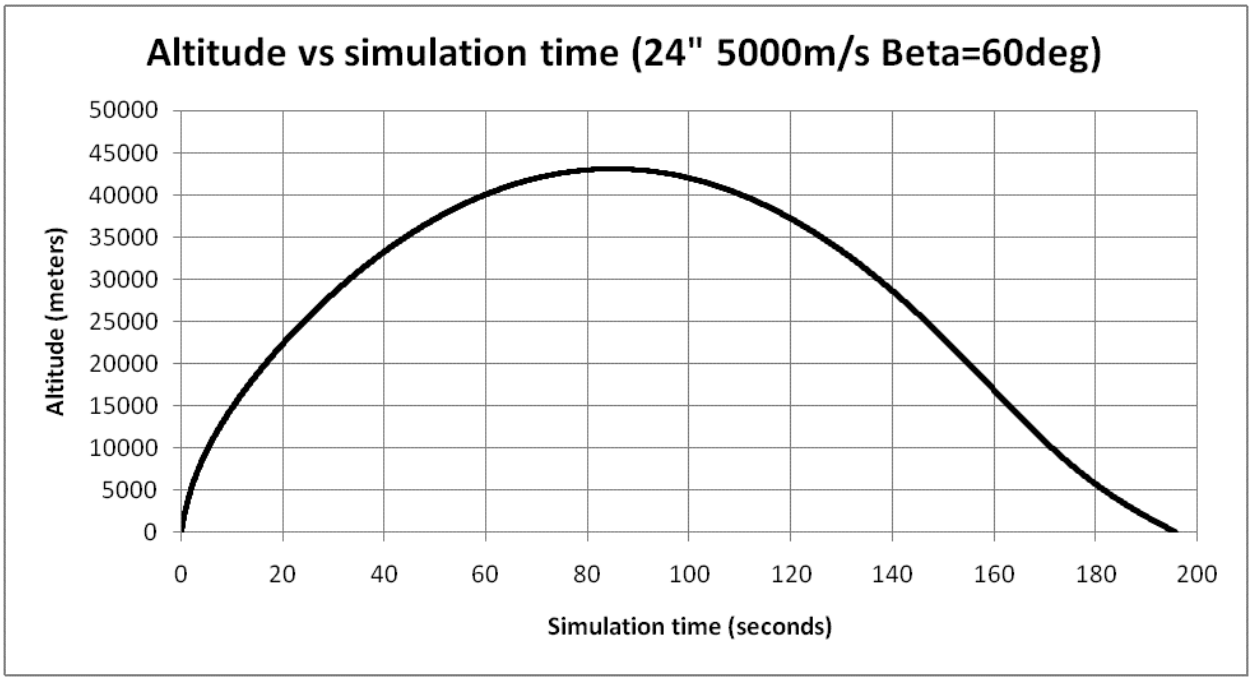
$$\vec{F}_{a,I} = -\frac{\vec{V}_{rel,I}}{V_{rel,I}} F_a \quad (36)$$

Some preliminary results

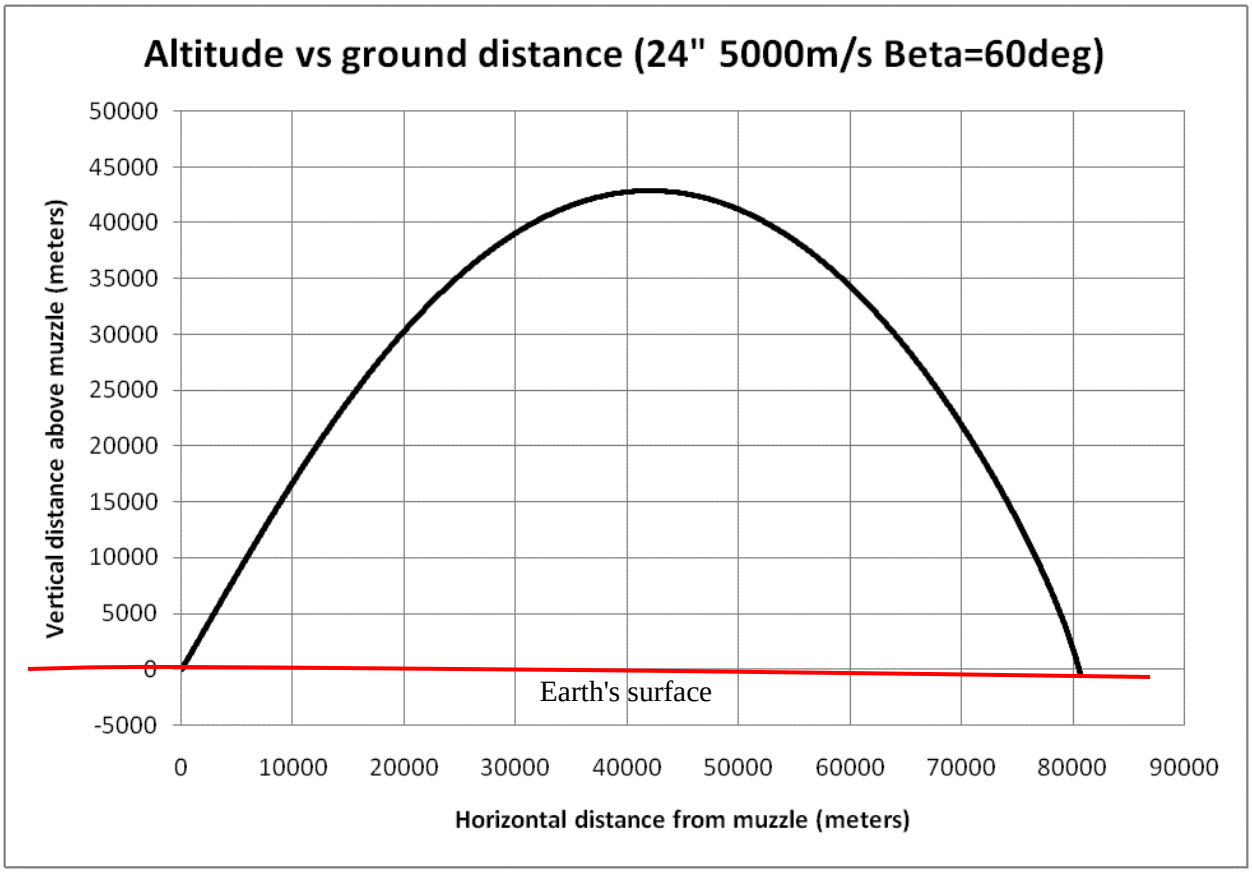
I have attached as Appendix "D" a listing of the Visual Basic code which implements the equations governing the exterior ballistics set out above. The parameters I used for this preliminary case are the following:

1. The sphere is 24 inches in diameter and cast from metal with a density of 7,900 kilograms per cubic meter.
2. The initial launch speed is 5,000 meters per second, at an elevation (angle β) of 60° in a direction (angle α) 8° north of due east.
3. The launch site is in the northern hemisphere at latitude 32.08275°. Since I will report the results in terms of distances along the ground, the longitude is arbitrary.
4. The base of the gun is 7.26 meters above mean sea level and the muzzle is ten meters above the gun's base.
5. For the purposes of ending the flight, I have assumed the sphere lands on a site where the altitude is 29 meters above mean sea level.
6. The duration of the time step for the numerical integration is one microsecond.

The time of flight is 195.74 seconds, or about three and one-third minutes. The following graph shows the altitude of the sphere above mean sea level with respect to the simulation time. The sphere reaches a maximum altitude of about 43 kilometers.

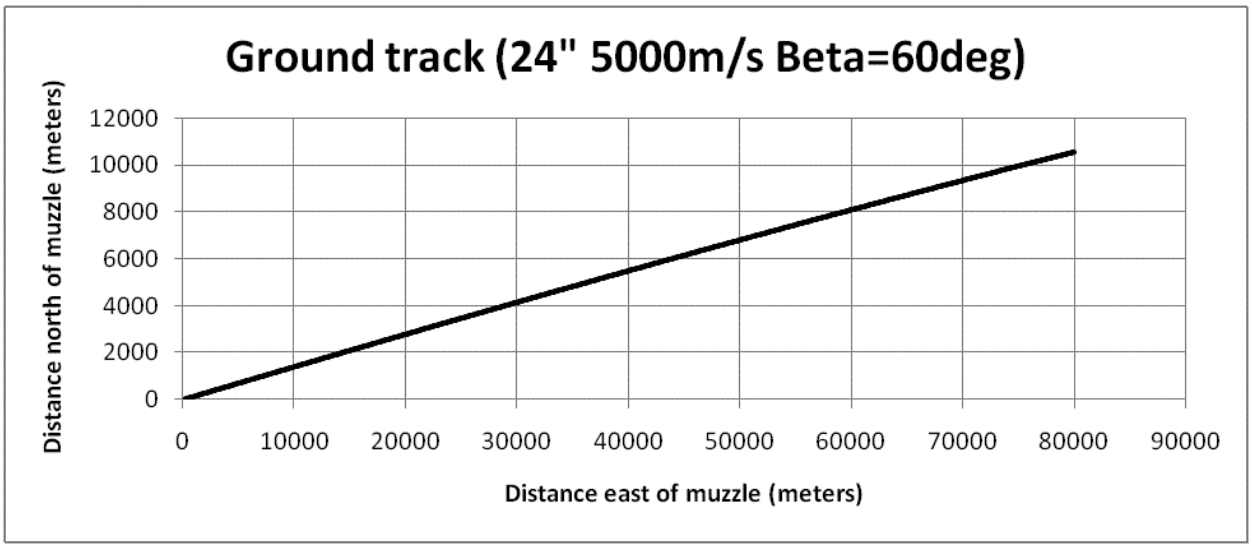


The following graph shows the trajectory of the sphere in the vertical plane. The distances plotted are stated in the $\hat{2}$ frame of reference (the muzzle's frame of reference), and represent the Cartesian co-ordinates of the sphere relative to the muzzle. The vertical axis of the graph is the distance of the sphere above the muzzle, which will increasingly diverge from the sphere's true altitude as the sphere moves over the surface of the Earth. The horizontal axis is the distance to the sphere's projection onto the horizontal plane passing through the muzzle. I have scaled the graph so that distances are the same along both axes. The sphere lands just over 80 kilometers from the gun.



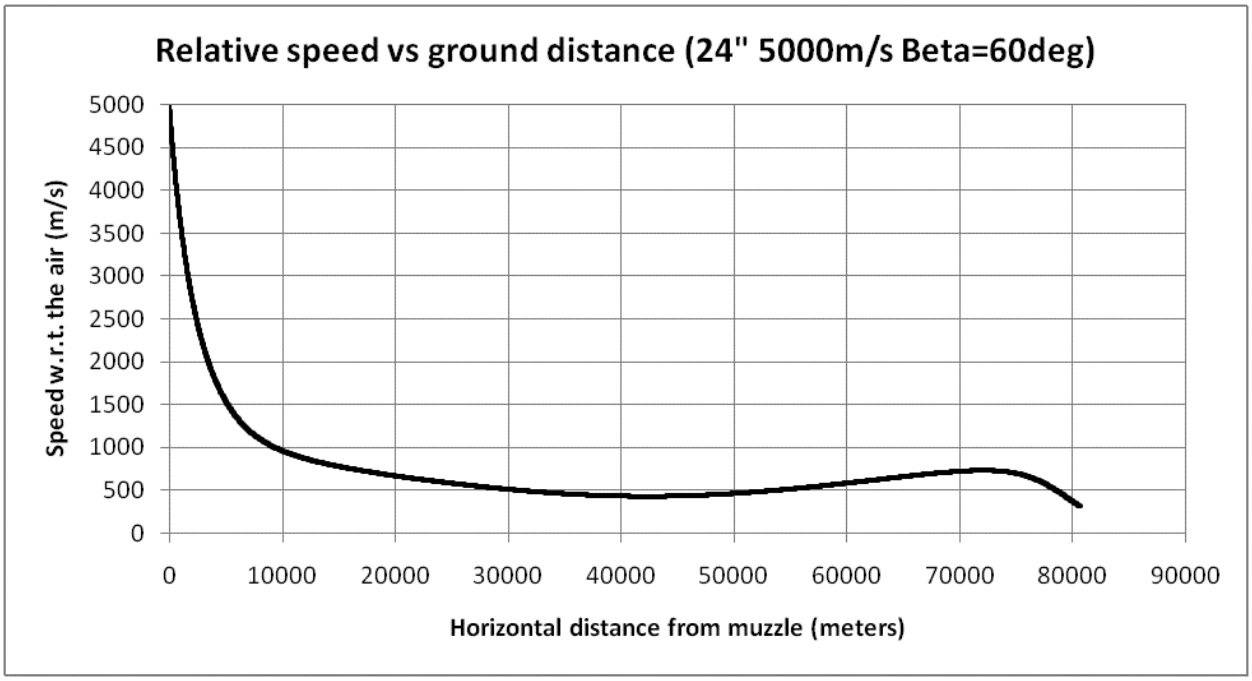
The red line I have superimposed on the graph represents the circular surface of the Earth. It is a cross - section of the Great Circle passing through the launch and landing sites. The landing site is far enough around the Earth that it is several hundred meters below the horizontal plane passing through the muzzle.

The following graph is the ground track.



The $(0, 0)$ co-ordinate on this graph is the muzzle, as seen from above. The vertical axis points due north; the horizontal axis points due east. However, note that the two axes are not scaled to comparable lengths. The sphere travels about 11 kilometers north and about 80 kilometers east.

The following graph shows the speed of the sphere with respect to the air (the "relative" speed) as a function of the horizontal distance from the gun.



The initial speed of the sphere (5,000 meters per second is an extraordinarily high speed, about Mach 14.7) is quickly reduced by aerodynamic drag. The sphere has slowed to 1,000 meters per second (Mach 3.39) when it is 11.13 seconds into flight, at which time it is at an altitude of 15.8 kilometers and 9.4 kilometers downrange.

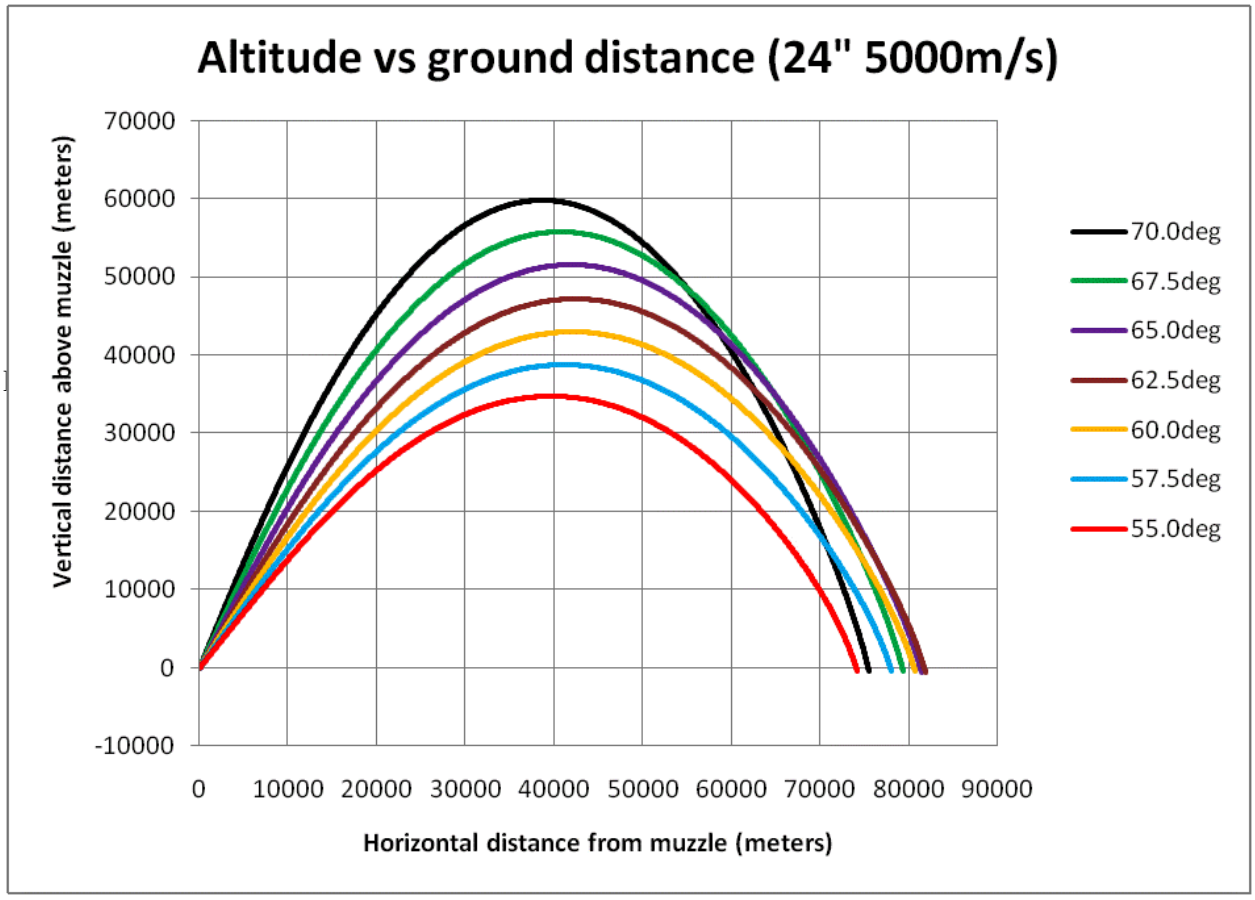
At apogee, say, when the sphere is 40 kilometers downrange, its relative speed has decreased to 322 meters per second, or Mach 1.34.

As the sphere then begins to descend, it picks up speed as its potential energy is converted into kinetic energy. But, once it begins to enter the thicker, lower, atmosphere, the increasing drag bleeds off energy and reduces the speed. The sphere lands with a relative speed of 340 meters per second, or Mach 0.92.

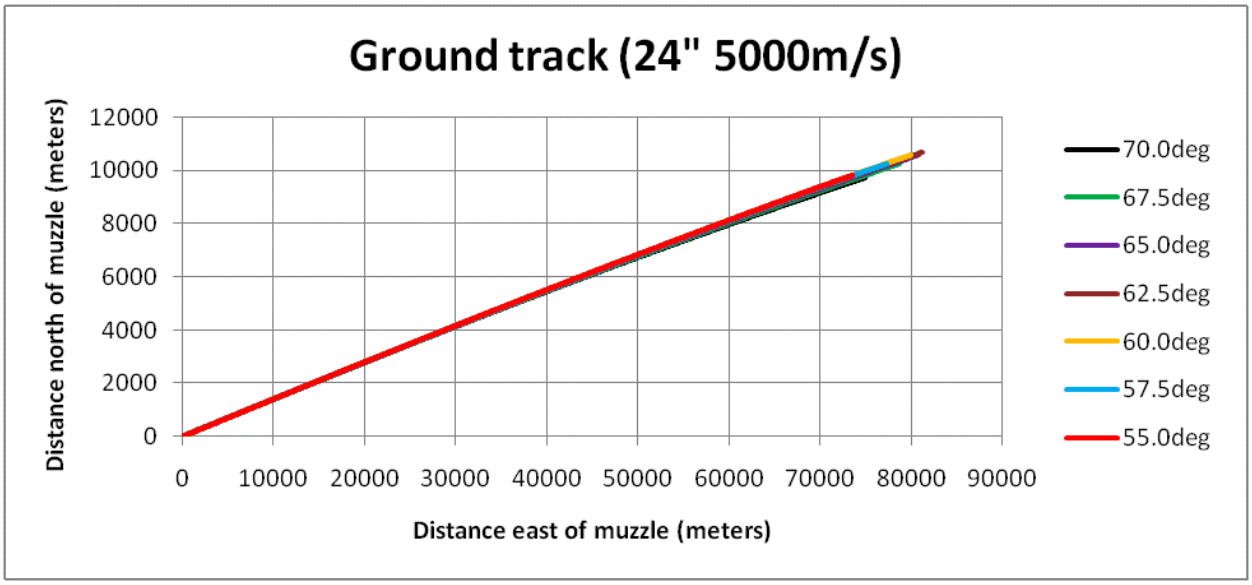
The landing speed is a fraction $340/5000 = 0.068$ of the launch speed, so the landing kinetic energy is a fraction $0.068^2 = 0.0046$ of the launch kinetic energy. Clearly, a supersonic sphere is a poor way to transmit kinetic energy from one place to another.

The effect of the gun's elevation angle β

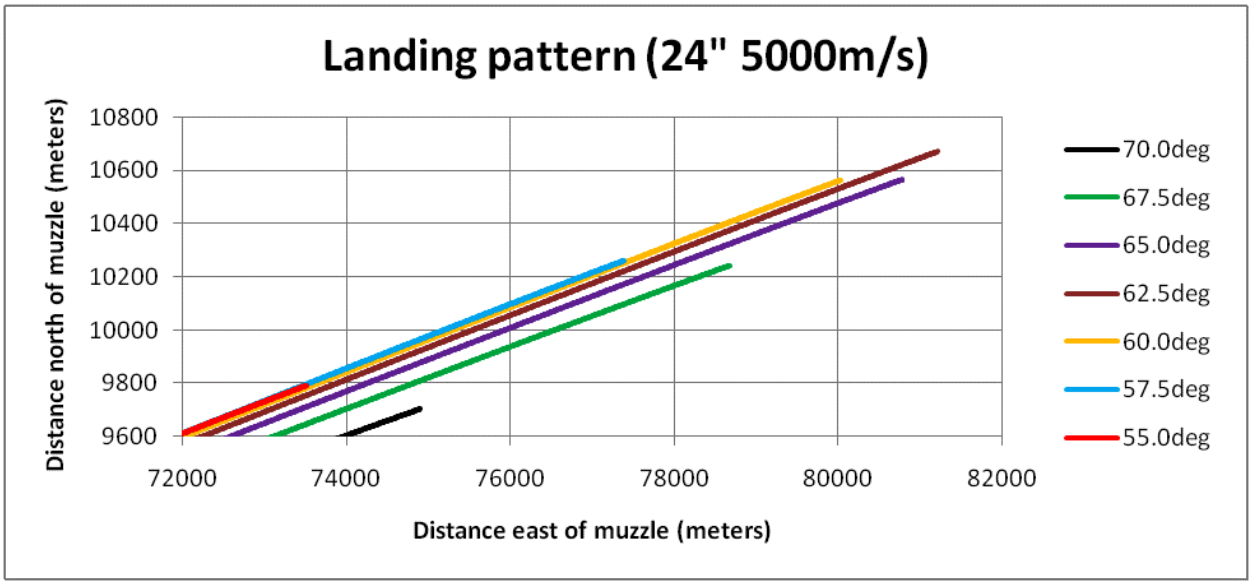
In the preliminary case described in the previous section, the barrel of the gun was elevated to an angle 60° above the horizontal plane. The following graph shows the effect of changing the elevation angle, while leaving all other parameters unchanged. This is a graph of the Cartesian location of the sphere with respect to the muzzle. The yellow curve is the preliminary case graphed above; the other curves show elevation angles from 55° to 70°. It can be seen that the maximum range is achieved at an angle of 62.5°.



The following graph shows the ground tracks which correspond to these flights.

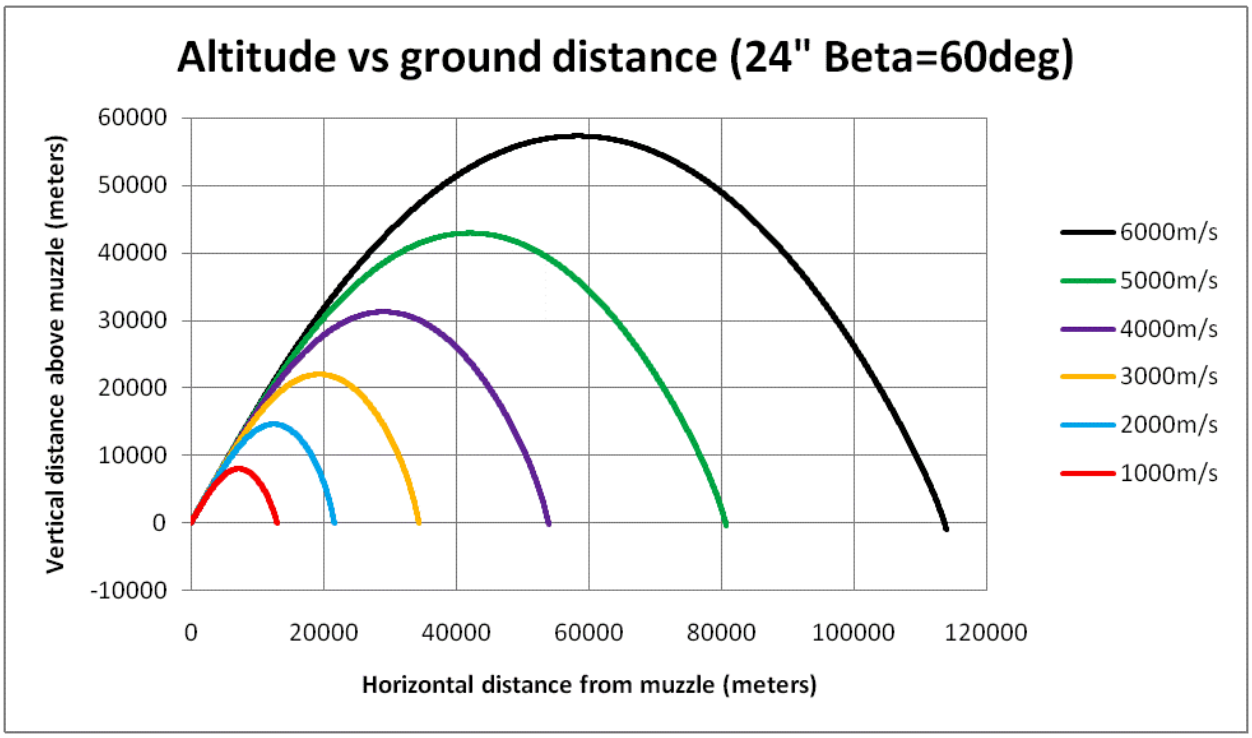


Since the ground tracks are similar, the landing pattern is not very clear. The following graph is a close-up of the landing area. It seems that all of these flights land in a strip about 10 kilometers wide in the east-west direction and about one kilometer high in the north-south direction.



The effect of the launch speed

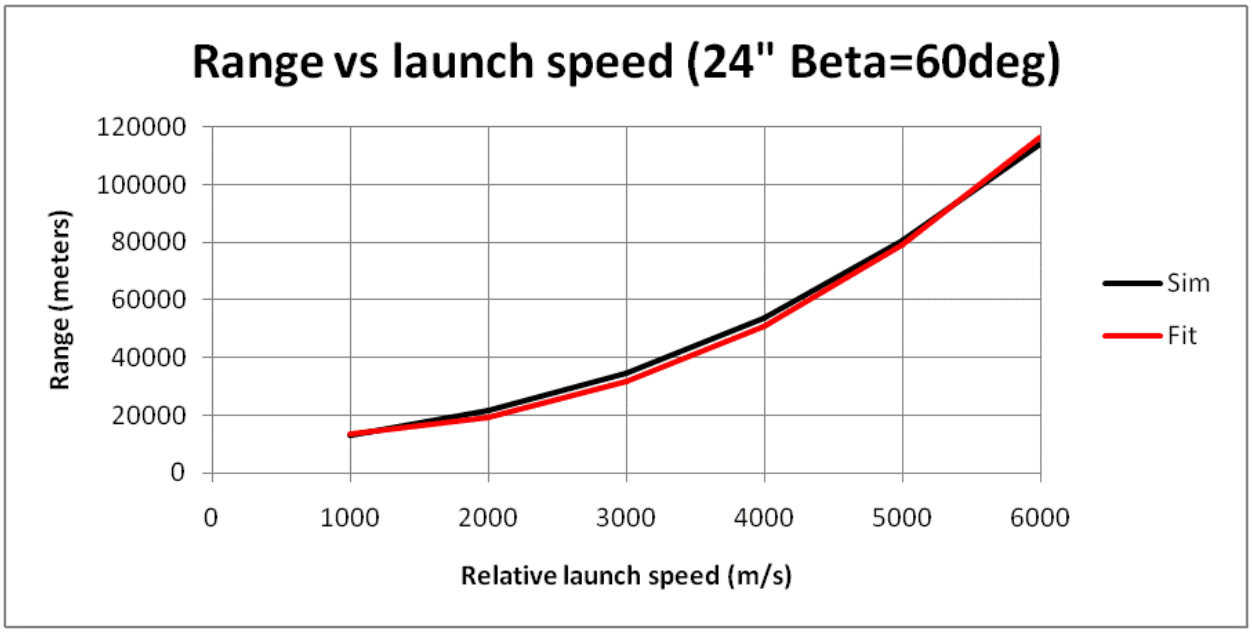
All of the cases so far assume the 24" sphere is launched at 5,000 meters per second. The following graph shows the effect of different launch speeds, from 1,000 m/s to 6,000 m/s. In all of these cases, the launch elevation is kept at 60°.



As one would expect, a greater launch speed gives a greater range. The following graph shows the relationship. The black curve is the simulation results. The red curve is the equation:

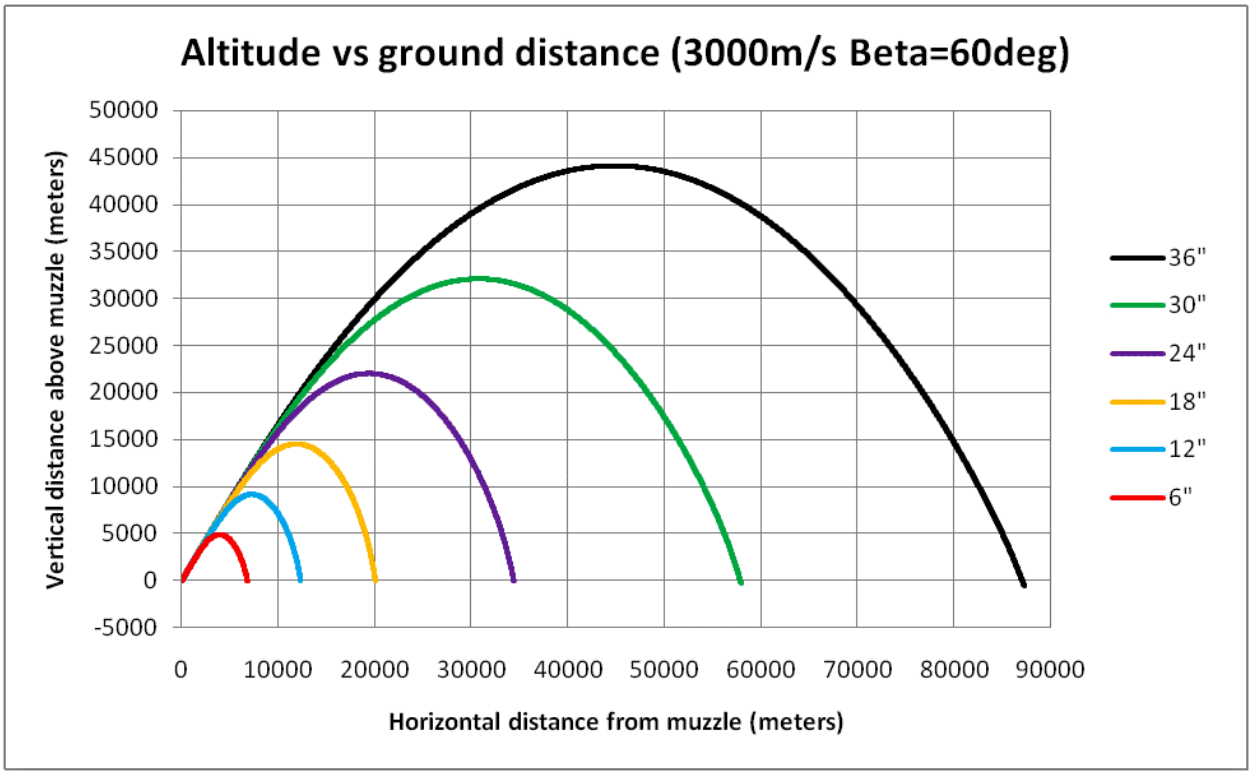
$$Range = 12{,}000 + (0.000075 \times Speed^{2.42})$$

The equation shows that the range increases aggressively with speed, at more than the square. This arises because the sphere spends more time at very high altitudes, where the aerodynamic drag is much lower.



The effect of the sphere's diameter

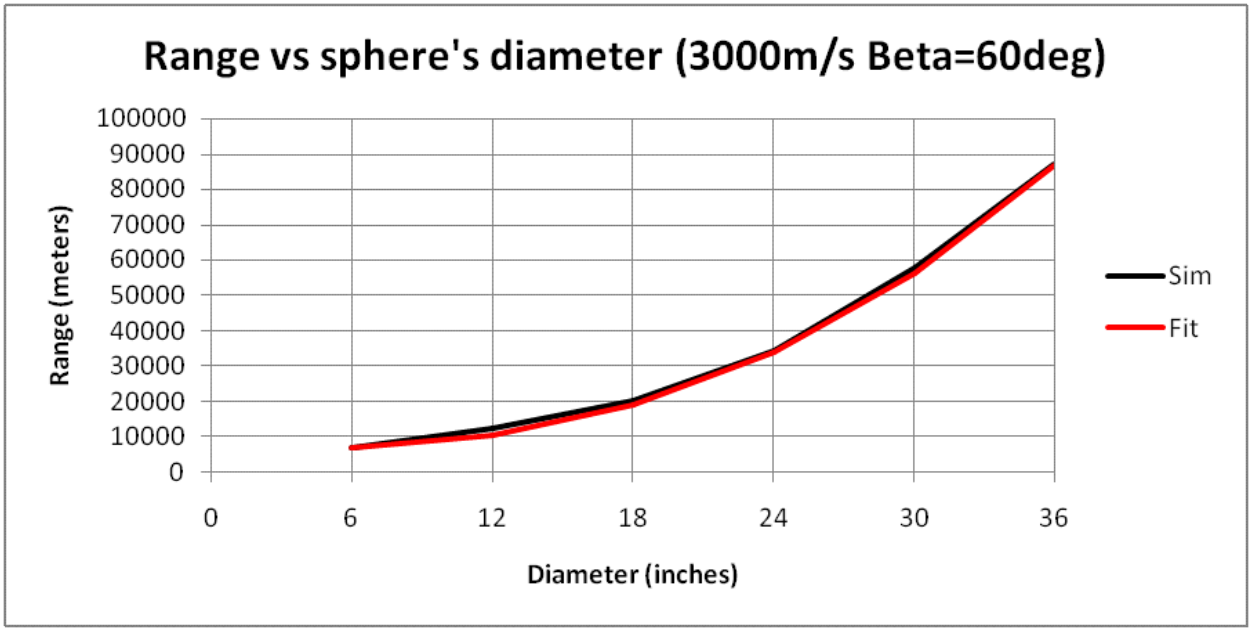
All of the cases so far assume the sphere has a diameter of 24 inches. The following graph shows the effect of changing the diameter, in the range from 6 inches to 36 inches. All of these runs had a launch speed of 3,000 meters per second and a launch elevation of 60°.



Increasing the diameter greatly increases the range. The following graph shows the relationship. The black curve connects the simulation results. The red curve is the equation:

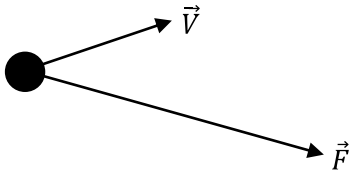
$$Range = 6000 + (7 \times Diameter^{2.61})$$

The equation shows that the range increases aggressively with the diameter, closer to the cube of the diameter than to the square.



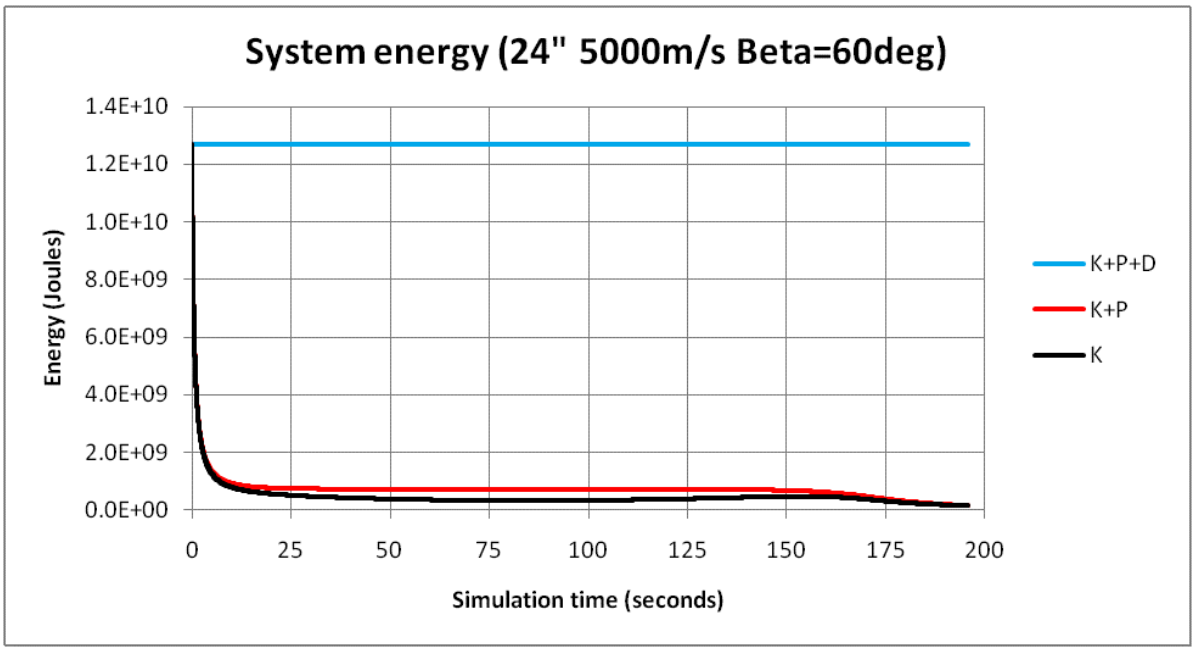
The energy balance

As a check on the consistency of the physical model, I examined the energy of the system. I considered three types of energy: (i) the kinetic energy of the sphere, (ii) the potential energy of its distance from the center of the Earth and (iii) the energy expended in overcoming the aerodynamic drag. I carried out all energy calculations in the inertial frame of reference. For example, if the launch speed (relative to the undisturbed air) is 5,000 meters per second, the speed in the inertial frame of reference is 5,205.7 meters per second, the increase being due to the Earth's rotation in the direction of travel.



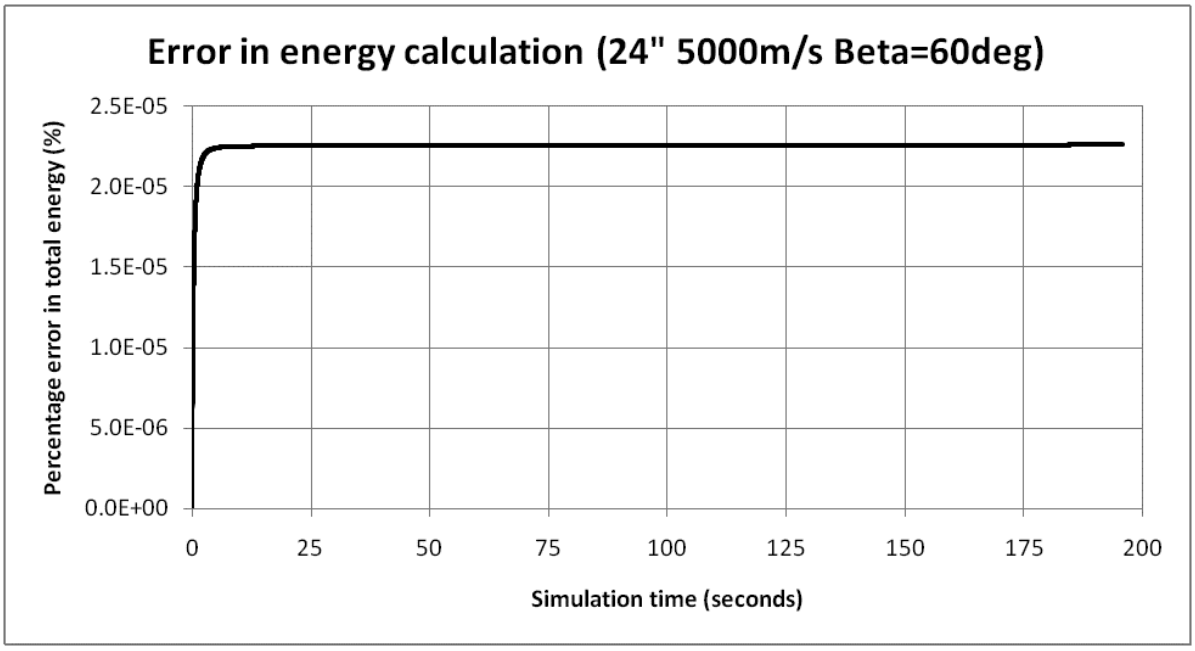
The diagram to the right shows the basis for calculating the energy expended in overcoming the aerodynamic drag (or any force, for that matter). If the sphere is travelling with velocity $\vec{V}$ (stated in the inertial frame of reference) and is subject to a force $\vec{F}$ (also stated in the inertial frame of reference), then the instantaneous mechanical power the force exerts on the sphere is the vector dot product $\vec{F} \cdot \vec{V}$. The energy added to the sphere by this force during some short period of time (say, one time step ΔT) is $\vec{F} \cdot \vec{V} \Delta T$. For a retarding force like the aerodynamic drag, the velocity and drag force will point in almost exactly opposite directions. (The velocity of the sphere relative to the undisturbed air and the drag will be in exactly opposite directions, but the velocity relative in the inertial frame of reference is not the same as the relative velocity.) The mechanical power, and the change in energy, will therefore be algebraically negative.

The following graph shows the three components of energy for the preliminary case described above. The launch speed of 5,000 m/s corresponds to kinetic energy of 12.7 GigaJoules.



The graph shows cumulative "layers" of energy. The lowest layer is the black curve, showing the kinetic energy (K) of the sphere. In the second layer (the red curve), the potential energy (P) is added to the kinetic energy. In the top layer (the blue curve), the cumulative drag (D) is added to the kinetic and potential energies. Note that about 90% of the initial kinetic energy is dissipated by the drag force with the first five seconds of flight. At mid-flight, the kinetic and potential energies have about the same magnitude.

That the blue line, representing the total energy of the system, is constant with simulation time is important. It confirms that energy in the system is conserved, as required by the physics of our universe. (Any material change in the calculation of the total energy of the system could be caused by arithmetic errors, algebraic errors, conceptual errors or numerical errors arising from the numerical integration process.) The following graph shows the size of the error. This is the percentage difference between the calculated sum of the three energy components at any time and the sphere's initial kinetic energy.



The percentage error is always less than 0.000023% – quite good. Almost all of it arises during the first few seconds of flight, when the aerodynamic drag is enormous.

The length of the time step used in the numerical integration process is an important determinant of this error. The numerical integration assumes that the sphere's acceleration remains constant during the whole of each time step. The longer the time steps, the more time is available for that assumption to go wrong. The following table shows the effect of the time step. The preliminary case was run using four different time steps, with the following results.

Time step	Time of flight	Landing latitude	Landing longitude	Energy error
10μs	195.70 s	32.175°	35.620°	0.000226%
5μs	195.70 s	32.175°	35.620°	0.000113%
1μs	195.74 s	32.175°	35.620°	0.000023%
0.5μs	195.74 s	32.175°	35.620°	0.000011%

The energy error is directly proportional to the length of the time step. This is a good indication that the source of these errors is the numerical procedure. The time step can be made as short as one wants, subject to one's willingness to wait for the computer. All the runs carried out for this paper were done on an old, slow, single-processor ThinkPad, and none took took longer than fifteen minutes.

A sphere is not a good projectile for supersonic guns. On the one hand, its symmetry makes it easy to model. On the other hand, it is such a blunt object that almost all of its initial energy is lost to drag. In a subsequent paper, I will look at the exterior ballistics of a more conventional projectile.

Jim Hawley

If you found this description helpful, please let me know. If you spot any errors or omissions, please send an e-mail. Thank you.

Appendix "A"

Expansion of Equation (15)

$$V^2 = \left\{ \begin{array}{c} \left[\dot{r}\cos\lambda\cos(\psi+\omega t) - r\dot{\lambda}\sin\lambda\cos(\psi+\omega t) - r\left(\dot{\psi}+\omega\right)\cos\lambda\sin(\psi+\omega t)\right]^2 + \cdots \\ \cdots + \left[\dot{r}\cos\lambda\sin(\psi+\omega t) - r\dot{\lambda}\sin\lambda\sin(\psi+\omega t) + r\left(\dot{\psi}+\omega\right)\cos\lambda\cos(\psi+\omega t)\right]^2 + \cdots \\ \cdots + \left[\dot{r}\sin\lambda + r\dot{\lambda}\cos\lambda\right]^2 \end{array} \right\}$$

$$= \left\{ \begin{array}{c} \dot{r}^2\cos^2\lambda\cos^2(\psi+\omega t) + r^2\dot{\lambda}^2\sin^2\lambda\cos^2(\psi+\omega t) + r^2\left(\dot{\psi}+\omega\right)^2\cos^2\lambda\sin^2(\psi+\omega t) + \cdots \\ \cdots - 2r\dot{r}\dot{\lambda}\sin\lambda\cos\lambda\cos^2(\psi+\omega t) + \cdots \\ \cdots - 2r\dot{r}\left(\dot{\psi}+\omega\right)\cos^2\lambda\sin(\psi+\omega t)\cos(\psi+\omega t) + \cdots \\ \cdots + 2r^2\dot{\lambda}\left(\dot{\psi}+\omega\right)\sin\lambda\cos\lambda\sin(\psi+\omega t)\cos(\psi+\omega t) + \cdots \\ \cdots + \dot{r}^2\cos^2\lambda\sin^2(\psi+\omega t) + r^2\dot{\lambda}^2\sin^2\lambda\sin^2(\psi+\omega t) + r^2\left(\dot{\psi}+\omega\right)^2\cos^2\lambda\cos^2(\psi+\omega t) + \cdots \\ \cdots - 2r\dot{r}\sin\lambda\cos\lambda\sin^2(\psi+\omega t) + \cdots \\ \cdots + 2r\dot{r}\left(\dot{\psi}+\omega\right)\cos^2\lambda\sin(\psi+\omega t)\cos(\psi+\omega t) + \cdots \\ \cdots - 2r^2\dot{\lambda}\left(\dot{\psi}+\omega\right)\sin\lambda\cos\lambda\sin(\psi+\omega t)\cos(\psi+\omega t) + \cdots \\ \cdots + \dot{r}^2\sin^2\lambda + 2r\dot{r}\dot{\lambda}\sin\lambda\cos\lambda + r^2\dot{\lambda}^2\cos^2\lambda \end{array} \right\}$$

Combine rows #1 and #5, rows #2 and #6, rows #3 and #7, and rows #4 and #8.

$$V^2 = \left\{ \begin{array}{c} \dot{r}^2\cos^2\lambda + r^2\dot{\lambda}^2\sin^2\lambda + r^2\left(\dot{\psi}+\omega\right)^2\cos^2\lambda + \cdots \\ \cdots - 2r\dot{r}\dot{\lambda}\sin\lambda\cos\lambda + \cdots \\ \cdots + \dot{r}^2\sin^2\lambda + 2r\dot{r}\dot{\lambda}\sin\lambda\cos\lambda + r^2\dot{\lambda}^2\cos^2\lambda \end{array} \right\}$$

Combine the two terms in $\dot{r}^2$, the two terms in $r^2\dot{\lambda}^2$ and the two terms in $2r\dot{r}\dot{\lambda}\sin\lambda\cos\lambda$.

$$V^2 = \dot{r}^2 + r^2\dot{\lambda}^2 + r^2\left(\dot{\psi}+\omega\right)^2\cos^2\lambda \qquad (A1)$$

Appendix "B"

Expansion of Equation (17)

The velocity of the sphere with respect to the air, in the $\hat{2}$ frame of reference, is:

$$\vec{V}_{rel} = \left\{ \begin{array}{c} \dot{r}\begin{bmatrix} \cos\lambda\cos(\psi-\psi_0)\cos\lambda_0 + \sin\lambda\sin\lambda_0 \\ \cos\lambda\sin(\psi-\psi_0) \\ -\cos\lambda\cos(\psi-\psi_0)\sin\lambda_0 + \sin\lambda\cos\lambda_0 \end{bmatrix} + \cdots \\ \cdots + r\begin{bmatrix} -\dot{\lambda}\sin\lambda\cos(\psi-\psi_0)\cos\lambda_0 - \dot{\psi}\cos\lambda\sin(\psi-\psi_0)\cos\lambda_0 + \dot{\lambda}\cos\lambda\sin\lambda_0 \\ -\dot{\lambda}\sin\lambda\sin(\psi-\psi_0) + \dot{\psi}\cos\lambda\cos(\psi-\psi_0) \\ \dot{\lambda}\sin\lambda\cos(\psi-\psi_0)\sin\lambda_0 + \dot{\psi}\cos\lambda\sin(\psi-\psi_0)\sin\lambda_0 + \dot{\lambda}\cos\lambda\cos\lambda_0 \end{bmatrix} \end{array} \right\}$$

The relative speed is:

$$V_{rel}^2 = \left\{ \begin{array}{c} \left[\begin{array}{c} \dot{r}\cos\lambda\cos(\psi-\psi_0)\cos\lambda_0 + \dot{r}\sin\lambda\sin\lambda_0 + \cdots \\ \cdots - r\dot{\lambda}\sin\lambda\cos(\psi-\psi_0)\cos\lambda_0 - r\dot{\psi}\cos\lambda\sin(\psi-\psi_0)\cos\lambda_0 + r\dot{\lambda}\cos\lambda\sin\lambda_0 \end{array} \right]^2 + \cdots \\ \cdots + \left[\begin{array}{c} \dot{r}\cos\lambda\sin(\psi-\psi_0) + \cdots \\ \cdots - r\dot{\lambda}\sin\lambda\sin(\psi-\psi_0) + r\dot{\psi}\cos\lambda\cos(\psi-\psi_0) \end{array} \right]^2 + \cdots \\ \cdots + \left[\begin{array}{c} -\dot{r}\cos\lambda\cos(\psi-\psi_0)\sin\lambda_0 + \dot{r}\sin\lambda\cos\lambda_0 + \cdots \\ \cdots + r\dot{\lambda}\ \sin\lambda\cos(\psi-\psi_0)\sin\lambda_0 + r\dot{\psi}\cos\lambda\sin(\psi-\psi_0)\sin\lambda_0 + r\dot{\lambda}\cos\lambda\cos\lambda_0 \end{array} \right]^2 \end{array} \right\}$$

$$= \left\{ \begin{array}{ll} \#1 & \dot{r}^2\cos^2\lambda\cos^2(\psi-\psi_0)\cos^2\lambda_0 + \cdots \\ \#2 & \cdots + \dot{r}^2\sin^2\lambda\sin^2\lambda_0 + \cdots \\ \#3 & \cdots + r^2\dot{\lambda}^2\sin^2\lambda\cos^2(\psi-\psi_0)\cos^2\lambda_0 + \cdots \\ \#4 & \cdots + r^2\dot{\psi}^2\cos^2\lambda\sin^2(\psi-\psi_0)\cos^2\lambda_0 + \cdots \\ \#5 & \cdots + r^2\dot{\lambda}^2\cos^2\lambda\sin^2\lambda_0 + \cdots \\ \#6 & \cdots + 2\dot{r}^2\sin\lambda\cos\lambda\cos(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#7 & \cdots - 2r\dot{r}\dot{\lambda}\sin\lambda\cos\lambda\cos^2(\psi-\psi_0)\cos^2\lambda_0 + \cdots \\ \#8 & \cdots - 2r\dot{r}\dot{\psi}\cos^2\lambda\sin(\psi-\psi_0)\cos(\psi-\psi_0)\cos^2\lambda_0 + \cdots \\ \#9 & \cdots + 2\dot{r}r\dot{\lambda}\cos^2\lambda\cos(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#10 & \cdots - 2r\dot{r}\dot{\lambda}\sin^2\lambda\cos(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#11 & \cdots - 2\dot{r}r\dot{\psi}\sin\lambda\cos\lambda\sin(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#12 & \cdots + 2\dot{r}r\dot{\lambda}\sin\lambda\cos\lambda\sin^2\lambda_0 + \cdots \\ \#13 & \cdots + 2r^2\dot{\lambda}\dot{\psi}\sin\lambda\cos\lambda\sin(\psi-\psi_0)\cos(\psi-\psi_0)\cos^2\lambda_0 + \cdots \\ \#14 & \cdots - 2r^2\dot{\lambda}^2\sin\lambda\cos\lambda\cos(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#15 & \cdots - 2r^2\dot{\lambda}\dot{\psi}\cos^2\lambda\sin(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#16 & \cdots + \dot{r}^2\cos^2\lambda\sin^2(\psi-\psi_0) + \cdots \\ \#17 & \cdots + r^2\dot{\lambda}^2\sin^2\lambda\sin^2(\psi-\psi_0) + \cdots \\ \#18 & \cdots + r^2\dot{\psi}^2\cos^2\lambda\cos^2(\psi-\psi_0) + \cdots \\ & \cdots + \text{continued on next page} \end{array} \right\}$$

$$V_{rel}^2 = \left\{ \begin{array}{ll} & \text{continued from previous page} + \cdots \\ \#19 & \cdots - 2r\dot{r}\dot{\lambda}\sin\lambda\cos\lambda\sin^2(\psi-\psi_0) + \cdots \\ \#20 & \cdots + 2r\dot{r}\dot{\psi}\cos^2\lambda\sin(\psi-\psi_0)\cos(\psi-\psi_0) + \cdots \\ \#21 & \cdots - 2r^2\dot{\lambda}\dot{\psi}\sin\lambda\cos\lambda\sin(\psi-\psi_0)\cos(\psi-\psi_0) + \cdots \\ \#22 & \cdots + \dot{r}^2\cos^2\lambda\cos^2(\psi-\psi_0)\sin^2\lambda_0 + \cdots \\ \#23 & \cdots + \dot{r}^2\sin^2\lambda\cos^2\lambda_0 + \cdots \\ \#24 & \cdots + r^2\dot{\lambda}^2\sin^2\lambda\cos^2(\psi-\psi_0)\sin^2\lambda_0 + \cdots \\ \#25 & \cdots + r^2\dot{\psi}^2\cos^2\lambda\sin^2(\psi-\psi_0)\sin^2\lambda_0 + \cdots \\ \#26 & \cdots + r^2\dot{\lambda}^2\cos^2\lambda\cos^2\lambda_0 + \cdots \\ \#27 & \cdots - 2\dot{r}^2\sin\lambda\cos\lambda\cos(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#28 & \cdots - 2r\dot{r}\dot{\lambda}\sin\lambda\cos\lambda\ \cos^2(\psi-\psi_0)\sin^2\lambda_0 + \cdots \\ \#29 & \cdots - 2r\dot{r}\dot{\psi}\cos^2\lambda\sin(\psi-\psi_0)\cos(\psi-\psi_0)\sin^2\lambda_0 + \cdots \\ \#30 & \cdots - 2\dot{r}r\dot{\lambda}\cos^2\lambda\cos(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#31 & \cdots + 2r\dot{r}\dot{\lambda}\sin^2\lambda\cos(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#32 & \cdots + 2\dot{r}r\dot{\psi}\sin\lambda\cos\lambda\sin(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#33 & \cdots + 2\dot{r}r\dot{\lambda}\sin\lambda\cos\lambda\cos^2\lambda_0 + \cdots \\ \#34 & \cdots + 2r^2\dot{\lambda}\dot{\psi}\sin\lambda\cos\lambda\sin(\psi-\psi_0)\cos(\psi-\psi_0)\sin^2\lambda_0 + \cdots \\ \#35 & \cdots + 2r^2\dot{\lambda}^2\sin\lambda\cos\lambda\cos(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 + \cdots \\ \#36 & \cdots + 2r^2\dot{\lambda}\dot{\psi}\cos^2\lambda\sin(\psi-\psi_0)\sin\lambda_0\cos\lambda_0 \end{array} \right\}$$

Rows can be combined using the trigonometric identity $\sin^2 A + \cos^2 A = 1$ or subtraction. We get:

$$V_{rel}^2 = \left\{ \begin{array}{lll} \#1 + \#22 & \dot{r}^2\cos^2\lambda\cos^2(\psi-\psi_0) + \cdots & \#37 \\ \#2 + \#23 & \cdots + \dot{r}^2\sin^2\lambda + \cdots & \#38 \\ \#3 + \#24 & \cdots + r^2\dot{\lambda}^2\sin^2\lambda\cos^2(\psi-\psi_0) + \cdots & \#39 \\ \#4 + \#25 & \cdots + r^2\dot{\psi}^2\cos^2\lambda\sin^2(\psi-\psi_0) + \cdots & \#40 \\ \#5 + \#26 & \cdots + r^2\dot{\lambda}^2\cos^2\lambda + \cdots & \#41 \\ \#6 + \#27 & \cdots + 0 + \cdots & \\ \#7 + \#28 & \cdots - 2r\dot{r}\dot{\lambda}\sin\lambda\cos\lambda\cos^2(\psi-\psi_0) + \cdots & \#42 \\ \#8 + \#29 & \cdots - 2r\dot{r}\dot{\psi}\cos^2\lambda\sin(\psi-\psi_0)\cos(\psi-\psi_0) + \cdots & \#43 \\ \#9 + \#30 & \cdots + 0 + \cdots & \\ \#10 + \#31 & \cdots + 0 + \cdots & \\ \#11 + \#32 & \cdots + 0 + \cdots & \\ \#12 + \#33 & \cdots + 2\dot{r}r\dot{\lambda}\sin\lambda\cos\lambda + \cdots & \#44 \\ \#13 + \#34 & \cdots + 2r^2\dot{\lambda}\dot{\psi}\sin\lambda\cos\lambda\sin(\psi-\psi_0)\cos(\psi-\psi_0) + \cdots & \#45 \\ \#14 + \#35 & \cdots + 0 + \cdots & \\ \#15 + \#36 & \cdots + 0 + \cdots & \\ \#16 & \cdots + \dot{r}^2\cos^2\lambda\sin^2(\psi-\psi_0) + \cdots & \#16 \\ \#17 & \cdots + r^2\dot{\lambda}^2\sin^2\lambda\sin^2(\psi-\psi_0) + \cdots & \#17 \\ \#18 & \cdots + r^2\dot{\psi}^2\cos^2\lambda\cos^2(\psi-\psi_0) + \cdots & \#18 \\ \#19 & \cdots - 2r\dot{r}\dot{\lambda}\sin\lambda\cos\lambda\sin^2(\psi-\psi_0) + \cdots & \#19 \\ \#20 & \cdots + 2r\dot{r}\dot{\psi}\cos^2\lambda\sin(\psi-\psi_0)\cos(\psi-\psi_0) + \cdots & \#20 \\ \#21 & \cdots - 2r^2\dot{\lambda}\dot{\psi}\sin\lambda\cos\lambda\sin(\psi-\psi_0)\cos(\psi-\psi_0) & \#21 \end{array} \right\}$$

Again, rows can be combined using the trigonometric identity $\sin^2 A + \cos^2 A = 1$ or subtraction. We get:

$$V_{rel}^2 = \begin{Bmatrix} \#37 + \#16 & \dot{r}^2 \cos^2\lambda + \cdots & \#46 \\ \#38 & \cdots + \dot{r}^2 \sin^2\lambda + \cdots & \#38 \\ \#39 + \#17 & \cdots + r^2\dot{\lambda}^2 \sin^2\lambda + \cdots & \#47 \\ \#40 + \#18 & \cdots + r^2\dot{\psi}^2 \cos^2\lambda + \cdots & \#48 \\ \#41 & \cdots + r^2\dot{\lambda}^2 \cos^2\lambda + \cdots & \#41 \\ \#42 + \#19 & \cdots - 2r\dot{r}\dot{\lambda} \sin\lambda \cos\lambda + \cdots & \#49 \\ \#43 + \#20 & \cdots + 0 + \cdots & \\ \#44 & \cdots + 2\dot{r}r\dot{\lambda} \sin\lambda \cos\lambda + \cdots & \#44 \\ \#45 + \#21 & \cdots + 0 & \end{Bmatrix}$$

Just a few more steps, and we get:

$$V_{rel}^2 = \begin{Bmatrix} \#46 + \#38 & \dot{r}^2 + \cdots \\ \#47 + \#41 & \cdots + r^2\dot{\lambda}^2 + \cdots \\ \#48 & \cdots + r^2\dot{\psi}^2 \cos^2\lambda + \cdots \\ \#49 + \#44 & \cdots + 0 \end{Bmatrix}$$

$$= \dot{r}^2 + r^2\dot{\lambda}^2 + r^2\dot{\psi}^2 \cos^2\lambda \qquad (B1)$$

Appendix "C"

The twelve functions and the Eulerian substitution procedure

$$\frac{d^2\vec{S}_I}{dt^2} = \left\{ \begin{array}{c} \ddot{r}\begin{bmatrix}\cos\lambda\cos(\psi+\omega t)\\ \cos\lambda\sin(\psi+\omega t)\\ \sin\lambda\end{bmatrix} + 2\dot{r}\begin{bmatrix}-\dot{\lambda}\sin\lambda\cos(\psi+\omega t) - (\dot{\psi}+\omega)\cos\lambda\sin(\psi+\omega t)\\ -\dot{\lambda}\sin\lambda\sin(\psi+\omega t) + (\dot{\psi}+\omega)\cos\lambda\cos(\psi+\omega t)\\ \dot{\lambda}\cos\lambda\end{bmatrix} + \cdots \\ \cdots + r\begin{bmatrix}-\ddot{\lambda}\sin\lambda\cos(\psi+\omega t) - \dot{\lambda}^2\cos\lambda\cos(\psi+\omega t) + \dot{\lambda}(\dot{\psi}+\omega)\sin\lambda\sin(\psi+\omega t)\\ -\ddot{\lambda}\sin\lambda\sin(\psi+\omega t) - \dot{\lambda}^2\cos\lambda\sin(\psi+\omega t) - \dot{\lambda}(\dot{\psi}+\omega)\sin\lambda\cos(\psi+\omega t)\\ \ddot{\lambda}\cos\lambda - \dot{\lambda}^2\sin\lambda\end{bmatrix} + \cdots \\ \cdots + r\begin{bmatrix}-\ddot{\psi}\cos\lambda\sin(\psi+\omega t) + \dot{\lambda}(\dot{\psi}+\omega)\sin\lambda\sin(\psi+\omega t) - (\dot{\psi}+\omega)^2\cos\lambda\cos(\psi+\omega t)\\ \ddot{\psi}\cos\lambda\cos(\psi+\omega t) - \dot{\lambda}(\dot{\psi}+\omega)\sin\lambda\cos(\psi+\omega t) - (\dot{\psi}+\omega)^2\cos\lambda\sin(\psi+\omega t)\\ 0\end{bmatrix} \end{array} \right\}$$

$$= \ddot{r}\begin{bmatrix}\cos\lambda\cos(\psi+\omega t)\\ \cos\lambda\sin(\psi+\omega t)\\ \sin\lambda\end{bmatrix} + \ddot{\lambda}\begin{bmatrix}-r\sin\lambda\cos(\psi+\omega t)\\ -r\sin\lambda\sin(\psi+\omega t)\\ r\cos\lambda\end{bmatrix} + \ddot{\psi}\begin{bmatrix}-r\cos\lambda\sin(\psi+\omega t)\\ r\cos\lambda\cos(\psi+\omega t)\\ 0\end{bmatrix} + \begin{bmatrix}a_{14}\\ a_{24}\\ a_{34}\end{bmatrix} \qquad (C1)$$

where:

$$a_{14} = \left\{ \begin{array}{c} -2\dot{r}\dot{\lambda}\sin\lambda\cos(\psi+\omega t) - 2\dot{r}(\dot{\psi}+\omega)\cos\lambda\sin(\psi+\omega t) + \cdots \\ \cdots - r\dot{\lambda}^2\cos\lambda\cos(\psi+\omega t) + r\dot{\lambda}(\dot{\psi}+\omega)\sin\lambda\sin(\psi+\omega t) + \cdots \\ \cdots + r\dot{\lambda}(\dot{\psi}+\omega)\sin\lambda\sin(\psi+\omega t) - r(\dot{\psi}+\omega)^2\cos\lambda\cos(\psi+\omega t) \end{array} \right\} \qquad (C2)$$

$$a_{24} = \left\{ \begin{array}{c} -2\dot{r}\dot{\lambda}\sin\lambda\sin(\psi+\omega t) + 2\dot{r}(\dot{\psi}+\omega)\cos\lambda\cos(\psi+\omega t) + \cdots \\ \cdots - r\dot{\lambda}^2\cos\lambda\sin(\psi+\omega t) - r\dot{\lambda}(\dot{\psi}+\omega)\sin\lambda\cos(\psi+\omega t) + \cdots \\ \cdots - r\dot{\lambda}(\dot{\psi}+\omega)\sin\lambda\cos(\psi+\omega t) - r(\dot{\psi}+\omega)^2\cos\lambda\sin(\psi+\omega t) \end{array} \right\} \qquad (C3)$$

$$a_{34} = \left\{ \begin{array}{c} 2\dot{r}\dot{\lambda}\cos\lambda + \cdots \\ \cdots - r\dot{\lambda}^2\sin\lambda \end{array} \right\} \qquad (C4)$$

And, from their positions in Equation $(C1)$, we have:

$$\left. \begin{array}{ll} a_{11} & = \cos\lambda\cos(\psi+\omega t) \\ a_{12} & = -r\sin\lambda\cos(\psi+\omega t) \\ a_{13} & = -r\cos\lambda\sin(\psi+\omega t) \\ a_{21} & = \cos\lambda\sin(\psi+\omega t) \\ a_{22} & = -r\sin\lambda\sin(\psi+\omega t) \\ a_{23} & = r\cos\lambda\cos(\psi+\omega t) \\ a_{31} & = \sin\lambda \\ a_{32} & = r\cos\lambda \\ a_{33} & = 0 \end{array} \right\} \qquad (C5)$$

With the appropriate substitutions for $\vec{b}$, the matrix equation can be written as:

$$\begin{bmatrix}\cos\lambda\cos(\psi+\omega t) & -\sin\lambda\cos(\psi+\omega t) & -\cos\lambda\sin(\psi+\omega t)\\ \cos\lambda\sin(\psi+\omega t) & -\sin\lambda\sin(\psi+\omega t) & \cos\lambda\cos(\psi+\omega t)\\ \sin\lambda & \cos\lambda & 0\end{bmatrix} \cdot \begin{bmatrix}\ddot{r}\\ r\ddot{\lambda}\\ r\ddot{\psi}\end{bmatrix} = \begin{bmatrix}b_1\\ b_2\\ b_3\end{bmatrix} \qquad (C6)$$

Dividing each row by its leading element gives:

$$\begin{bmatrix} 1 & -\tan\lambda & -\tan(\psi+\omega t) \\ 1 & -\tan\lambda & \dfrac{1}{\tan(\psi+\omega t)} \\ 1 & \dfrac{1}{\tan\lambda} & 0 \end{bmatrix} \cdot \begin{bmatrix} \ddot{r} \\ r\ddot{\lambda} \\ r\ddot{\psi} \end{bmatrix} = \begin{bmatrix} \dfrac{b_1}{\cos\lambda\cos(\psi+\omega t)} \\ \dfrac{b_2}{\cos\lambda\sin(\psi+\omega t)} \\ \dfrac{b_3}{\sin\lambda} \end{bmatrix} \quad (C7)$$

Subtracting the first row from each of the last two rows gives:

$$\begin{bmatrix} 1 & -\tan\lambda & -\tan(\psi+\omega t) \\ 0 & 0 & \dfrac{1+\tan^2(\psi+\omega t)}{\tan(\psi+\omega t)} \\ 0 & \dfrac{1+\tan^2\lambda}{\tan\lambda} & \tan(\psi+\omega t) \end{bmatrix} \cdot \begin{bmatrix} \ddot{r} \\ r\ddot{\lambda} \\ r\ddot{\psi} \end{bmatrix} = \begin{bmatrix} \dfrac{b_1}{\cos\lambda\cos(\psi+\omega t)} \\ \dfrac{b_2}{\cos\lambda\sin(\psi+\omega t)} - \dfrac{b_1}{\cos\lambda\cos(\psi+\omega t)} \\ \dfrac{b_3}{\sin\lambda} - \dfrac{b_1}{\cos\lambda\cos(\psi+\omega t)} \end{bmatrix} \quad (C8)$$

Note that:

$$\begin{aligned} \frac{1}{\tan A} + \tan A &= \frac{1+\tan^2 A}{\tan A} \\ &= \frac{1}{\sin A\cos A} \end{aligned} \quad (C9)$$

so that:

$$\begin{bmatrix} 1 & -\tan\lambda & -\tan(\psi+\omega t) \\ 0 & 0 & \dfrac{1}{\sin(\psi+\omega t)\cos(\psi+\omega t)} \\ 0 & \dfrac{1}{\sin\lambda\cos\lambda} & \tan(\psi+\omega t) \end{bmatrix} \cdot \begin{bmatrix} \ddot{r} \\ r\ddot{\lambda} \\ r\ddot{\psi} \end{bmatrix} = \begin{bmatrix} \dfrac{b_1}{\cos\lambda\cos(\psi+\omega t)} \\ \dfrac{b_2}{\cos\lambda\sin(\psi+\omega t)} - \dfrac{b_1}{\cos\lambda\cos(\psi+\omega t)} \\ \dfrac{b_3}{\sin\lambda} - \dfrac{b_1}{\cos\lambda\cos(\psi+\omega t)} \end{bmatrix} \quad (C10)$$

The second equation can immediately be solved for $r\ddot{\psi}$:

$$\begin{aligned} r\ddot{\psi} &= \left[\frac{b_2}{\cos\lambda\sin(\psi+\omega t)} - \frac{b_1}{\cos\lambda\cos(\psi+\omega t)}\right]\sin(\psi+\omega t)\cos(\psi+\omega t) \\ &= \frac{b_2\cos(\psi+\omega t) - b_1\sin(\psi+\omega t)}{\cos\lambda} \end{aligned} \quad (C11)$$

With this known value of $r\ddot{\psi}$, the third equation can be solved for $r\ddot{\lambda}$:

$$\frac{r\ddot{\lambda}}{\sin\lambda\cos\lambda} + r\ddot{\psi}\tan(\psi+\omega t) = \frac{b_3}{\sin\lambda} - \frac{b_1}{\cos\lambda\cos(\psi+\omega t)}$$

$$\rightarrow \quad r\ddot{\lambda} = \left[\frac{b_3}{\sin\lambda} - \frac{b_1}{\cos\lambda\cos(\psi+\omega t)} - r\ddot{\psi}\tan(\psi+\omega t)\right]\sin\lambda\cos\lambda$$

$$\rightarrow \quad r\ddot{\lambda} = \left[b_3\cos\lambda - \frac{b_1\sin\lambda}{\cos(\psi+\omega t)} - r\ddot{\psi}\sin\lambda\cos\lambda\tan(\psi+\omega t)\right] \qquad (C12)$$

Finally, with these known values of $r\ddot{\lambda}$ and $r\ddot{\psi}$, the third equation can be solved for $\ddot{r}$:

$$\ddot{r} = \frac{b_1}{\cos\lambda\cos(\psi+\omega t)} + r\ddot{\lambda}\tan\lambda + r\ddot{\psi}\tan(\psi+\omega t) \qquad (C13)$$

This procedure will succeed so long as none of the denominators in Equations $(C7)$ through $(C13)$ happens to be zero. If one of them happens to be zero, then the system of equations is even simpler than assumed here, and can be solved with some minor adjustments.

Appendix "D"

Listing of the VB2010 code for the preliminary case

The program consists of a Windows Forms application (Form1) and five modules: Variables, Procedures, MagnitudeOfGravity, USStandardAtmosphere and CoefficientOfDrag,

Windows Form application Form1

```
Option Strict On
Option Explicit On

Public Class Form1
    Inherits System.Windows.Forms.Form

    Public Sub New()
        InitializeComponent()
        With Me
            Text = "Exterior ballistics of a cannonball"
            FormBorderStyle = Windows.Forms.FormBorderStyle.None
            Size = New Drawing.Size(900, 600)
            MinimizeBox = True
            MaximizeBox = True
            FormBorderStyle = Windows.Forms.FormBorderStyle.Fixed3D
            With Me
                Controls.Add(buttonStart) : buttonStart.BringToFront()
                Controls.Add(buttonExit) : buttonExit.BringToFront()
                Controls.Add(labelResults) : labelResults.BringToFront()
            End With
            Visible = True
            PerformLayout()
        End With
        InitializeCDLookupTable()
        InitializeAtmosphereLayers()
    End Sub

'////////////////////////////////////////////////////////////////////////////////
'// Controls and handles
'////////////////////////////////////////////////////////////////////////////////

    Private WithEvents buttonStart As New Windows.Forms.Button With _
       {.Size = New Drawing.Size(100, 30), _
        .Location = New Drawing.Point(5, 5), _
        .Text = "Start simulation"}
    Private Sub buttonStart_Click() Handles buttonStart.Click
        ' Initialize the sphere's location and speed
        ConvertGunParametersToSphereStartParameters()
        ' Open the output text file
        FileWriter = New System.IO.StreamWriter( _
            ThisDirectory & "\" & OutputFileName)
        ' Write an information header in the output text file
        FileWriter.WriteLine("Sphere diameter (inches) = " & Trim(Str(DsphereInch)))
        FileWriter.WriteLine("Launch speed (m's) = " & Trim(Str(Speed0)))
        FileWriter.WriteLine("Launch beta (deg) = " & Trim(Str(Beta0deg)))
        FileWriter.WriteLine("Launch alpha (deg) = " & Trim(Str(Alpha0deg)))
        ' Write the header for rows in the output text file
        FileWriter.Write("Time(s), Altitude(m), Latitude(deg), Longitude(deg), ")
        FileWriter.Write("DistanceX_M(m), DistanceY_M(m), DistanceZ_M(m), ")
```

```
        FileWriter.Write("SpeedX_I(m/s), SpeedY_I(m/s), SpeedZ_I(m/s), ")
        FileWriter.Write("Speed_I(m/s), ")
        FileWriter.Write("RelSpeedX_I(m/s), RelSpeedY_I(m/s), RelSpeedZ_I(m/s), ")
        FileWriter.Write("RelSpeed_I(m/s), ")
        FileWriter.Write("SpeedSound(m/s), Mach number, CD, ")
        FileWriter.Write("RHOair(kg/m^3), Faero(N), Fgrav(N), ")
        FileWriter.Write("GravForceX_I(N), GravForceY_I(N), GravForceZ_I(N), ")
        FileWriter.Write("DragForceX_I(N), DragForceY_I(N), DragForceZ_I(N), ")
        FileWriter.Write("AccelRad_I(m/s^2), AccelLat_I(r/s^2), AccelLong_I(r/s^2), ")
        FileWriter.Write("KineticEng(J), CumGravPwr(J), CumDragPwr(J), ")
        FileWriter.WriteLine("TotalEng(J), EnergyErr(%)")
        ' Initialize the simulation variables
        Time = -deltaT
        WriteCounter = WriteInterval + 1
        ScreenCounter = 0
        ' Main loop
        Do
            ' Increment the simulation time
            Time = Time + deltaT
            ' Check to see if the maximum simulation time has been exceeded
            If (Time > MaxTime) Then
                ' Write final iteration to output file
                WriteResultsToOutputTextFile()
                MsgBox("Maximum simulation time has been exceeded.")
                Exit Do
            End If
            ' Proceed through one time step
            ExecuteOneTimeStep()
            ' Check to see if the sphere has landed
            If ((Time > 10) And (SphereInstRadius_start < TargetRadius)) Then
                ' Write final iteration to output file
                WriteResultsToOutputTextFile()
                MsgBox("The sphere has landed.")
                Exit Do
            End If
        Loop
        ' Close the output text file
        FileWriter.Close()
    End Sub

    Private WithEvents buttonExit As New Windows.Forms.Button With _
       {.Size = New Drawing.Size(100, 30), _
        .Location = New Drawing.Point(5, 40), _
        .Text = "Exit"}
    Private Sub buttonSExit_Click() Handles buttonExit.Click
        Application.Exit()
    End Sub

    Public labelResults As New Windows.Forms.Label With _
        {.Size = New Drawing.Size(900, 500), _
         .Location = New Drawing.Point(110, 5), _
         .Text = ""}

End Class
```

Module Variables

```
Option Strict On
Option Explicit On

Public Module Variables

    ' ***** Time ********************************************************************
    ' Time = Simulation time, in seconds
    ' deltaT = Duration of time step, in seconds
    ' MaxTime = Maximum simulation time, in seconds
    ' WriteInterval = Number of time steps between write to output text file
    ' WriteCounter = Counter of time steps since last write to output text file
    ' ScreenInterval = Number of time steps between updates to the screen
    ' ScreenCounter = Counter of time steps since last update to the screen
    Public Time As Double
    Public deltaT As Double = 0.000001
    Public MaxTime As Double = 360
    Public WriteInterval As Int32 = CInt(10000)
    Public WriteCounter As Int32
    Public ScreenInterval As Int32 = CInt(10000)
    Public ScreenCounter As Int32

    ' ***** Physical constants ********************************************************
    ' EarthRadius = Radius of the Earth, to mean sea level, in meters
    ' Omega = Rotation speed of the Earth, radians per sidereal second
    Public EarthRadius As Double = Val("6356766")
    Public Omega As Double = 2 * Math.PI / (23.9344699 * 3600)

    ' ***** Sphere's properties *******************************************************
    ' DsphereInch = Diameter of sphere, in inches
    ' Rsphere = Radius of sphere, in meters
    ' Asphere = Frontal area of sphere, in square meters
    ' RHOsphere = Density of hard steel, in kilograms per cubic meter
    ' Msphere = Mass of sphere, in kilograms
    Public DsphereInch As Double = 24
    Public Rsphere As Double = DsphereInch * 2.54 / 200
    Public Asphere As Double = Math.PI * (Rsphere ^ 2)
    Public RHOsphere As Double = 7900
    Public Msphere As Double = (4 / 3) * Asphere * Rsphere * RHOsphere

    ' ***** Firing parameters *********************************************************
    ' Speed0 = Exit speed from muzzle, in meters per second
    ' Alpha0deg = Exit angle north of true east, in degrees
    ' Alpha0rad = Alpha0deg, expressed in radians
    ' Beta0deg = Exit angle above the horizontal plane, in degrees
    ' Beta0rad = Beta0deg, expressed in radians
    ' MuzzleHeight = Height of muzzle above local ground, in meters
    Public Speed0 As Double = 5000
    Public Alpha0deg As Double = 8
    Public Alpha0rad As Double = Alpha0deg * Math.PI / 180
    Public Beta0deg As Double = 60
    Public Beta0rad As Double = Beta0deg * Math.PI / 180
    Public MuzzleHeight As Double = 10

    ' ***** Launch site ***************************************************************
    ' LaunchLatDeg = Launch site latitude, in decimal degrees
    ' LaunchLatRad = Launch site latitude, in radians
    ' LaunchLongDeg = Launch site longitude, in decimal degrees
```

```
' LaunchLongRad = Launch site longitude, in radians
' LaunchAltAMSL = Launch site altitude, meters above mean sea level
' LaunchRadius = Radius of muzzle from Earth's center, in meters
Public LaunchLatDeg As Double = 32.08275
Public LaunchLatRad As Double = LaunchLatDeg * Math.PI / 180
Public LaunchLongDeg As Double = 34.76776
Public LaunchLongRad As Double = LaunchLongDeg * Math.PI / 180
Public LaunchAltAMSL As Double = 7.26
Public LaunchRadius As Double = EarthRadius + LaunchAltAMSL + MuzzleHeight

' ***** Sphere's instantaneous kinematic variables *********************************
' Location variables:
'   SphereInstLatRad = Instantaneous latitude, in radians
'   SphereInstLongRad = Instantaneous longitude, in radians
'   SphereInstRadius = Instantaneous distance from Earth's center, in meters
Public SphereInstLatRad_start, SphereInstLatRad_end As Double
Public SphereInstLongRad_start, SphereInstLongRad_end As Double
Public SphereInstRadius_start, SphereInstRadius_end As Double
' Derivatives of location variables:
'   SphereInstLatDot = Derivative of SphereInstLatRad
'   SphereInstLongDot = Derivative of SphereInstLongRad
'   SphereInstRadiusDot = Derivative of SphereInstRadius
Public SphereInstLatDot_start, SphereInstLatDot_end As Double
Public SphereInstLongDot_start, SphereInstLongDot_end As Double
Public SphereInstRadiusDot_start, SphereInstRadiusDot_end As Double
' Second derivatives of location variables:
'   SphereInstLatDot2 = Derivative of SphereInstLatDot
'   SphereInstLongDot2 = Derivative of SphereInstLongDot
'   SphereInstRadiusDot2 = Derivative of SphereInstRadiusDot
Public SphereInstLatDot2 As Double
Public SphereInstLongDot2 As Double
Public SphereInstRadiusDot2 As Double
' Location of the sphere in the inertial frame, in meters:
Public SphereInstX_I_start, SphereInstX_I_end As Double
Public SphereInstY_I_start, SphereInstY_I_end As Double
Public SphereInstZ_I_start, SphereInstZ_I_end As Double
' Speed of the sphere in the inertial frame, in meters per second:
'   SphereInstSpeed_I = Total instantaneous speed, in m/s
'   SphereInstSpeedX_I = X-I component of SphereInstSpeed_I
'   SphereInstSpeedY_I = Y_I component of SphereInstSpeed_I
'   SphereInstSpeedZ_I = Z_I component of SphereInstSpeed_I
Public SphereInstSpeed_I_start, SphereInstSpeed_I_end As Double
Public SphereInstSpeedX_I_start, SphereInstSpeedX_I_end As Double
Public SphereInstSpeedY_I_start, SphereInstSpeedY_I_end As Double
Public SphereInstSpeedZ_I_start, SphereInstSpeedZ_I_end As Double
' Location of the sphere w.r.t. the muzzle, in the muzzle's 2-frame, in meters:
Public SphereInstX_M_start As Double
Public SphereInstY_M_start As Double
Public SphereInstZ_M_start As Double
' Speed of the sphere w.r.t. the air, in the inertial frame, in meters per second
'   SphereInstRelSpeed_I = Total relative instantaneous speed, in m/s
'   SphereInstRelSpeedX_I = X_I component of SphereRelInstSpeed_I
'   SphereInstRelSpeedY_I = Y_I component of SphereRelInstSpeed_I
'   SphereInstRelSpeedZ_I = Z_I component of SphereRelInstSpeed_I
Public SphereInstRelSpeed_I_start As Double
Public SphereInstRelSpeedX_I_start As Double
Public SphereInstRelSpeedY_I_start As Double
Public SphereInstRelSpeedZ_I_start As Double
```

```
' ***** Sphere's instantaneous dynamic and associated variables ******************
' SphereInstGeometricAltitude = Sphere's instantaneous geometric altitude, meters ASL
' SphereInstGeopotentialAltitude = Sphere's corresponding geopotential altitude
' LocalTemperature = Local temperature at sphere's altitude, in degK
' LocalPressure = Local static pressure at sphere's altitude, in N/m^2
' LocalDensity = Local air density at sphere's altitude, in kg/m^3
' LocalDviscosity = Local air dynamic viscosity at sphere's altitude
' LocalKviscosity = Local air kinematic viscosity at sphere's altitude
' LocalSpeedOfSound = Local speed of sound at sphere's altitude, in m/s
' SphereInstMach = Sphere's instantaneous Mach number
' SphereInstCD = Sphere's instantaneous coefficient of drag
' SphereInstDragF = Sphere's instantaneous total drag, in Newtons
' SphereInstRe = Sphere's instantaneous Reynold's number
' SphereInstGravF = Sphere's instantaneous gravitational force, in Newtons
' GravFOnSphereX_I, GravFOnSphereY_I, GravFOnSphereZ_I = Three components
' DragFOnSphereX_I, DragFOnSphereY_I, DragFOnSphereZ_I = Three components
' TotalFOnSphereX_I, TotalFOnSphereY_I, TotalFOnSphereZ_I = Three components
Public SphereInstGeometricAltitude As Double
Public SphereInstGeopotentialAltitude As Double
Public LocalTemperature As Double
Public LocalPressure As Double
Public LocalDensity As Double
Public LocalDviscosity As Double
Public LocalKviscosity As Double
Public LocalSpeedOfSound As Double
Public SphereInstMach As Double
Public SphereInstCD As Double
Public SphereInstDragF As Double
Public SphereInstRe As Double
Public SphereInstGravF As Double
Public GravFOnSphereX_I, GravFOnSphereY_I, GravFOnSphereZ_I As Double
Public DragFOnSphereX_I, DragFOnSphereY_I, DragFOnSphereZ_I As Double
Public TotalFOnSphereX_I, TotalFOnSphereY_I, TotalFOnSphereZ_I As Double

' ***** Target site ***************************************************************
' TargetLatDeg = Target site latitude, in decimal degrees
' TargetLatRad = Target site latitude, in radians
' TargetLongDeg = Target site longitude, in decimal degrees
' TargetLongRad = Target site longitude, in radians
' TargetAltAMSL = Target site altitude, meters above mean sea level
' TargetRadius = Target site radius from Earth's center, in meters
Public TargetLatDeg As Double = 33.3135
Public TargetLatRad As Double = TargetLatDeg * Math.PI / 180
Public TargetLongDeg As Double = 44.4135
Public TargetLongRad As Double = TargetLongDeg * Math.PI / 180
Public TargetAltAMSL As Double = 29
Public TargetRadius As Double = EarthRadius + TargetAltAMSL

' ***** Landing site **************************************************************
' LandingLatDeg = Landing site latitude, in decimal degrees
' LandingLatRad = Landing site latitude, in radians
' LandingLongDeg = Landing site longitude, in decimal degrees
' LandingLongRad = Landing site longitude, in radians
Public LandingLatDeg As Double
Public LandingLatRad As Double
Public LandingLongDeg As Double
Public LandingLongRad As Double
```

```
    ' ***** Landing error ************************************************************
    ' ErrorLatRad = Landing error in latitude, in radians
    ' ErrorLongRad = Landing error in longitude, in radians
    ' ErrorLatDeg = Landing error in latitude, in decimal degrees
    ' ErrorLongDeg = Landing error in longitude, in decimal degrees
    Public ErrorLatRad As Double
    Public ErrorLongRad As Double
    Public ErrorLatDeg As Double
    Public ErrorLongDeg As Double

    ' ***** Energy variables *********************************************************
    ' Force times speed products in the inertial frame of reference
    Public FdragVProductX_I_start, FdragVProductX_I_end As Double
    Public FdragVProductY_I_start, FdragVProductY_I_end As Double
    Public FdragVProductZ_I_start, FdragVProductZ_I_end As Double
    Public FgravVProductX_I_start, FgravVProductX_I_end As Double
    Public FgravVProductY_I_start, FgravVProductY_I_end As Double
    Public FgravVProductZ_I_start, FgravVProductZ_I_end As Double
    ' KE = Sphere's instantaneous kinetic energy
    ' PE = Sphere's instantaneous potential energy
    ' DelGE = Gravitational power x deltaT during one time step
    ' CumGE = Cumulative gravitational power x time
    ' DelDE = Aerodynamic power x deltaT during one time step
    ' CumDE = Cumulative aerodynamic drag x time
    ' TE = System's instantaneous total energy
    ' TE0 = System's initial total energy
    ' EE = Error in system's instantaneous total energy
    Public KE_end As Double
    Public DelGE, CumGE_end As Double
    Public DelDE, CumDE_end As Double
    Public TE_end, TE0 As Double
    Public EE_end As Double

    ' ***** Files ********************************************************************
    ' ThisDirectory = /bin/Debug directory from which this program was executed
    ' OutputFileName = Short name of the output text file, including .txt
    Public ThisDirectory As String = FileSystem.CurDir.ToString
    Public OutputFileName As String = "Exterior_ballistics_results.txt"
    Public FileWriter As System.IO.StreamWriter
End Module
```

Module Procedures

```
Option Strict On
Option Explicit On

Public Module Procedures

    ' List of subroutines:
    '   ConvertGunParametersToSphereStartParameters()
    '   ExecuteOneTimeStep()
    '   WriteResultsToOutputTextFile()
    '   SolveTheMatrixEquation()

    Public Sub ConvertGunParametersToSphereStartParameters()
        ' This subroutines sets up the sphere's initial position and velocity based on
        ' the parameters of the gun, using Equation (13).
        SphereInstRadius_end = LaunchRadius
        SphereInstLongRad_end = LaunchLongRad
```

```
    SphereInstLatRad_end = LaunchLatRad
    SphereInstRadiusDot_end = Speed0 * Math.Sin(Beta0rad)
    SphereInstLongDot_end = _
        Speed0 * Math.Cos(Beta0rad) * Math.Cos(Alpha0rad) / _
        (LaunchRadius * Math.Cos(LaunchLatRad))
    SphereInstLatDot_end = _
        Speed0 * Math.Cos(Beta0rad) * Math.Sin(Alpha0rad) / LaunchRadius
End Sub

Public Sub ExecuteOneTimeStep()
    Dim Temp1, Temp2, Temp3, Temp4, Temp5 As Double
    ' Bring forward all the simulation variables
    SphereInstRadius_start = SphereInstRadius_end
    SphereInstLongRad_start = SphereInstLongRad_end
    SphereInstLatRad_start = SphereInstLatRad_end
    SphereInstRadiusDot_start = SphereInstRadiusDot_end
    SphereInstLongDot_start = SphereInstLongDot_end
    SphereInstLatDot_start = SphereInstLatDot_end
    ' Make a note of some useful trigonometric terms
    Dim CosLam As Double = Math.Cos(SphereInstLatRad_start)
    Dim SinLam As Double = Math.Sin(SphereInstLatRad_start)
    Dim Psi_wt As Double = SphereInstLongRad_start + (Omega * Time)
    Dim CosPsi_wt As Double = Math.Cos(Psi_wt)
    Dim SinPsi_wt As Double = Math.Sin(Psi_wt)
    ' Calculate the components of the sphere's location in the inertial frame,
    ' using Equation (6).
    SphereInstX_I_start = SphereInstRadius_start * CosLam * CosPsi_wt
    SphereInstY_I_start = SphereInstRadius_start * CosLam * SinPsi_wt
    SphereInstZ_I_start = SphereInstRadius_start * SinLam
    ' Calculate the components of the sphere's velocity in the inertial frame,
    ' using Equation (14).
    Temp1 = SphereInstRadiusDot_start * CosLam
    Temp2 = SphereInstRadius_start * SphereInstLatDot_start * SinLam
    Temp3 = SphereInstRadius_start * (SphereInstLongDot_start + Omega) * CosLam
    Temp4 = SphereInstRadiusDot_start * SinLam
    Temp5 = SphereInstRadius_start * SphereInstLatDot_start * CosLam
    SphereInstSpeedX_I_start = _
        (Temp1 * CosPsi_wt) + (-Temp2 * CosPsi_wt) + (-Temp3 * SinPsi_wt)
    SphereInstSpeedY_I_start = _
        (Temp1 * SinPsi_wt) + (-Temp2 * SinPsi_wt) + (Temp3 * CosPsi_wt)
    SphereInstSpeedZ_I_start = Temp4 + Temp5
    SphereInstSpeed_I_start = Math.Sqrt( _
        (SphereInstSpeedX_I_start ^ 2) + _
        (SphereInstSpeedY_I_start ^ 2) +
        (SphereInstSpeedZ_I_start ^ 2))
    ' Calculate the components of the sphere's location w.r.t. the muzzle,
    ' using Equation (9).
    Dim CosPsiPsi0 As Double = Math.Cos(SphereInstLongRad_start - LaunchLongRad)
    Dim SinPsiPsi0 As Double = Math.Sin(SphereInstLongRad_start - LaunchLongRad)
    Dim CosLam0 As Double = Math.Cos(LaunchLatRad)
    Dim SinLam0 As Double = Math.Sin(LaunchLatRad)
    Temp1 = SphereInstRadius_start * CosLam * CosPsiPsi0
    Temp2 = SphereInstRadius_start * SinLam
    Temp3 = SphereInstRadius_start * CosLam * SinPsiPsi0
    SphereInstX_M_start = _
        (-LaunchRadius) + (Temp1 * CosLam0) + (Temp2 * SinLam0)
    SphereInstY_M_start = Temp3
    SphereInstZ_M_start = (-Temp1 * SinLam0) + (Temp2 * CosLam0)
```

```
' Calculate the components of the sphere's velocity w.r.t. the stationary air,
' using Equation (20).
Temp1 = SphereInstRadiusDot_start * CosLam
Temp2 = SphereInstRadius_start * SphereInstLatDot_start * SinLam
Temp3 = SphereInstRadius_start * SphereInstLongDot_start * CosLam
Temp4 = SphereInstRadiusDot_start * SinLam
Temp5 = SphereInstRadius_start * SphereInstLatDot_start * CosLam
SphereInstRelSpeedX_I_start = _
    ((Temp1 - Temp2) * CosPsi_wt) + (-Temp3 * SinPsi_wt)
SphereInstRelSpeedY_I_start = _
    ((Temp1 - Temp2) * SinPsi_wt) + (Temp3 * CosPsi_wt)
SphereInstRelSpeedZ_I_start = Temp4 + Temp5
SphereInstRelSpeed_I_start = Math.Sqrt( _
    (SphereInstRelSpeedX_I_start ^ 2) + _
    (SphereInstRelSpeedY_I_start ^ 2) +
    (SphereInstRelSpeedZ_I_start ^ 2))
' Calculate the aerodynamic drag
SphereInstGeometricAltitude = SphereInstRadius_start - EarthRadius
' Look up state variables of the U.S. Standard Atmosphere
LookupAtmosphericStateVariables( _
    SphereInstGeometricAltitude, _
    LocalTemperature, _
    LocalPressure, _
    LocalDensity, _
    LocalDviscosity, _
    LocalKviscosity, _
    LocalSpeedOfSound, _
    SphereInstGeopotentialAltitude)
' Equation (33)
SphereInstMach = SphereInstRelSpeed_I_start / LocalSpeedOfSound
' Look up the coefficient of drag from the graph
SphereInstCD = LookUpCoefficientOfDrag(SphereInstMach)
' Equation (34)
SphereInstDragF = SphereInstCD * _
    0.5 * LocalDensity * (SphereInstRelSpeed_I_start ^ 2) * Asphere
' Equation (35)
SphereInstRe = SphereInstRelSpeed_I_start * 2 * Rsphere / LocalKviscosity
' Calculate the gravitational force
SphereInstGravF = CalculateMagnitudeOfGravity( _
    Msphere, _
    SphereInstRadius_start)
' Resolve the gravitational force into its components in the inertial frame,
' using Equation (32).
GravFOnSphereX_I = -SphereInstGravF * CosLam * CosPsi_wt
GravFOnSphereY_I = -SphereInstGravF * CosLam * SinPsi_wt
GravFOnSphereZ_I = -SphereInstGravF * SinLam
' Resolve the drag force into its components in the inertial frame,
' using Equation (36).
DragFOnSphereX_I = -SphereInstDragF * _
    SphereInstRelSpeedX_I_start / SphereInstRelSpeed_I_start
DragFOnSphereY_I = -SphereInstDragF * _
    SphereInstRelSpeedY_I_start / SphereInstRelSpeed_I_start
DragFOnSphereZ_I = -SphereInstDragF * _
    SphereInstRelSpeedZ_I_start / SphereInstRelSpeed_I_start
' Construct the total force
TotalFOnSphereX_I = GravFOnSphereX_I + DragFOnSphereX_I
TotalFOnSphereY_I = GravFOnSphereY_I + DragFOnSphereY_I
TotalFOnSphereZ_I = GravFOnSphereZ_I + DragFOnSphereZ_I
```

```
' Solve Newton's Law for the three accelerations
SolveTheMatrixEquation( _
    Time, _
    SphereInstRadius_start, _
    SphereInstLatRad_start, _
    SphereInstLongRad_start, _
    SphereInstRadiusDot_start, _
    SphereInstLatDot_start, _
    SphereInstLongDot_start)
' Integrate once for the velocity using Equation (28)
SphereInstRadiusDot_end = _
    SphereInstRadiusDot_start + (SphereInstRadiusDot2 * deltaT)
SphereInstLatDot_end = _
    SphereInstLatDot_start + (SphereInstLatDot2 * deltaT)
SphereInstLongDot_end = _
    SphereInstLongDot_start + (SphereInstLongDot2 * deltaT)
' Integrate a second time for the location using Equation (29)
SphereInstRadius_end = _
    SphereInstRadius_start + _
    (SphereInstRadiusDot_start * deltaT) + _
    (0.5 * SphereInstRadiusDot2 * deltaT * deltaT)
SphereInstLatRad_end = _
    SphereInstLatRad_start + _
    (SphereInstLatDot_start * deltaT) + _
    (0.5 * SphereInstLatDot2 * deltaT * deltaT)
SphereInstLongRad_end = _
    SphereInstLongRad_start + _
    (SphereInstLongDot_start * deltaT) + _
    (0.5 * SphereInstLongDot2 * deltaT * deltaT)
'
' ////////////////////////////////////////////////////////////////////////////
' ///// Energy calculations at END of time step
' ////////////////////////////////////////////////////////////////////////////
' Make a note of some useful trigonometric terms at the end of the time step
Dim CosLam_end As Double = Math.Cos(SphereInstLatRad_end)
Dim SinLam_end As Double = Math.Sin(SphereInstLatRad_end)
Dim Psi_wt_end As Double = SphereInstLongRad_end + (Omega * Time)
Dim CosPsi_wt_end As Double = Math.Cos(Psi_wt_end)
Dim SinPsi_wt_end As Double = Math.Sin(Psi_wt_end)
' Calculate the components of the sphere's velocity in the inertial frame,
' using Equation (14), at the END of the time step.
Temp1 = SphereInstRadiusDot_end * CosLam_end
Temp2 = SphereInstRadius_end * SphereInstLatDot_end * SinLam_end
Temp3 = SphereInstRadius_end * (SphereInstLongDot_end + Omega) * CosLam_end
Temp4 = SphereInstRadiusDot_end * SinLam_end
Temp5 = SphereInstRadius_end * SphereInstLatDot_end * CosLam_end
SphereInstSpeedX_I_end = _
    (Temp1 * CosPsi_wt_end) + _
    (-Temp2 * CosPsi_wt_end) + _
    (-Temp3 * SinPsi_wt_end)
SphereInstSpeedY_I_end = _
    (Temp1 * SinPsi_wt_end) + _
    (-Temp2 * SinPsi_wt_end) + _
    (Temp3 * CosPsi_wt_end)
SphereInstSpeedZ_I_end = Temp4 + Temp5
SphereInstSpeed_I_end = Math.Sqrt( _
    (SphereInstSpeedX_I_end ^ 2) + _
    (SphereInstSpeedY_I_end ^ 2) +
```

```
    (SphereInstSpeedZ_I_end ^ 2))
' Calculate the Force x Velocity products at the START of the time step
FdragVProductX_I_start = DragFOnSphereX_I * SphereInstSpeedX_I_start
FdragVProductY_I_start = DragFOnSphereY_I * SphereInstSpeedY_I_start
FdragVProductZ_I_start = DragFOnSphereZ_I * SphereInstSpeedZ_I_start
FgravVProductX_I_start = GravFOnSphereX_I * SphereInstSpeedX_I_start
FgravVProductY_I_start = GravFOnSphereY_I * SphereInstSpeedY_I_start
FgravVProductZ_I_start = GravFOnSphereZ_I * SphereInstSpeedZ_I_start
' At start of first time step, initialize the energy variables
If (Time <= (0.5 * deltaT)) Then
    TE0 = 0.5 * Msphere * (SphereInstSpeed_I_start ^ 2)
    CumGE_end = 0
    CumDE_end = 0
    ' Print the first line to the output text file
    FileWriter.Write("Start, , , , , , , ")
    FileWriter.Write(Trim(Str(SphereInstSpeedX_I_start)) & ", ")
    FileWriter.Write(Trim(Str(SphereInstSpeedY_I_start)) & ", ")
    FileWriter.Write(Trim(Str(SphereInstSpeedZ_I_start)) & ", ")
    FileWriter.Write(Trim(Str(SphereInstSpeed_I_start)) & ", ")
    FileWriter.Write(", , , , , , , , , , , , , , , , , , , , ")
    FileWriter.Write(Trim(Str(TE0)) & ", ")
    FileWriter.Write(Trim(Str(0)) & ", ")
    FileWriter.Write(Trim(Str(0)) & ", ")
    FileWriter.Write(Trim(Str(TE0)) & ", ")
    FileWriter.WriteLine(Trim(Str(0)))
End If
' Calculate the sphere's kinetic energy at the END of the time step
KE_end = 0.5 * Msphere * (SphereInstSpeed_I_end ^ 2)
' Calculate the energy used during this time step to overcome gravity
DelGE = -deltaT * _
    (FgravVProductX_I_start + _
     FgravVProductY_I_start + _
     FgravVProductZ_I_start)
CumGE_end = CumGE_end + DelGE
' Calculate the energy used during this time step to overcome drag
DelDE = -deltaT * _
    (FdragVProductX_I_start + _
     FdragVProductY_I_start + _
     FdragVProductZ_I_start)
CumDE_end = CumDE_end + DelDE
' Calculate the total energy and energy error at the END of the time step
TE_end = KE_end + CumGE_end + CumDE_end
EE_end = 100 * (TE_end - TE0) / TE0
' ////////////////////////////////////////////////////////////////////////////
' ///// End of energy calculations
' ////////////////////////////////////////////////////////////////////////////
'
' Check to see if it is time to write results to the output text file
WriteCounter = WriteCounter + 1
If (WriteCounter >= WriteInterval) Then
    WriteCounter = 0
    WriteResultsToOutputTextFile()
End If
' Write update to screen
ScreenCounter = ScreenCounter + 1
If (ScreenCounter >= ScreenInterval) Then
    Dim lTemp As String
    Dim lds As String
```

```
            ScreenCounter = 0
            lds = "Time(s)=" & Trim(Str(Time))
            lTemp = Trim(Str(SphereInstRadius_end - EarthRadius))
            lds = lds & " Alt(m)=" & lTemp
            lTemp = Trim(Str(SphereInstLatRad_end * 180 / Math.PI))
            lds = lds & " Lat(deg)=" & lTemp
            lTemp = Trim(Str(SphereInstLongRad_end * 180 / Math.PI))
            lds = lds & " Long(deg)=" & lTemp
            lTemp = Trim(Str(SphereInstRelSpeed_I_start))
            lds = lds & " RelSpeed(m/s)=" & lTemp
            Form1.labelResults.Text = _
                Strings.Right(Form1.labelResults.Text & lds & vbCrLf, 2000)
            Form1.labelResults.Refresh()
        End If
    End Sub

    Public Sub WriteResultsToOutputTextFile()
        Dim lTemp As String
        FileWriter.Write(Trim(Str(Time)) & ", ")
        lTemp = Trim(Str(SphereInstRadius_end - EarthRadius))
        FileWriter.Write(lTemp & ", ")
        lTemp = Trim(Str(SphereInstLatRad_end * 180 / Math.PI))
        FileWriter.Write(lTemp & ", ")
        lTemp = Trim(Str(SphereInstLongRad_end * 180 / Math.PI))
        FileWriter.Write(lTemp & ", ")
        FileWriter.Write(Trim(Str(SphereInstX_M_start)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstY_M_start)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstZ_M_start)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstSpeedX_I_start)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstSpeedY_I_start)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstSpeedZ_I_start)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstSpeed_I_start)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstRelSpeedX_I_start)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstRelSpeedY_I_start)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstRelSpeedZ_I_start)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstRelSpeed_I_start)) & ", ")
        FileWriter.Write(Trim(Str(LocalSpeedOfSound)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstMach)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstCD)) & ", ")
        FileWriter.Write(Trim(Str(LocalDensity)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstDragF)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstGravF)) & ", ")
        FileWriter.Write(Trim(Str(GravFOnSphereX_I)) & ", ")
        FileWriter.Write(Trim(Str(GravFOnSphereY_I)) & ", ")
        FileWriter.Write(Trim(Str(GravFOnSphereZ_I)) & ", ")
        FileWriter.Write(Trim(Str(DragFOnSphereX_I)) & ", ")
        FileWriter.Write(Trim(Str(DragFOnSphereY_I)) & ", ")
        FileWriter.Write(Trim(Str(DragFOnSphereZ_I)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstRadiusDot2)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstLatDot2)) & ", ")
        FileWriter.Write(Trim(Str(SphereInstLongDot2)) & ", ")
        FileWriter.Write(Trim(Str(KE_end)) & ", ")
        FileWriter.Write(Trim(Str(CumGE_end)) & ", ")
        FileWriter.Write(Trim(Str(CumDE_end)) & ", ")
        FileWriter.Write(Trim(Str(TE_end)) & ", ")
        FileWriter.WriteLine(Trim(Str(EE_end)))
    End Sub
```

```
    Public Sub SolveTheMatrixEquation( _
        ByVal T As Double, _
        ByVal Rad As Double, ByVal Lam As Double, ByVal Psi As Double, _
        ByVal RadDot As Double, ByVal LamDot As Double, ByVal PsiDot As Double)
        ' This subroutine takes as inputs the three location variables, their first
        ' derivatives, and time.  It then solves the matrix equation.
        Dim A34(3, 4) As Double
        Dim B(3) As Double
        ' Build the last column of the A-matrix
        Dim Temp1, Temp2, Temp3, Temp4, Temp5, Temp6 As Double
        Dim CosLam As Double = Math.Cos(Lam)
        Dim SinLam As Double = Math.Sin(Lam)
        Dim TanLam As Double = SinLam / CosLam
        Dim CosPsi As Double = Math.Cos(Psi)
        Dim SinPsi As Double = Math.Sin(Psi)
        Dim CosPsi_wT As Double = Math.Cos(Psi + (Omega * T))
        Dim SinPsi_wT As Double = Math.Sin(Psi + (Omega * T))
        Dim TanPsi_wT As Double = SinPsi_wT / CosPsi_wT
        Dim PsiDot_w As Double = PsiDot + Omega
        Temp1 = -2 * RadDot * LamDot * SinLam * CosPsi_wT
        Temp2 = -2 * RadDot * PsiDot_w * CosLam * SinPsi_wT
        Temp3 = -Rad * LamDot * LamDot * CosLam * CosPsi_wT
        Temp4 = Rad * LamDot * PsiDot_w * SinLam * SinPsi_wT
        Temp5 = Rad * LamDot * PsiDot_w * SinLam * SinPsi_wT
        Temp6 = -Rad * PsiDot_w * PsiDot_w * CosLam * CosPsi_wT
        A34(1, 4) = Temp1 + Temp2 + Temp3 + Temp4 + Temp5 + Temp6
        Temp1 = -2 * RadDot * LamDot * SinLam * SinPsi_wT
        Temp2 = 2 * RadDot * PsiDot_w * CosLam * CosPsi_wT
        Temp3 = -Rad * LamDot * LamDot * CosLam * SinPsi_wT
        Temp4 = -Rad * LamDot * PsiDot_w * SinLam * CosPsi_wT
        Temp5 = -Rad * LamDot * PsiDot_w * SinLam * CosPsi_wT
        Temp6 = -Rad * PsiDot_w * PsiDot_w * CosLam * SinPsi_wT
        A34(2, 4) = Temp1 + Temp2 + Temp3 + Temp4 + Temp5 + Temp6
        Temp1 = 2 * RadDot * LamDot * CosLam
        Temp2 = -Rad * LamDot * LamDot * SinLam
        A34(3, 4) = Temp1 + Temp2
        ' Build the B-vector
        B(1) = (TotalFOnSphereX_I / Msphere) - A34(1, 4)
        B(2) = (TotalFOnSphereY_I / Msphere) - A34(2, 4)
        B(3) = (TotalFOnSphereZ_I / Msphere) - A34(3, 4)
        ' Solve for LongDot2 using (C11)
        Temp1 = (B(2) * CosPsi_wT) - (B(1) * SinPsi_wT)
        SphereInstLongDot2 = Temp1 / (Rad * CosLam)
        ' Solve for LatDot2 using (C12)
        Temp1 = _
            (B(3) * CosLam) + _
            (-B(1) * SinLam / CosPsi_wT) + _
            (-Rad * SphereInstLongDot2 * SinLam * CosLam * TanPsi_wT)
        SphereInstLatDot2 = Temp1 / Rad
        ' Solve for RDot2 using (C13)
        Temp1 = B(1) / (CosLam * CosPsi_wT)
        SphereInstRadiusDot2 = _
            Temp1 + _
            (Rad * SphereInstLatDot2 * TanLam) + _
            (Rad * SphereInstLongDot2 * TanPsi_wT)
    End Sub
End Module
```

Module MagnitudeOfGravity

```
Option Strict On
Option Explicit On

' List of subroutines
'   CalculateMagnitudeOfGravity(lMass, lDistance) as Force

Public Module MagnitudeOfGravity

    ' ***** Physical constants ************************************************
    ' EarthMass = Mass of Earth, in kilograms
    ' GravConstant = Gravitational constant, in Nm^2/kg^2
    Private EarthMass As Double = Val("5.97219E+24")
    Private GravConstant As Double = Val("6.673E-11")

    Public Function CalculateMagnitudeOfGravity( _
        ByVal lMass As Double, ByVal lDistance As Double) As Double
        ' This function calculates the magnitude of the Earth's gravitational pull
        ' on a mass lMass at distance lDistance from the Earth's center.  lMass is
        ' to be stated in kilograms and lAltitude is to be stated in meters.
        CalculateMagnitudeOfGravity = _
            GravConstant * EarthMass * lMass / (lDistance ^ 2)
    End Function
End Module
```

Module USStandardAtmosphere

```
Option Strict On
Option Explicit On

' List of subroutines
'   InitializeAtmosphereLayers()
'   LookupAtmosphericStateVariables(returns ByRef)

Public Module USStandardAtmosphere

    ' ***** Definition of variables at the boundary altitudes *************************
    Private AltAtBotOfLayer(6) As Double
    Private AltAtTopOfLayer(6) As Double
    Private TempAtBotOfLayer(6) As Double
    Private PresAtBotOfLayer(6) As Double

    ' ***** Definition of lapse rates *************************************************
    Private LapseRateInLayer(6) As Double
    Private LapseExponentInLayer(6) As Double

    ' ***** Physical constants ********************************************************
    ' g0 = Local gravitational constant, in m/s^2
    ' Rair = Ideal Gas Constant for air, in J/kg-degK
    ' EarthRadius = Radius of the Earth, to mean sea level, in meters
    ' DViscCoef1 = Dynamic viscosity coefficient #1, units to suit
    ' DViscCoef2 = Dynamic viscosity coefficient #2, in degK
    ' GAMMAair = Ratio of specific heats for air, Cp/Cv
    Private g0 As Double = 9.80665
    Private Rair As Double = 287.053
    Private EarthRadius As Double = Val("6356766")
    Private DViscCoef1 As Double = Val("1.458E-6")
    Private DViscCoef2 As Double = 110.4
```

```
Private GAMMAair As Double = 1.4

Public Sub InitializeAtmosphereLayers()
    ' Set sea level conditions
    TempAtBotOfLayer(0) = 15 + 273.15
    PresAtBotOfLayer(0) = 101325
    ' Set altitude of boundaries between layers, in meters
    AltAtBotOfLayer(0) = 0 : AltAtTopOfLayer(0) = 11000
    AltAtBotOfLayer(1) = 11000 : AltAtTopOfLayer(1) = 20000
    AltAtBotOfLayer(2) = 20000 : AltAtTopOfLayer(2) = 32000
    AltAtBotOfLayer(3) = 32000 : AltAtTopOfLayer(3) = 47000
    AltAtBotOfLayer(4) = 47000 : AltAtTopOfLayer(4) = 51000
    AltAtBotOfLayer(5) = 51000 : AltAtTopOfLayer(5) = 71000
    AltAtBotOfLayer(6) = 71000 : AltAtTopOfLayer(6) = 85000
    ' Set lapse rates, in degrees Kelvin per meter
    LapseRateInLayer(0) = -6.5 / 1000
    LapseRateInLayer(1) = 0
    LapseRateInLayer(2) = 1 / 1000
    LapseRateInLayer(3) = 2.8 / 1000
    LapseRateInLayer(4) = 0
    LapseRateInLayer(5) = -2.8 / 1000
    LapseRateInLayer(6) = -2 / 1000
    ' Set lapse "exponents" and temperatures at the boundary altitudes
    For I As Int32 = 0 To 6 Step 1
        ' Calculate the lapse "exponent" in this layer
        If (LapseRateInLayer(I) = 0) Then
            LapseExponentInLayer(I) = -g0 / (Rair * TempAtBotOfLayer(I))
        Else
            LapseExponentInLayer(I) = -g0 / (Rair * LapseRateInLayer(I))
        End If
        ' Calculate the temperature at the top of layers #0 through #5
        If (I <= 5) Then
            Dim lThicknessOfLayer As Double = _
                AltAtTopOfLayer(I) - AltAtBotOfLayer(I)
            TempAtBotOfLayer(I + 1) = TempAtBotOfLayer(I) + _
                (lThicknessOfLayer * LapseRateInLayer(I))
        End If
    Next I
    ' Set pressures at the boundary altitudes
    For I As Int32 = 0 To 5 Step 1
        Dim lThicknessOfLayer As Double = _
            AltAtTopOfLayer(I) - AltAtBotOfLayer(I)
        Dim lTemp As Double
        If (LapseRateInLayer(I) = 0) Then
            lTemp = LapseExponentInLayer(I) * lThicknessOfLayer
            PresAtBotOfLayer(I + 1) = _
                PresAtBotOfLayer(I) * Math.Exp(lTemp)
        Else
            lTemp = TempAtBotOfLayer(I + 1) / TempAtBotOfLayer(I)
            PresAtBotOfLayer(I + 1) = _
                PresAtBotOfLayer(I) * (lTemp ^ LapseExponentInLayer(I))
        End If
    Next I
End Sub

Public Sub LookupAtmosphericStateVariables( _
    ByVal lGeometricAlt As Double, _
    ByRef lLocalTemp As Double, _
```

```
        ByRef lLocalPres As Double, _
        ByRef lLocalDens As Double, _
        ByRef lLocalDVisc As Double, _
        ByRef lLocalKVisc As Double, _
        ByRef lLocalSpeedOfSound As Double, _
        ByRef lGeopotentialAlt As Double)
        Dim lLayer As Int32             ' Layer which contains lGeometricAlt
        Dim lDifferenceInAlt As Double  ' Distance to the bottom of its layer
        Dim lTemp As Double             ' A temporary variable
        ' Convert the geometric altitude to a geopotential altitude
        lGeopotentialAlt = EarthRadius * _
            (1 - (EarthRadius / (EarthRadius + lGeometricAlt)))
        ' Check that the given lGeometricAlt is within the range of data
        If (lGeopotentialAlt < 0) Then
            MsgBox( _
                "The given altitude of " & Trim(Str(lGeometricAlt)) & _
                " meters is below sea level." & vbCrLf & _
                "No calculations will be done.")
            Exit Sub
        End If
        If (lGeopotentialAlt > AltAtTopOfLayer(6)) Then
            MsgBox( _
                "The given altitude of " & Trim(Str(lGeometricAlt)) & _
                " meters lies above Layer #6 in the model atmosphere." & vbCrLf & _
                "No calculations will be done.")
            Exit Sub
        End If
        ' Determine which layer the lGeopotentialAlt lies within
        For I As Int32 = 0 To 6 Step 1
            If (lGeopotentialAlt <= AltAtTopOfLayer(I)) Then
                lLayer = I
                Exit For
            End If
        Next I
        ' Calculate the temperature at this lGeopotentialAlt
        lDifferenceInAlt = lGeopotentialAlt - AltAtBotOfLayer(lLayer)
        lLocalTemp = _
            TempAtBotOfLayer(lLayer) + (lDifferenceInAlt * LapseRateInLayer(lLayer))
        ' Calculate the pressure at this lGeopotentialAlt
        If (LapseRateInLayer(lLayer) = 0) Then
            lTemp = LapseExponentInLayer(lLayer) * lDifferenceInAlt
            lLocalPres = _
                PresAtBotOfLayer(lLayer) * Math.Exp(lTemp)
        Else
            lTemp = lLocalTemp / TempAtBotOfLayer(lLayer)
            lLocalPres = _
                PresAtBotOfLayer(lLayer) * (lTemp ^ LapseExponentInLayer(lLayer))
        End If
        ' Calculate the density at this lGeopotentialAl
        lLocalDens = lLocalPres / (Rair * lLocalTemp)
        ' Calculate the dynamic viscosity at this lGeopotentialAlt
        lLocalDVisc = DViscCoef1 * (lLocalTemp ^ 1.5) / (lLocalTemp + DViscCoef2)
        ' Calculate the kinematic viscosity at this lGeopotentialAlt
        lLocalKVisc = lLocalDVisc / lLocalDens
        ' Calculate the local speed of sound
        lLocalSpeedOfSound = Math.Sqrt(GAMMAair * lLocalPres / lLocalDens)
    End Sub
End Module
```

Module CoefficientOfDrag

```
Option Strict On
Option Explicit On

' List of subroutines
'   InitializeCDLookupTable()
'   LookUpCoefficientOfDrag(lMach) as CD

Public Module CoefficientOfDrag

    ' ***** Coefficient of drag data ************************************************
    ' CDLookupTable(51, 51) = Mach number versus CD at 51 points
    Private CDLookupTable(51, 51) As Double

    Public Sub InitializeCDLookupTable()
        CDLookupTable(1, 1) = 0 : CDLookupTable(1, 2) = 0.1076
        CDLookupTable(2, 1) = 0.2036 : CDLookupTable(2, 2) = 0.1211
        CDLookupTable(3, 1) = 0.3937 : CDLookupTable(3, 2) = 0.1426
        CDLookupTable(4, 1) = 0.5023 : CDLookupTable(4, 2) = 0.1722
        CDLookupTable(5, 1) = 0.5566 : CDLookupTable(5, 2) = 0.2179
        CDLookupTable(6, 1) = 0.5837 : CDLookupTable(6, 2) = 0.2664
        CDLookupTable(7, 1) = 0.6109 : CDLookupTable(7, 2) = 0.3094
        CDLookupTable(8, 1) = 0.638 : CDLookupTable(8, 2) = 0.374
        CDLookupTable(9, 1) = 0.6516 : CDLookupTable(9, 2) = 0.4197
        CDLookupTable(10, 1) = 0.6787 : CDLookupTable(10, 2) = 0.4682
        CDLookupTable(11, 1) = 0.7059 : CDLookupTable(11, 2) = 0.5139
        CDLookupTable(12, 1) = 0.7602 : CDLookupTable(12, 2) = 0.5839
        CDLookupTable(13, 1) = 0.7873 : CDLookupTable(13, 2) = 0.6269
        CDLookupTable(14, 1) = 0.8145 : CDLookupTable(14, 2) = 0.6619
        CDLookupTable(15, 1) = 0.8552 : CDLookupTable(15, 2) = 0.7372
        CDLookupTable(16, 1) = 0.8824 : CDLookupTable(16, 2) = 0.7857
        CDLookupTable(17, 1) = 0.9367 : CDLookupTable(17, 2) = 0.8368
        CDLookupTable(18, 1) = 0.9774 : CDLookupTable(18, 2) = 0.8879
        CDLookupTable(19, 1) = 1.0995 : CDLookupTable(19, 2) = 0.9471
        CDLookupTable(20, 1) = 1.2624 : CDLookupTable(20, 2) = 0.9821
        CDLookupTable(21, 1) = 1.4525 : CDLookupTable(21, 2) = 1.0009
        CDLookupTable(22, 1) = 1.6968 : CDLookupTable(22, 2) = 1.0117
        CDLookupTable(23, 1) = 1.9819 : CDLookupTable(23, 2) = 1.0036
        CDLookupTable(24, 1) = 2.1448 : CDLookupTable(24, 2) = 0.9928
        CDLookupTable(25, 1) = 2.3484 : CDLookupTable(25, 2) = 0.9767
        CDLookupTable(26, 1) = 2.552 : CDLookupTable(26, 2) = 0.9659
        CDLookupTable(27, 1) = 2.7828 : CDLookupTable(27, 2) = 0.9552
        CDLookupTable(28, 1) = 3.1222 : CDLookupTable(28, 2) = 0.9498
        CDLookupTable(29, 1) = 3.3394 : CDLookupTable(29, 2) = 0.9498
        CDLookupTable(30, 1) = 3.5701 : CDLookupTable(30, 2) = 0.9471
        CDLookupTable(31, 1) = 3.7602 : CDLookupTable(31, 2) = 0.9444
        CDLookupTable(32, 1) = 4.0452 : CDLookupTable(32, 2) = 0.9444
        CDLookupTable(33, 1) = 4.276 : CDLookupTable(33, 2) = 0.9417
        CDLookupTable(34, 1) = 4.5475 : CDLookupTable(34, 2) = 0.9417
        CDLookupTable(35, 1) = 4.7919 : CDLookupTable(35, 2) = 0.9417
        CDLookupTable(36, 1) = 5.1041 : CDLookupTable(36, 2) = 0.9363
        CDLookupTable(37, 1) = 5.4163 : CDLookupTable(37, 2) = 0.9336
        CDLookupTable(38, 1) = 5.7421 : CDLookupTable(38, 2) = 0.9309
        CDLookupTable(39, 1) = 6.095 : CDLookupTable(39, 2) = 0.9283
        CDLookupTable(40, 1) = 6.4344 : CDLookupTable(40, 2) = 0.9256
        CDLookupTable(41, 1) = 6.8009 : CDLookupTable(41, 2) = 0.9229
        CDLookupTable(42, 1) = 7.1538 : CDLookupTable(42, 2) = 0.9202
        CDLookupTable(43, 1) = 7.6018 : CDLookupTable(43, 2) = 0.9202
```

```
        CDLookupTable(44, 1) = 8.0226 : CDLookupTable(44, 2) = 0.9175
        CDLookupTable(45, 1) = 8.4977 : CDLookupTable(45, 2) = 0.9175
        CDLookupTable(46, 1) = 8.9186 : CDLookupTable(46, 2) = 0.9175
        CDLookupTable(47, 1) = 9.2579 : CDLookupTable(47, 2) = 0.9175
        CDLookupTable(48, 1) = 9.5837 : CDLookupTable(48, 2) = 0.9148
        CDLookupTable(49, 1) = 9.9638 : CDLookupTable(49, 2) = 0.9148
        CDLookupTable(50, 1) = 10.2896 : CDLookupTable(50, 2) = 0.9148
        CDLookupTable(51, 1) = 10.5068 : CDLookupTable(51, 2) = 0.9148
    End Sub

    Public Function LookUpCoefficientOfDrag( _
        ByVal lMach As Double) As Double
        ' This function interpolates in the CDLookupTable(,) to calculate and return
        ' the coefficient of drag for the given Mach number lMach.
        ' Check for extreme values
        If (lMach <= 0) Then
            LookUpCoefficientOfDrag = CDLookupTable(1, 2)
            Exit Function
        End If
        If (lMach >= CDLookupTable(51, 1)) Then
            LookUpCoefficientOfDrag = CDLookupTable(51, 2)
            Exit Function
        End If
        ' Determine the nearest index
        Dim lIndex As Int32
        For I As Int32 = 1 To 50 Step 1
            If ((lMach > CDLookupTable(I, 1)) And _
                (lMach <= CDLookupTable(I + 1, 1))) Then
                lIndex = I
                Exit For
            End If
        Next I
        ' Determine the slope
        Dim lSlope As Double
        lSlope = _
            (CDLookupTable(lIndex + 1, 2) - CDLookupTable(lIndex, 2)) / _
            (CDLookupTable(lIndex + 1, 1) - CDLookupTable(lIndex, 1))
        ' Determine the value
        Dim ldeltaX As Double
        Dim ldeltaY As Double
        ldeltaX = lMach - CDLookupTable(lIndex, 1)
        ldeltaY = lSlope * ldeltaX
        LookUpCoefficientOfDrag = CDLookupTable(lIndex, 2) + ldeltaY
    End Function
End Module
```