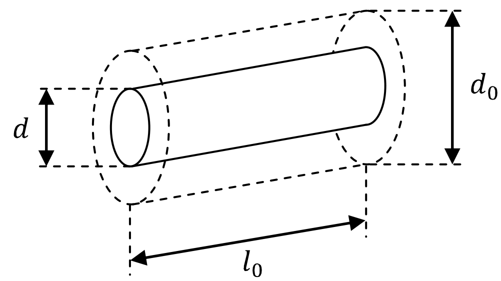
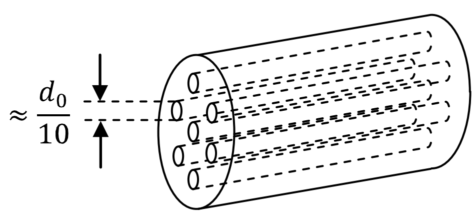
Interior ballistics: The effect of holes in the web

In an earlier paper, titled *Interior ballistics of a large naval gun or artillery piece*, I assumed that the grains of propellant were solid cylinders with original unburned diameter d_0 and length l_0. Burning takes place from the outside inwards, so the surface area being burned at any instant of time is $\pi d l_0$. This surface area decreases as the grain is consumed.



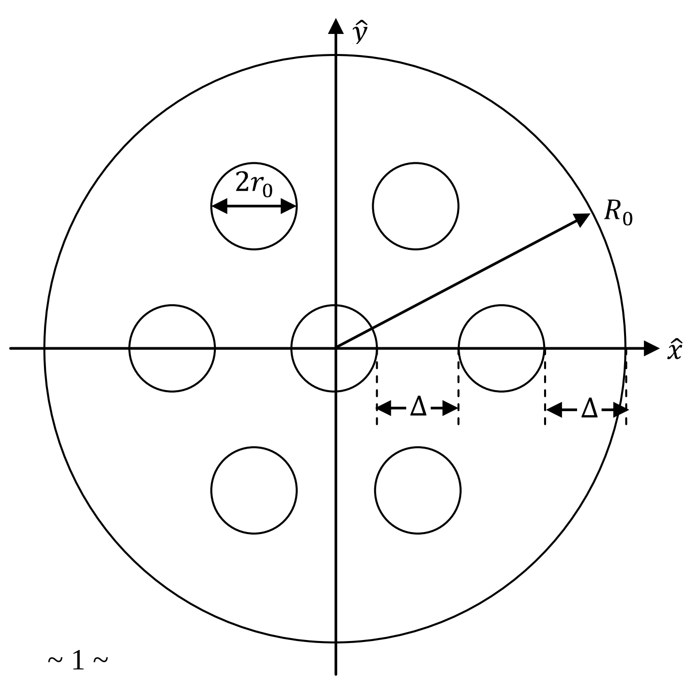
The effective rate at which propellant is consumed (at comparable pressures) can be changed by adding holes to the web. The following figure shows the same grain with seven holes. This particular configuration of holes was very common in the British, American and Japanese navies. The German navy preferred tubular grains, with a single hole at the central axis of the grain.



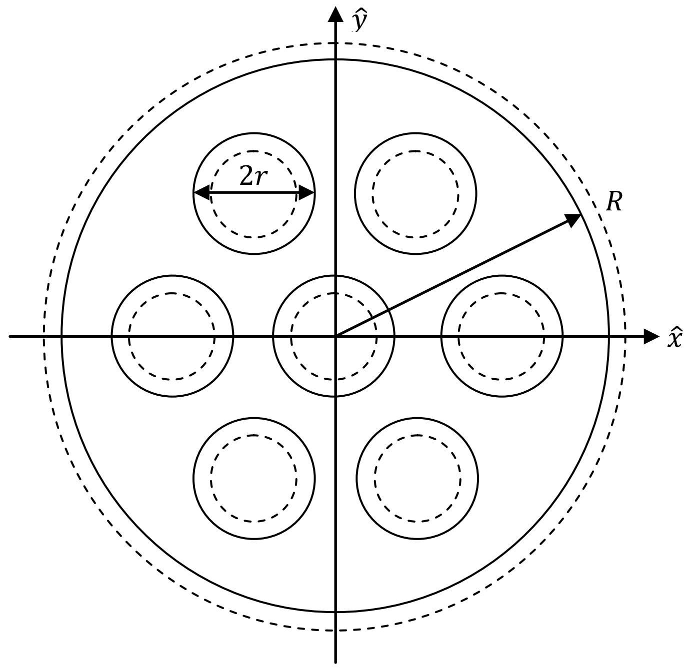
The holes burn from the inside outwards, so their burning surface increases as the grain is consumed. This offsets the decrease with time in the outer surface area of the cylinder, giving a more uniform rate of consumption to the whole. On large grains, the holes were typically one-tenth the diameter of the grain. On small grains, the holes were mere pinholes.

The volume of a propellant grain during the initial phase of burning

The figure to the right shows a cross-section of a grain of propellant before it starts burning. All seven holes have the same radius, r_0 say, before burning starts. The six small holes in the middle ring are equally spaced around the grain. It is important that the two distances Δ be equal. As we will see, setting these distances equal will cause the grain to remain intact as it burns until the burning surfaces make contact with each other and the grain breaks into 12 pieces, each being roughly a triangular cylinder. Note that the distance Δ is not necessaily related to the size of the interior holes. It is possible to have smaller holes and larger Δ's, or vice versa.



The situation after the propellant has burned for a time is shown in this figure. The solid lines are the burning surfaces; the dotted lines are the original surfaces.



During this period of burning, the radius of the unburned grain decreased from R_0 to R and the radii of the holes, including the substance burned, increased from r_0 to r. If we assume that all surfaces burned at the same rate (an important assumption), then the change in radii must be the same:

$$R_0 - R = r - r_0 \quad (1)$$

We can calculate the volume of unburned propellant at the start and end of this period of burning. The calculation is easier if we ignore burning at the end faces of the grain, just like we did in the original analysis. If so, then the length of the grain remains constant at l_0.

The volume of the original propellant is:

$$\begin{aligned} Volume_{start} &= (Area_{outer\ circle} - 7Area_{inner\ hole})l_0 \\ &= (\pi R_0^2 - 7\pi r_0^2)l_0 \qquad (2) \end{aligned}$$

The volume of unburned propellant at the end is:

$$\begin{aligned} Volume_{end} &= (Area_{outer\ circle} - 7Area_{inner\ hole})l_0 \\ &= (\pi R^2 - 7\pi r^2)l_0 \qquad (3) \end{aligned}$$

The decrease in volume (an algebraically positive number) during this period of burning is:

$$\begin{aligned} Decrease\ in\ Volume &= Volume_{start} - Volume_{end} \\ &= \pi[(R_0^2 - R^2) + 7(r^2 - r_0^2)]l_0 \quad (4) \end{aligned}$$

Assume that this period of burning has a duration ΔT and that the burning rate B throughout the period is constant. Since the burning rate is the rate at which the burning surface eats down into the unburned surface, we can say that:

$$B\Delta T = R_0 - R = r - r_0 \quad (5)$$

This can be re-arranged to give:

$$\left.\begin{aligned} R &= R_0 - B\Delta T \\ r &= r_0 + B\Delta T \end{aligned}\right\} \quad (6)$$

We can substitute these expressions into Equation (4) to express the decrease in volume in terms of quantities which exist at the start of the period of burning. We get:

$$\begin{aligned}
\textit{Decrease in Volume} \quad &= \pi\{[R_0^2 - (R_0 - B\Delta T)^2] + 7[(r_0 + B\Delta T)^2 - r_0^2]\}l_0 \\
&= \pi[(2R_0 B\Delta T - B^2\Delta T^2) + 7(2r_0 B\Delta T + B^2\Delta T^2)]l_0 \\
&= \pi B\Delta T[(2R_0 - B\Delta T) + 7(2r_0 + B\Delta T)]l_0 \\
&= \pi B\Delta T(2R_0 + 14r_0 + 6B\Delta T)l_0 \qquad (7)
\end{aligned}$$

The mass of propellant burned during this period of burning is this volume multiplied by the density of the propellant, thus:

$$\textit{Mass burned} = \pi\rho_{charge} B\Delta T(2R_0 + 14r_0 + 6B\Delta T)l_0 \qquad (8)$$

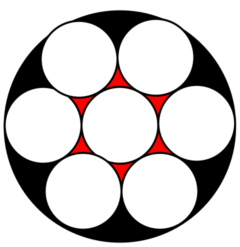
When does the initial phase of burning come to an end?

It will end when the radius of the outer surface of the grain (R) is equal to three times the radius of the interior holes (r).

$$R_{End\ of\ Phase1} = 3r_{End\ of\ Phase1} \qquad (9)$$

Equation (1) holds at all times during the initial phase, including the instant at the end of this phase. Substituting Equation (9) into Equation (1) gives:

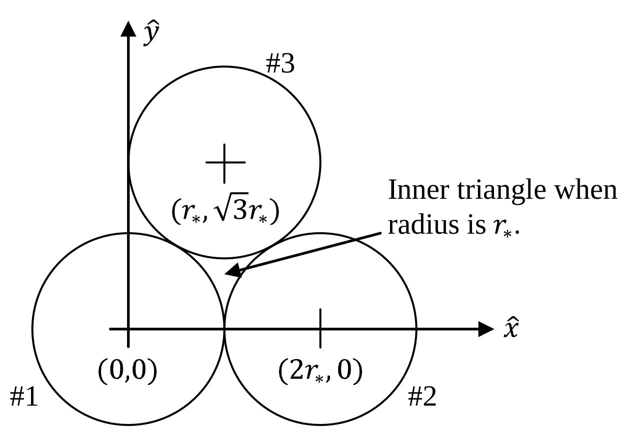
$$\begin{aligned}
R_0 - 3r_{End\ of\ Phase1} &= r_{End\ of\ Phase1} - r_0 \\
\rightarrow \qquad 4r_{End\ of\ Phase1} &= R_0 + r_0 \\
\rightarrow \qquad r_{End\ of\ Phase1} &= \frac{R_0 + r_0}{4} \qquad (10)
\end{aligned}$$



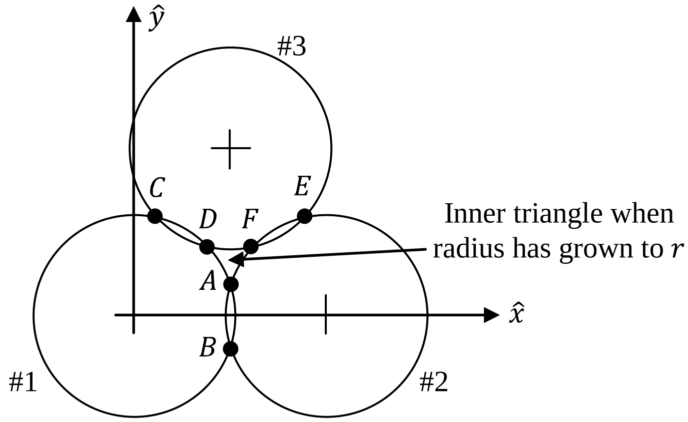
When the initial phase of burning is complete, the holes will have grown to a size where they touch each other and the grain falls into pieces. There will be 12 pieces. The figure at the left shows there will be six inner "triangles" (rendered in red) and six outer "triangles (rendered in black). In the second phase of burning, all 12 of these "triangles" will burn. But, the red "triangles are smaller, and will be consumed first. After they have been consumed, there will be a third phase of burning, in which only the remnants of the six outer "triangles" will be left burning.

The volume of one of the inner "triangles" during the second phase of burning

Consider three circles with radius r_* which are mutually tangent. The following figure shows the arrangement. Think of r_* as the radius of the holes in the web when the burn depths have just reached the point when a grain of propellant ceases to be a single entity and separates into 12 triangular prisms. Therefore, r_* will be equal to $r_{End\ of\ Phase1}$ as calculated in Equation (10). The orientation of the three circles is arbitrary, so I have laid them out with two of them side-by-side horizontally, with the origin of this two-dimensional plane at the center of the hole on the left.



Now consider the circles (holes) at some later time, when their radii have increased from r_* to r. There are now six points of intersection among the three circles. Three of them – A, D and F – are the three vertices of the inner triangle, but we are going to need all six points to calculate the surface area.



.

The equations of these three circles are:

$$\left.\begin{array}{ll} \#1: & x^2 + y^2 = r^2 \\ \#2: & (x - 2r_*)^2 + y^2 = r^2 \\ \#3: & (x - r_*)^2 + \left(y - \sqrt{3}r_*\right)^2 = r^2 \end{array}\right\} \quad (11)$$

<u>The two points of intersection between circles #1 and #2</u>

Setting x and y common to circles #1 and #2 gives:

$$\begin{array}{rl} & x^2 + y^2 = (x - 2r_*)^2 + y^2 \\ \rightarrow & 0 = -4r_* x + 4r_*^2 \\ \rightarrow & x = r_* \end{array}$$

Now substituting this value of x into the equation for circle #1 gives:

$$\begin{array}{rl} & r_*^2 + y^2 = r^2 \\ \rightarrow & y = \pm\sqrt{r^2 - r_*^2} \end{array}$$

and the co-ordinates of the two points of intersection are:

$$\left.\begin{array}{ll} A: & (x_A, y_A) = \left(r_* \ ; \ +\sqrt{r^2 - r_*^2}\right) \\ B: & (x_B, y_B) = \left(r_* \ ; \ -\sqrt{r^2 - r_*^2}\right) \end{array}\right\} \quad (12)$$

<u>The two points of intersection between circles #1 and #3</u>

Setting x and y common to circles #1 and #3 gives:

$$\begin{array}{rl} & x^2 + y^2 = (x - r_*)^2 + \left(y - \sqrt{3}r_*\right)^2 \\ \rightarrow & x = 2r_* - \sqrt{3}y \end{array}$$

Now substituting this value of x into the equation for circle #1 gives:

$$
\begin{aligned}
&\left(2r_* - \sqrt{3}y\right)^2 + y^2 = r^2 \\
\rightarrow \quad & 4y^2 - 4\sqrt{3}r_* y + (4r_*^2 - r^2) = 0 \\
\rightarrow \quad & y = \frac{4\sqrt{3}r_* \pm \sqrt{48r_*^2 - 16(4r_*^2 - r^2)}}{8} \\
\rightarrow \quad & y = \frac{\sqrt{3}r_* \pm \sqrt{r^2 - r_*^2}}{2}
\end{aligned}
$$

and the co-ordinates of the two points of intersection are:

$$
\left.
\begin{aligned}
C: \quad (x_C, y_C) &= \left(\frac{r_* - \sqrt{3r^2 - 3r_*^2}}{2} \; ; \; \frac{\sqrt{3}r_* + \sqrt{r^2 - r_*^2}}{2}\right) \\
D: \quad (x_D, y_D) &= \left(\frac{r_* + \sqrt{3r^2 - 3r_*^2}}{2} \; ; \; \frac{\sqrt{3}r_* - \sqrt{r^2 - r_*^2}}{2}\right)
\end{aligned}
\right\} \qquad (13)
$$

<u>The two points of intersection between circles #2 and #3</u>

Setting x and y common to circles #2 and #3 gives:

$$
\begin{aligned}
&(x - 2r_*)^2 + y^2 = (x - r_*)^2 + \left(y - \sqrt{3}r_*\right)^2 \\
\rightarrow \quad & -2r_* x = -2\sqrt{3}r_* y \\
\rightarrow \quad & x = \sqrt{3}y
\end{aligned}
$$

Now substituting this value of x into the equation for circle #2 gives:

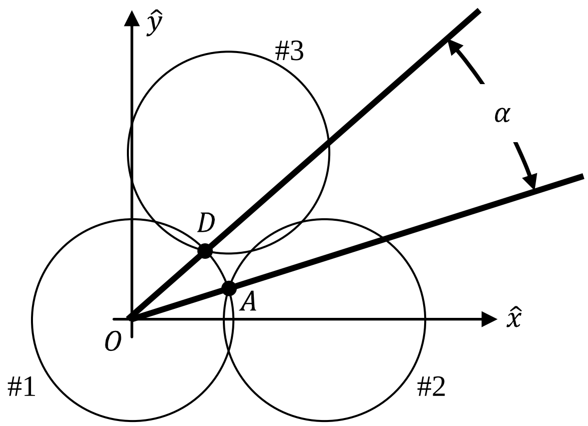
$$
\begin{aligned}
&\left(\sqrt{3}y - 2r_*\right)^2 + y^2 = r^2 \\
\rightarrow \quad & 4y^2 - 4\sqrt{3}r_* y + (4r_*^2 - r^2) = 0 \\
\rightarrow \quad & y = \frac{4\sqrt{3}r_* \pm \sqrt{48r_*^2 - 16(4r_*^2 - r^2)}}{8} \\
\rightarrow \quad & y = \frac{\sqrt{3}r_* \pm \sqrt{r^2 - r_*^2}}{2}
\end{aligned}
$$

and the co-ordinates of the two points of intersection are:

$$
\left.
\begin{aligned}
E: \quad (x_E, y_E) &= \left(\frac{3r_* + \sqrt{3r^2 - 3r_*^2}}{2} \; ; \; \frac{\sqrt{3}r_* + \sqrt{r^2 - r_*^2}}{2}\right) \\
F: \quad (x_F, y_F) &= \left(\frac{3r_* - \sqrt{3r^2 - 3r_*^2}}{2} \; ; \; \frac{\sqrt{3}r_* - \sqrt{r^2 - r_*^2}}{2}\right)
\end{aligned}
\right\} \qquad (14)
$$

The angle subtended by two vertices of an inner triangle

The following figure shows angle α centered at the center of circle #1. Angle α is the angle subtended by two vertices of the inner triangle.



Using the spatial co-ordinates we have already calculated, we can write the following angles:

$$\left.\begin{aligned}\tan(<DO\hat{x}) &= \frac{y_D}{x_D} = \frac{\sqrt{3}r_* - \sqrt{r^2 - r_*^2}}{r_* + \sqrt{3r^2 - 3r_*^2}} \\ \tan(<AO\hat{x}) &= \frac{y_A}{x_A} = \frac{\sqrt{r^2 - r_*^2}}{r_*}\end{aligned}\right\} \quad (15)$$

And, from these, we can write the desired angle α as:

$$\alpha = <DO\hat{x} - <AO\hat{x} = \tan^{-1}\left(\frac{\sqrt{3}r_* - \sqrt{r^2 - r_*^2}}{r_* + \sqrt{3r^2 - 3r_*^2}}\right) - \tan^{-1}\left(\frac{\sqrt{r^2 - r_*^2}}{r_*}\right) \quad (16)$$

The lengths of the arc and line segment between points A and D

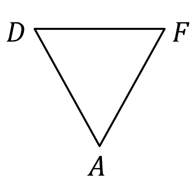
The length of arc $\widetilde{AD}$ is proportional to angle α:

$$\left|\widetilde{AD}\right| = r\alpha \quad (17)$$

The length of line segment $\overline{AD}$ can be calculated using the Pythagorean Theorem:

$$\begin{aligned}|\overline{AD}| &= \sqrt{(x_A - x_D)^2 + (y_A - y_D)^2} \\ &= \sqrt{\left(r_* - \frac{r_* + \sqrt{3r^2 - 3r_*^2}}{2}\right)^2 + \left(\sqrt{r^2 - r_*^2} - \frac{\sqrt{3}r_* - \sqrt{r^2 - r_*^2}}{2}\right)^2} \quad (18)\end{aligned}$$

The area of the straight-sided equilateral triangle ΔDAF

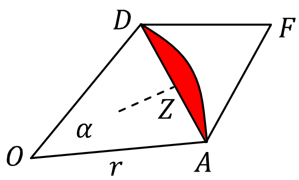


Consider the equilateral triangle with straight sides having length $|\overline{AD}|$. We can write its area as:

$$
\begin{aligned}
Area_{\Delta ADF} &= ½ \times Base \times Height \\
&= ½ \times |\overline{AD}| \times \frac{\sqrt{3}}{2}|\overline{AD}| \\
&= \frac{\sqrt{3}}{4}|\overline{AD}|^2 \qquad (19)
\end{aligned}
$$

<u>The area of one of the circular segments</u>

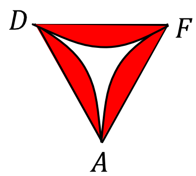
Consider the segment of a circle, rendered in red in this figure, which impinges on one side of triangle DAF.



$$
\begin{aligned}
Area_{seg} &= Area_{pie\ wedge\ ODA} - Area_{\Delta ODA} \\
&= \left(\frac{\alpha}{2\pi}\right) Area_{circle\ O} - 2Area_{\Delta OZA} \\
&= \left(\frac{\alpha}{2\pi}\right)\pi r^2 - 2\left[\frac{1}{2} \times \frac{1}{2}|\overline{AD}| \times r\cos\left(\frac{\alpha}{2}\right)\right] \\
&= \left(\frac{\alpha}{2}\right) r^2 - \frac{1}{2}|\overline{AD}| r\cos\frac{\alpha}{2} \qquad (20)
\end{aligned}
$$

<u>The cross-sectional area of one of the inner "triangles" of a propellant grain</u>

Now we have the data needed to calculate the cross-sectional area of one of the inner "triangles" of propellant. That is the cross-section of the triangular star which is <u>not</u> shaded in red in the figure. It is equal to the area of straight-sided triangle ΔADF less the area of three of the red segments.



$$
\begin{aligned}
Area_{inner\ "triangle"} &= Area_{\Delta ADF} - 3Area_{seg} \\
&= \frac{\sqrt{3}}{4}|\overline{AD}|^2 - 3\left[\left(\frac{\alpha}{2}\right) r^2 - \frac{1}{2}|\overline{AD}| r\cos\frac{\alpha}{2}\right] \qquad (21)
\end{aligned}
$$

<u>How do we apply all this information to our problem?</u>

Our goal is to calculate the mass (or volume) of propellant of an inner "triangle" which is burned during one time step. Here's what we will know at the start of a time step when we are ready to make the calculation.

- The duration of the upcoming time step ΔT
- The effective radius of the inner holes at the end of the previous time step r_{start}
- As before, assume the ends of the grain do not burn, so the length of the grain remains constant at l_0
- The burn rate for the upcoming time step will be B_{start}
- The radius r_* when the grain disintegrated into 12 "triangles"

Step #1: Calculate angle α_{start} at the start of the time step.

$$
\alpha_{start} = \tan\left(\frac{\sqrt{3}r_* - r_{start}}{r_* + \sqrt{3}r_{start}}\right) - \tan\left(\frac{\sqrt{r_*^2 - r_{start}^2}}{r_*}\right) \qquad (22)
$$

Step #2: Calculate the length of line segment $|\overline{AD}|$ at the start of the time step.

$$|\overline{AD}|_{start} = \sqrt{2r_*^2 - \sqrt{3}r_* r_{start} - \sqrt{r_*^2 - r_{start}^2}\left(\sqrt{3}r_* - r_{start}\right)} \quad (23)$$

Step #3: Calculate the cross-sectional area of the "triangle" at the start of the time step.

$$Area_{start} = \frac{\sqrt{3}}{4}|\overline{AD}|_{start}^2 - 3\left[\left(\frac{\alpha_{start}}{2}\right)r_{start}^2 - \frac{1}{2}|\overline{AD}|_{start} r_{start}\cos\frac{\alpha_{start}}{2}\right] \quad (24)$$

Step #4: Calculate the volume of the "triangle" at the start of the time step.

$$Volume_{start} = l_0 \times Area_{start} \quad (25)$$

Step #5: Calculate the depth down into the surface which will be burned during this time step.

$$Depth = B_{start}\Delta T \quad (26)$$

Step #6: Calculate the effective radius of the inner holes at the end of this time step r_{end}.

$$r_{end} = r_{start} - Depth \quad (27)$$

Step #7: Calculate angle α_{end} at the end of this time step.

$$\alpha_{end} = \tan\left(\frac{\sqrt{3}r_* - r_{end}}{r_* + \sqrt{3}r_{end}}\right) - \tan\left(\frac{\sqrt{r_*^2 - r_{end}^2}}{r_*}\right) \quad (28)$$

Step #8: Calculate the length of line segment $|\overline{AD}|$ at the end of this time step.

$$|\overline{AD}|_{end} = \sqrt{2r_*^2 - \sqrt{3}r_* r_{end} - \sqrt{r_*^2 - r_{end}^2}\left(\sqrt{3}r_* - r_{end}\right)} \quad (29)$$

Step #9: Calculate the cross-sectional area of the "triangle" at the end of this time step.

$$Area_{end} = \frac{\sqrt{3}}{4}|\overline{AD}|_{end}^2 - 3\left[\left(\frac{\alpha_{end}}{2}\right)r_{end}^2 - \frac{1}{2}|\overline{AD}|_{end} r_{end}\cos\frac{\alpha_{end}}{2}\right] \quad (30)$$

Step #10: Calculate the volume of the "triangle" at the end of this time step.

$$Volume_{end} = l_0 \times Area_{end} \quad (31)$$

Step #11: Calculate the reduction in volume during this time step.

$$VolumeBurned = Volume_{end} - Volume_{start} \quad (32)$$

When is the burning of the inner "triangles" complete?

Burning will be complete when points A, D and F are coincident. Setting the x-values of points A and D equal gives:

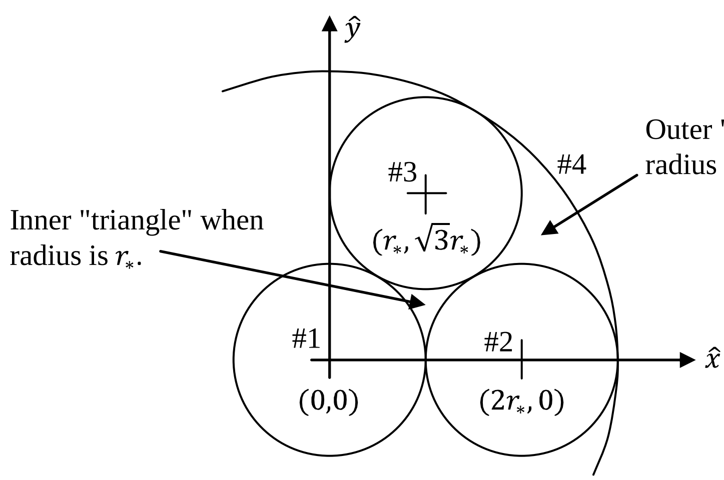
$$
\begin{aligned}
& x_A = x_D \\
\rightarrow \quad & r_* = \frac{r_* + \sqrt{3r^2 - 3r_*^2}}{2} \\
\rightarrow \quad & r_* = \sqrt{3r^2 - 3r_*^2} \\
\rightarrow \quad & r_*^2 = 3r^2 - 3r_*^2 \\
\rightarrow \quad & 3r^2 = 4r_*^2 \\
\rightarrow \quad & r_{End\ of\ Phase2} = \frac{2}{\sqrt{3}} r_* \qquad (33)
\end{aligned}
$$

The outer "triangles" will continue to burn even after the inner "triangles" have been totally consumed.

The volume of one of the outer "triangles" during the second and third phases of burning

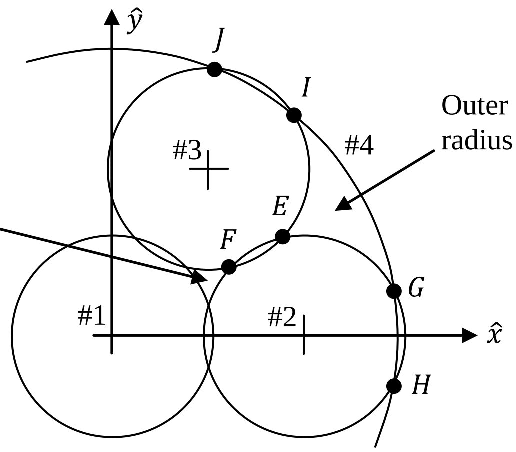
I will start this analysis back at the end of phase 1, when the radius of the holes is equal to $r_* = r_{End\ of\ Phase1}$. The following figure shows four circles. The three smaller, complete, circles are the same ones considered in the previous section, whose intersection defines a typical inner "triangle". In this figure, a portion of a fourth circumscribing circle, labeled circle #4, is also shown. Its intersection with circles #2 and #3 defines a typical outer "triangle".

.



All of these circles represent the burning surface at the instant when the propellant grain ceases to be a single entity and separates into 12 roughly triangular prisms. At this moment, the inner circles (that is, the holes) have radius r_* and the circumscribing circle #4 has radius $3r_*$.

Now consider these same circles at some later time, when the radii of the holes has increased from r_* to r. During this time, the radius of the circumscribing circle will have decreased by the same amount, $r - r_*$. I have identified six points of intersection. Two of them – points E and F – are the same as were defined in the previous section. Three of the points – points E, G and I – are the three vertices of one of the outer "triangles".



.

The equations of three of the circles of interest are:

$$\left.\begin{array}{ll} \#2: & (x-2r_*)^2+y^2=r^2 \\ \#3: & (x-r_*)^2+\left(y-\sqrt{3}r_*\right)^2=r^2 \\ \#4: & x^2+y^2=(4r_*-r)^2 \end{array}\right\} \quad (34)$$

Note the radius of the circumscribing circle. As the radius of the smaller circles increases from r_* to r, the radius of the large circle decreases by distance $r-r_*$ from its starting value $3r_*$ to $4r_*-r$.

<u>The two points of intersection between circles #2 and #4</u>

Setting y^2 common to circles #2 and #4 gives:

$$\begin{array}{rc} & r^2-(x-2r_*)^2=(4r_*-r)^2-x^2 \\ \rightarrow & r^2-(x^2-4r_*x+4r_*^2)=(16r_*^2-8r_*r+r^2)-x^2 \\ \rightarrow & x=5r_*-2r \end{array}$$

Now substituting this value of x into the equation for circle #2 gives:

$$\begin{array}{rl} & (3r_*-2r)^2+y^2=r^2 \\ \rightarrow & 9r_*^2-12r_*r+3r^2+y^2=0 \\ \rightarrow & 3(3r_*-r)(r_*-r)+y^2=0 \\ \rightarrow & y=\pm\sqrt{3(3r_*-r)(r-r_*)} \end{array}$$

and the co-ordinates of the two points of intersection are:

$$\left.\begin{array}{ll} G: & (x_G,y_G)=\left(5r_*-2r\ ;\ +\sqrt{3(3r_*-r)(r-r_*)}\right) \\ H: & (x_H,y_H)=\left(5r_*-2r\ ;\ -\sqrt{3(3r_*-r)(r-r_*)}\right) \end{array}\right\} \quad (35)$$

<u>The two points of intersection between circles #2 and #3</u>

Setting x and y common to circles #2 and #3 gives:

$$(x - 2r_*)^2 + y^2 = (x - r_*)^2 + \left(y - \sqrt{3}r_*\right)^2$$

$$\rightarrow \qquad x = \sqrt{3}y$$

Now substituting this value of x into the equation for circle #2 gives:

$$\left(\sqrt{3}y - 2r_*\right)^2 + y^2 = r^2$$

$$\rightarrow \quad y = \frac{4\sqrt{3}r_* \pm \sqrt{48r_*^2 - 16(4r_*^2 - r^2)}}{8}$$

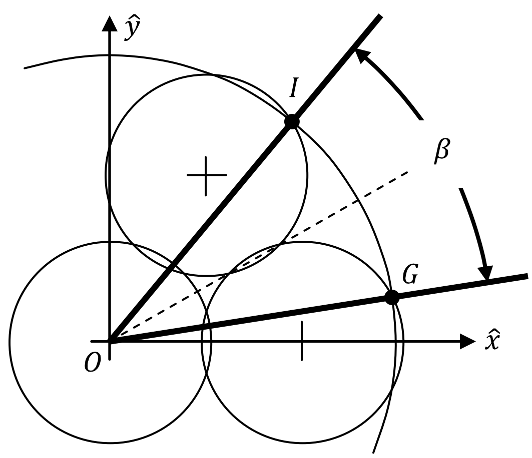
$$\rightarrow \qquad y = \frac{\sqrt{3}r_* \pm \sqrt{r^2 - r_*^2}}{2}$$

and the co-ordinates of the two points of intersection are:

$$\left.\begin{array}{l} E: \quad (x_E, y_E) = \left(\dfrac{3r_* + \sqrt{3r^2 - 3r_*^2}}{2} \; ; \; \dfrac{\sqrt{3}r_* + \sqrt{r^2 - r_*^2}}{2}\right) \\ F: \quad (x_F, y_F) = \left(\dfrac{3r_* - \sqrt{3r^2 - 3r_*^2}}{2} \; ; \; \dfrac{\sqrt{3}r_* - \sqrt{r^2 - r_*^2}}{2}\right) \end{array}\right\} \qquad (36)$$

<u>The angle β subtended by two vertices of an outer "triangle"</u>

The following figure shows angle β centered at the center of circle #1, which is also the center of circle #4. Angle β is the angle subtended by vertices G and I of the outer "triangle".

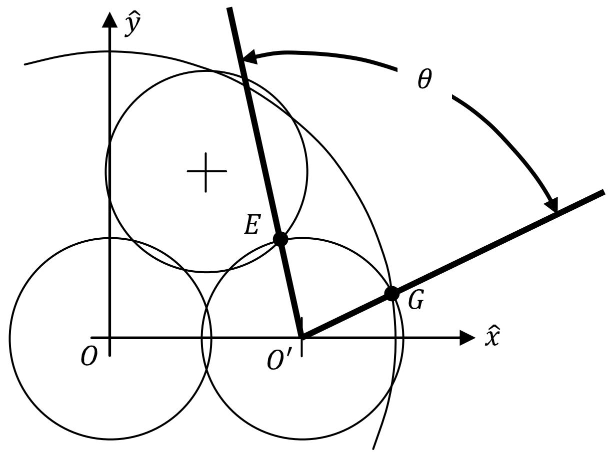


The dotted line bisects angle β. Since circles #2 and #3 are two of the six circles arranged symmetrically around the center of the grain, the dotted line will have a slope of 30°. We can express angle β as follows:

$$\begin{aligned} \beta &= 2[30° - (< GO\hat{x})] \\ &= 2\left[\frac{\pi}{6} - \tan^{-1}\left(\frac{y_G}{x_G}\right)\right] \\ &= 2\left[\frac{\pi}{6} - \tan^{-1}\left(\frac{\sqrt{(3r_* - r)(r - r_*)}}{5r_* - 2r}\right)\right] \qquad (37) \end{aligned}$$

The angle θ subtended by two other vertices of an outer "triangle"

The following figure shows angle θ centered at the center of circle #2. Angle θ is the angle subtended by vertices E and G of the outer "triangle".



When we calculated the co-ordinates of points E and G above, we expressed them with respect to the origin at point O. These co-ordinates can be restated with respect to origin O' by reducing the x-co-ordinate by $2r_*$ as follows:

$$E_{wrt\ O'}: \quad (\Delta x_E, \Delta y_E) = \left(\frac{-r_* + \sqrt{3r^2 - 3r_*^2}}{2} \ ; \ \frac{\sqrt{3}r_* + \sqrt{r^2 - r_*^2}}{2} \right)$$

$$G_{wrt\ O'} \quad (\Delta x_G, \Delta y_G) = \left(3r_* - 2r \ ; \ \sqrt{(3r_* - r)(r - r_*)} \right)$$

Angle θ can then be calculated using straight-forward trigonometry:

$$\begin{aligned} \theta \quad &= \tan^{-1}\left(\frac{\Delta y_E}{\Delta x_E}\right) - \tan^{-1}\left(\frac{\Delta y_G}{\Delta x_G}\right) \\ &= \tan^{-1}\left(\frac{\sqrt{3}r_* + \sqrt{r^2 - r_*^2}}{-r_* + \sqrt{3r^2 - 3r_*^2}}\right) - \tan^{-1}\left(\frac{\sqrt{(3r_* - r)(r - r_*)}}{3r_* - 2r}\right) \quad (38) \end{aligned}$$

The lengths of arc segments $\widetilde{GI}$ and $\widetilde{EG}$

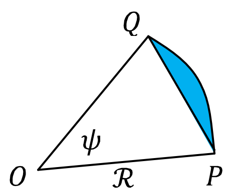
The length of arc $\widetilde{GI}$ is proportional to angle β and the radius of circle #4:

$$\left|\widetilde{GI}\right| = (4r_* - r)\beta \quad (39)$$

The length of arc $\widetilde{EG}$ is proportional to angle θ and the radius of the small circles:

$$\left|\widetilde{EG}\right| = r\theta \quad (40)$$

The area enclosed between an arc segment and a straight line segment



I will do this calculation for the generalized situation, as drawn at the left. The area between arc $\widetilde{PQ}$ and line segment $\overline{PQ}$ is shaded in blue. The radius and subtended angle are also labeled, as $\mathcal{R}$ and ψ, respectively. The lengths of the two segments can be written down using the same relationships from above:

$$\left|\widetilde{PQ}\right| = \psi\mathcal{R} \qquad (41A)$$
$$\left|\overline{PQ}\right| = 2\mathcal{R}\sin(½\psi) \qquad (41B)$$

The area of the wedge-shaped pie segment OPQ is given by:

$$Area_{pie} = ½\psi\mathcal{R}^2 \qquad (42)$$

The area of straight-sided triangle OPQ is given by:

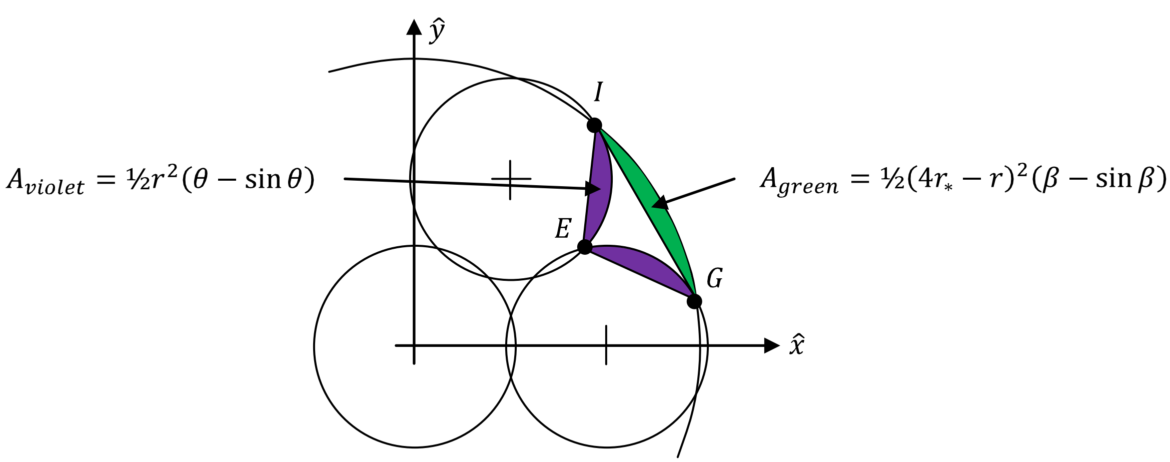
$$Area_{<OPQ} = ½ \times \left|\overline{PQ}\right| \times \mathcal{R}\cos(½\psi) = \mathcal{R}^2\sin(½\psi)\cos(½\psi) \qquad (43)$$

The blue-shaded area is the difference:

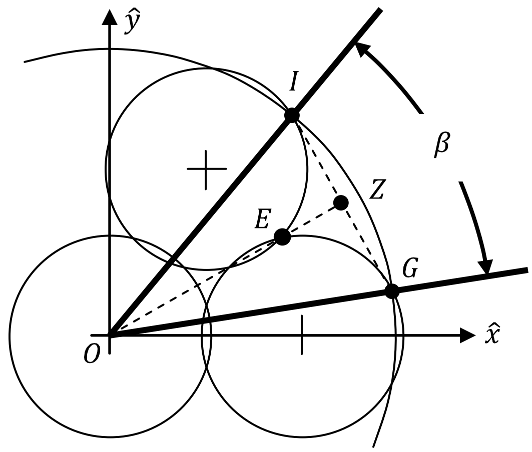
$$\begin{aligned} Area_{blue} &= Area_{pie} - Area_{<OPQ} \\ &= ½\mathcal{R}^2[\psi - 2\sin(½\psi)\cos(½\psi)] \\ &= ½\mathcal{R}^2(\psi - \sin\psi) \qquad (44) \end{aligned}$$

The cross-sectional area of an outer "triangle" of a propellant grain

A typical outer "triangle" has vertices E, G and I, as shown in the figure below. Its cross-sectional area is equal to: (i) the area of the straight-sided triangle EGI, (ii) plus the area of the green circular segment shown below, and (iii) less the areas of the two purple circular segments. The areas of the circular segments have been determined using Equation (41) with the appropriate central angles and radii.



In order to complete the calculation, we need to find the area of the straight-sided triangle EGI. I will do that with the aid of the following figure.



The "base" of triangle EGI is line segment $\overline{GI}$, whose length can be calculated using Equation $(41B)$ as:

$$|\overline{GI}| = 2(4r_* - r)\sin(½\beta) \qquad (45)$$

The length of line segment $\overline{OZ}$ can be found in a similar way, as:

$$|\overline{OZ}| = (4r_* - r)\cos(½\beta) \qquad (46)$$

The length of line segment $\overline{OE}$ can be calculated directly using the Pythagorean Theorem, as:

$$\begin{aligned} |\overline{OE}| &= \sqrt{x_E^2 + y_E^2} \\ &= \sqrt{\left(\frac{3r_* + \sqrt{3r^2 - 3r_*^2}}{2}\right)^2 + \left(\frac{\sqrt{3}r_* + \sqrt{r^2 - r_*^2}}{2}\right)^2} \qquad (47) \end{aligned}$$

The "height" of triangle EGI is line segment $\overline{OZ}$, whose length can be found by subtraction, as:

$$|\overline{EZ}| = |\overline{OZ}| - |\overline{OE}| \qquad (49)$$

Then, the area of triangle EGI can be calculated as:

$$\begin{aligned} Area_{<EGI} &= ½|\overline{GI}||\overline{EZ}| \\ &= (4r_* - r)\sin(½\beta)\left[(4r_* - r)\cos(½\beta) - \sqrt{\left(\frac{3r_* + \sqrt{3r^2 - 3r_*^2}}{2}\right)^2 + \left(\frac{\sqrt{3}r_* + \sqrt{r^2 - r_*^2}}{2}\right)^2}\right] \qquad (50) \end{aligned}$$

<u>How do we apply all this information to our problem?</u>

Our goal is to calculate the mass (or volume) of propellant in an outer "triangle" which is burned during one time step. Here's what we will know at the start of a time step when we are ready to make the calculation.

- The duration of the upcoming time step ΔT
- The effective radius of the inner holes at the end of the previous time step r_{start}

- As before, assume the ends of the grain do not burn, so the length of the grain is l_0
- The burn rate for the upcoming time step will be B_{start}
- The radius r_* when the grain disintegrated into 12 "triangles"

Step #1: Calculate angle β_{start} at the start of the time step.

$$\beta_{start} = 2\left[\frac{\pi}{6} - \tan^{-1}\left(\frac{\sqrt{(3r_* - r_{start})(r_{start} - r_*)}}{5r_* - 2r_{start}}\right)\right] \quad (51)$$

Step #2: Calculate angle θ_{start} at the start of the time step.

$$\theta_{start} = \tan^{-1}\left(\frac{\sqrt{3}r_* + \sqrt{r_{start}^2 - r_*^2}}{-r_* + \sqrt{3r_{start}^2 - 3r_*^2}}\right) - \tan^{-1}\left(\frac{\sqrt{(3r_* - r_{start})(r_{start} - r_*)}}{3r_* - 2r_{start}}\right) \quad (52)$$

Step #3: Calculate the lengths of certain straight line segments at the start of the time step:

$$\left.\begin{aligned}
|\overline{GI}|_{start} &= 2(4r_* - r_{start})\sin(½\beta_{start}) \\
|\overline{OZ}|_{start} &= (4r_* - r_{start})\cos(½\beta_{start}) \\
|\overline{OE}|_{start} &= ½\sqrt{\left(3r_* + \sqrt{3r_{start}^2 - 3r_*^2}\right)^2 + \left(\sqrt{3}r_* + \sqrt{r_{start}^2 - r_*^2}\right)^2} \\
|\overline{EZ}|_{start} &= |\overline{OZ}|_{start} - |\overline{OE}|_{start}
\end{aligned}\right\} \quad (53)$$

Step #4: Calculate the cross-sectional area of the outer-bulge green segments at the start of the time step.

$$A_{green}\big|_{start} = ½(4r_* - r_{start})^2(\beta_{start} - ½\sin\beta_{start}) \quad (54)$$

Step #5: Calculate the cross-sectional area of the inner-bulge violet segments at the start of the time step.

$$A_{violet}|_{start} = ½r_{start}^2(\theta_{start} - ½\sin\theta_{start}) \quad (55)$$

Step #6: Calculate the cross-sectional area of the "triangle" at the start of the time step.

$$Area_{start} = ½\,|\overline{GI}|_{start}|\overline{EZ}|_{start} + A_{green}\big|_{start} - 2A_{violet}|_{start} \quad (56)$$

Step #7: Calculate the volume of the "triangle" at the start of the time step.

$$Volume_{start} = l_0 \times Area_{start} \quad (57)$$

Step #8: Calculate the depth down into the surface which will be burned during this time step.

$$Depth = B_{start}\Delta T \quad (58)$$

Step #9: Calculate the effective radius of the inner holes at the end of this time step r_{end}.

$$r_{end} = r_{start} + Depth \quad (59)$$

Step #10: Repeat Steps #1 through #7 at the end of this time step, to get $Volume_{end}$.

Step #11: Calculate the reduction in volume during this time step.

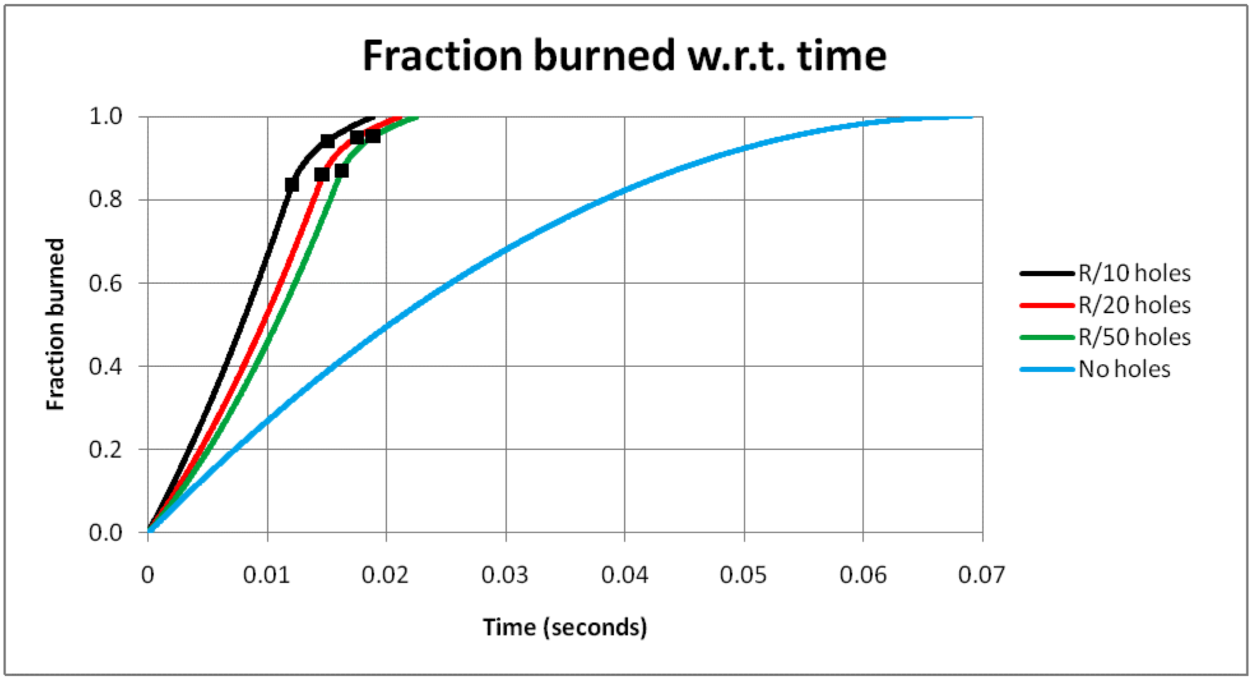
$$VolumeBurned = Volume_{end} - Volume_{start} \qquad (60)$$

<u>When is the burning of the outer "triangles" complete?</u>

Burning will be complete when points E, G and I are coincident. Setting the x-values of points E and G equal gives:

$$
\begin{aligned}
& x_E = x_G \\
\rightarrow\quad & \frac{3r_* + \sqrt{3r^2 - 3r_*^2}}{2} = 5r_* - 2r \\
\rightarrow\quad & \sqrt{3r^2 - 3r_*^2} = 7r_* - 4r \\
\rightarrow\quad & 3r^2 - 3r_*^2 = 49r_*^2 - 56r_*r + 16r^2 \\
\rightarrow\quad & 13r^2 - 56r_*r - 52r_*^2 = 0 \\
\rightarrow\quad & r = \frac{56r_* \pm \sqrt{3136r_*^2 + 2704r_*^2}}{26} \\
\rightarrow\quad & r = \frac{56 \pm \sqrt{432}}{26} r_* \\
\rightarrow\quad & r = \frac{28 \pm 6\sqrt{3}}{13} r_* \qquad (61)
\end{aligned}
$$

The following figure illustrates the effect of holes in the web of the propellant grain I used in the earlier paper, which had an initial outer diameter of 6.9 millimeters. Note that these curves use a constant burning rate of 50 millimeters (into the surface) per second. This is simply a test to compare the rate at which the propellant is consumed. The blue curve represents the burning when the grain has no holes, as was assumed in the earlier paper. For the black curve, the holes have a diameter one-tenth that of the grain. For the red curve, the holes are smaller, with a diameter only one-twentieth of the grain diameter. For the green curve, it is one-fiftieth. In each case, the fraction burned is plotted with respect to time.



In order to make sure that burning processes illustrated in the graph are comparable, I adjusted the axial length of the grains to ensure that each grain contained the same volume of propellant.

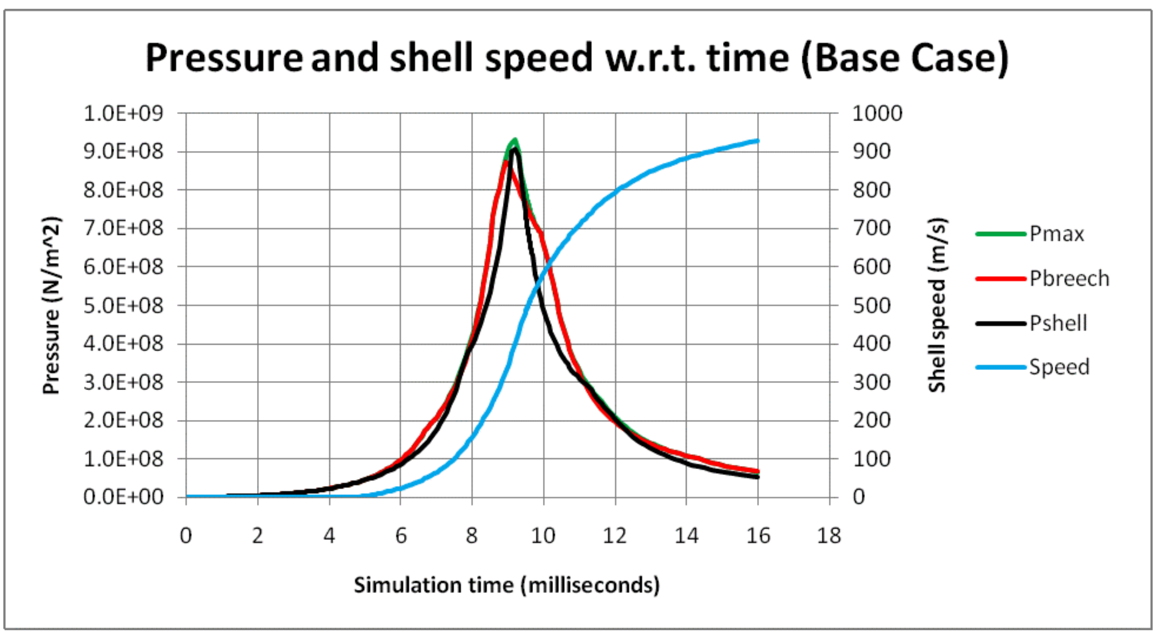
It is clear that the existence of holes causes the burning to take place more quickly. Even so, the starting size of the holes has surprisingly little effect on the time it takes to consume the propellant. The very fact that there are holes seems to be much more important than the holes' actual size, although larger holes do result in slightly faster consumption.

There are two small black squares on the curves which represent the burning when there are holes. The first (earlier in time) square is the moment when the grain separates into 12 "triangular" cylinders. The later square is the moment when the inner "triangular" cylinders have been consumed and only the six outer "triangles" remain. These two later phases arise only in the last 20% or less of the process.

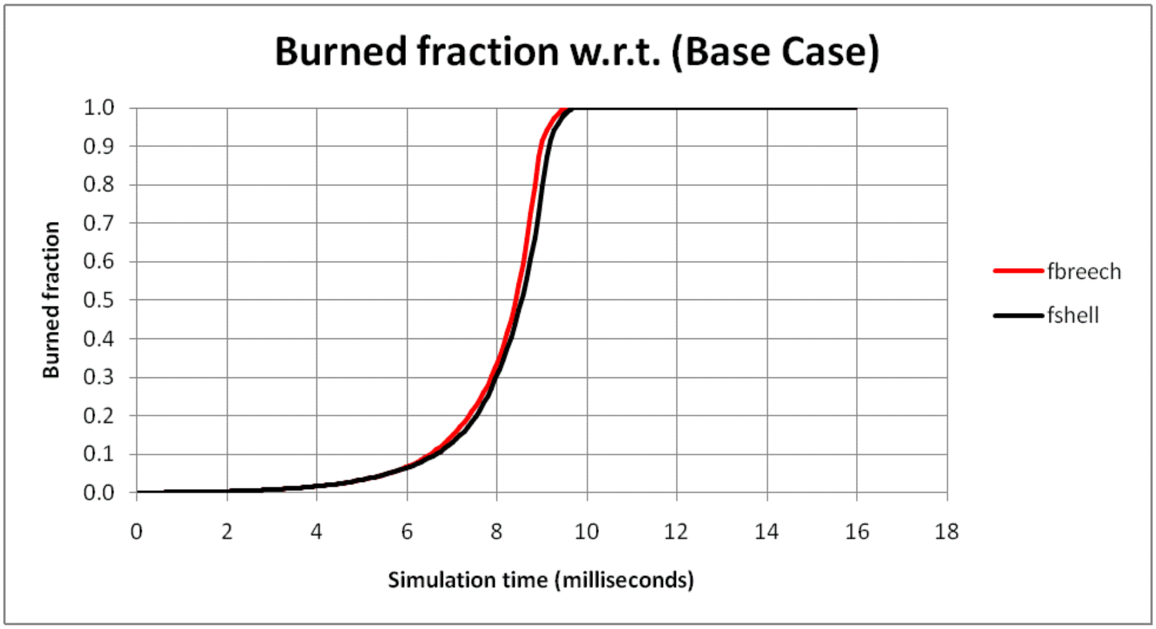
Bear in mind that I have assumed that there are seven holes in the grain, six holes arranged in a circle around a central one. This is a common configuration, particularly for very large naval guns, but it is not the only configuration. If a grain has fewer holes, then the burning rate will lie somewhere between the blue and green curves above. An easy way to control the rate of burning, and thus to control the power of the resulting shot, is to use propellant with a different number of holes.

Results from the Base Case simulation

As a Base Case, I used holes with a diameter one-twentieth the diameter of the grains. I avoided the use of artificial viscosity by setting the C_1 viscosity coefficient to zero. The following graph shows the results. The speed of the shell along the barrel is the blue line, to be referenced to the vertical axis on the right-hand side. The pressure at three points along the active chamber are plotted against the left-hand vertical axis. They are the pressures at the breech (element #1), halfway along (element #1000) and next to the shell (element #2000).

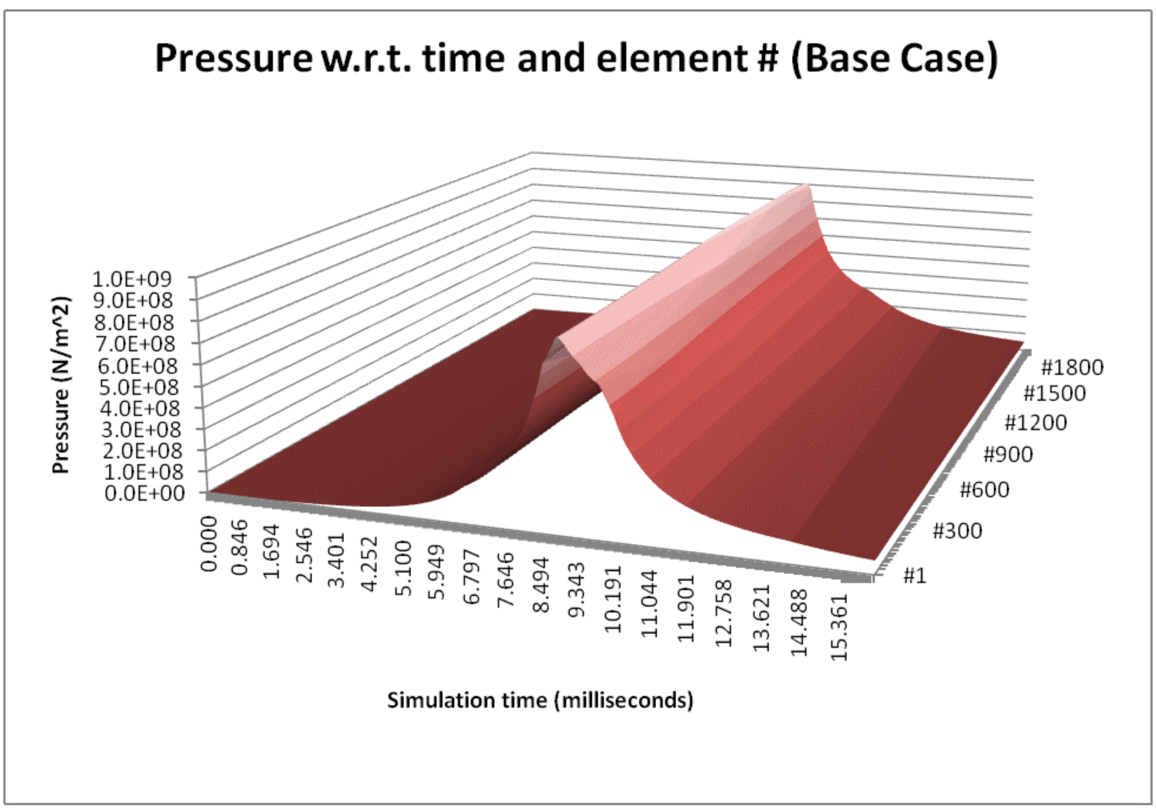


The pressure peaks at about 9,000 atmospheres and the shell reaches a speed of about 920 meters per second. The addition of holes has dramatically increased the pressure and speed compared with the unpierced grains we looked at in the earlier paper. The following graph shows the fraction of propellant burned with respect to time, both at the breech and right behind the shell. Not only is all of the propellant consumed (unlike the unpierced grains), but it is entirely consumed long before the shell reaches the end of the barrel.



In fact, things are now happening too fast. For this gun, the pressure should peak at about 4,000 atmospheres and the shell should reach an exit speed of about 750 meters per second. In the earlier paper, burning took place too slowly. Now, it takes place too quickly. The only difference is the number of holes in the web. A tentative conclusion could be drawn that seven holes is two or three too many.

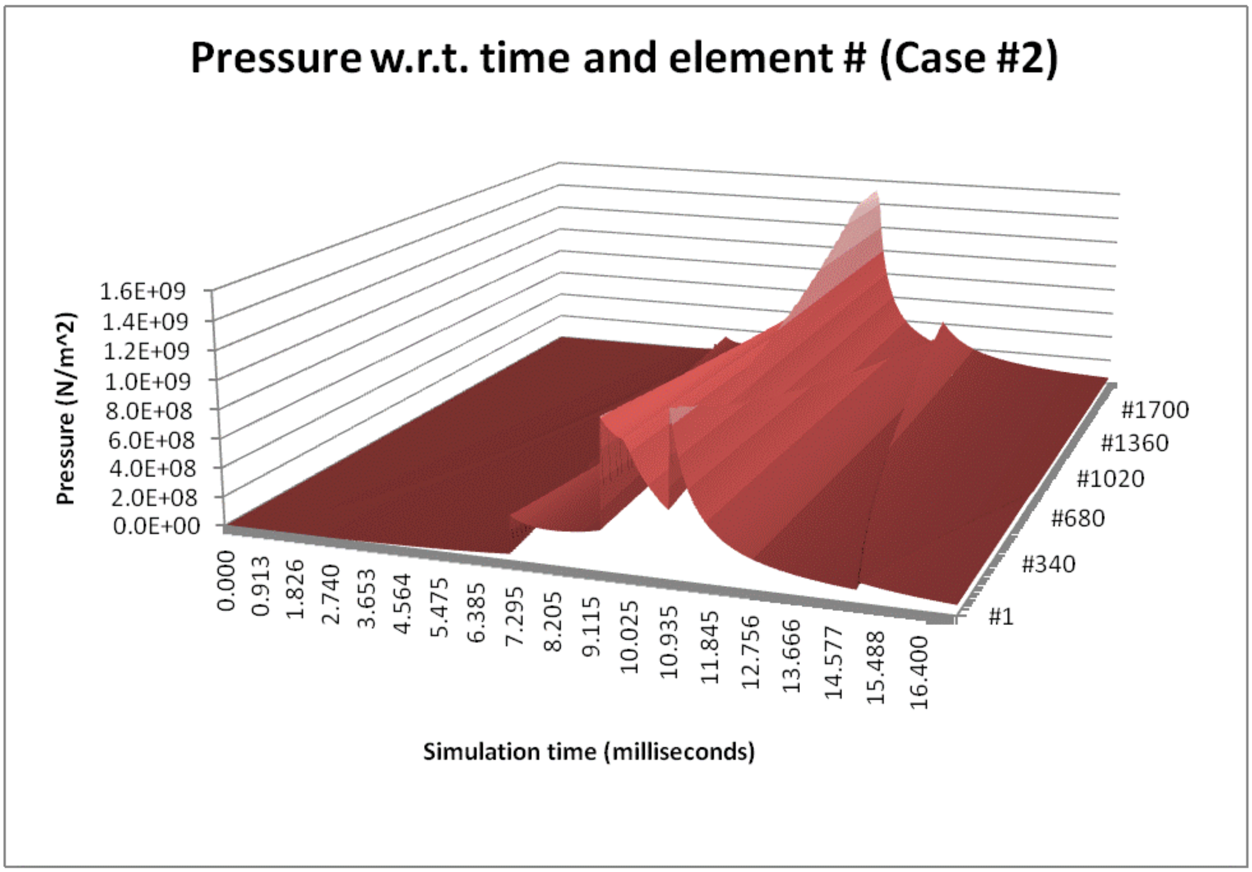
The following surface chart shows the pressure (the vertical axis) through time (the horizontal axis) all along the active chamber (the depth axis). There is a little evidence of discontinuities in the pressure, but no real shock waves.



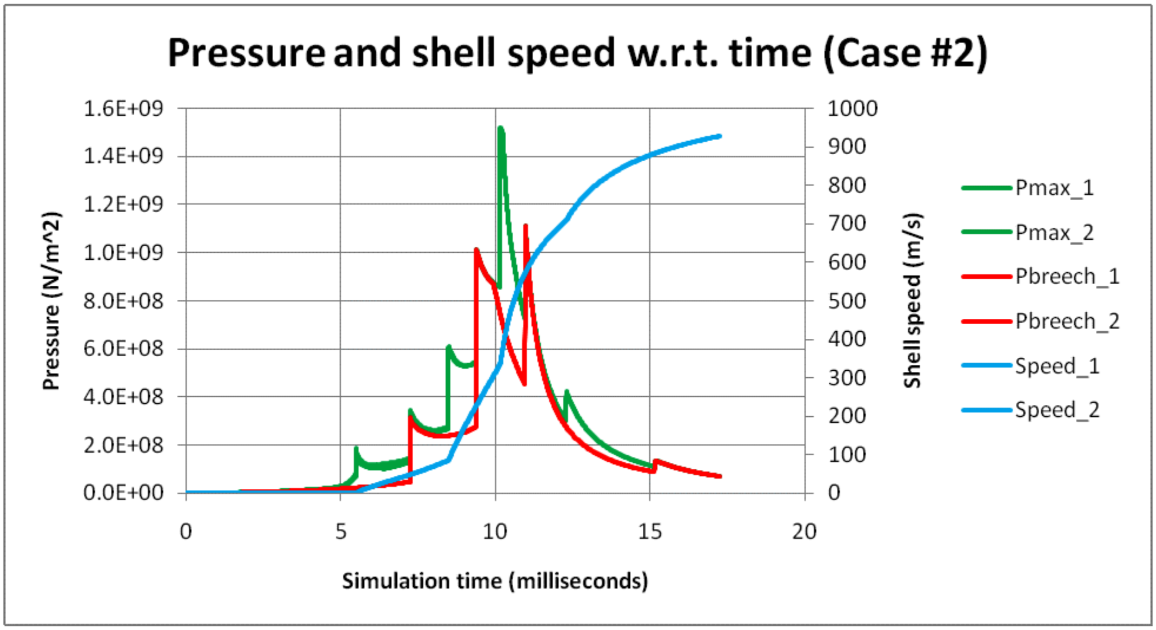
Progressive ignition

Up to this point, I have always assumed that combustion starts at the same time everywhere in the chamber. That is not so. Combustion starts at the breech and then propagates through the propellant down towards the shell. As each bit of propellant starts to burn, it heats up and ignites a neighbouring bit. A really thorough analysis could capture this. It would track the temperature of each element. Only when the temperature reached the flash point temperature would the neighbouring element begin to burn.

I have used a slightly simpler model for ignition. I have assumed that the ignition point moves from the breech towards the shell at a constant speed. I have used the speed of sound, 343.2 meters per second, for this constant speed. Since the chamber is approximately one meter long, element #2000 (situated right next to the shell) does not start burning until about three milliseconds after element #1 (situated right next to the breech) starts burning. The effects are dramatic; significant shock waves develop. In order to carry through the numerical integration, I had to invoke artificial viscosity. I set the two viscosity coefficients to $C_1 = 5$ and $C_2 = C_1/10$. The following graph shows the pressure (the vertical axis) through time (the horizontal axis) all along the active chamber (the depth axis).

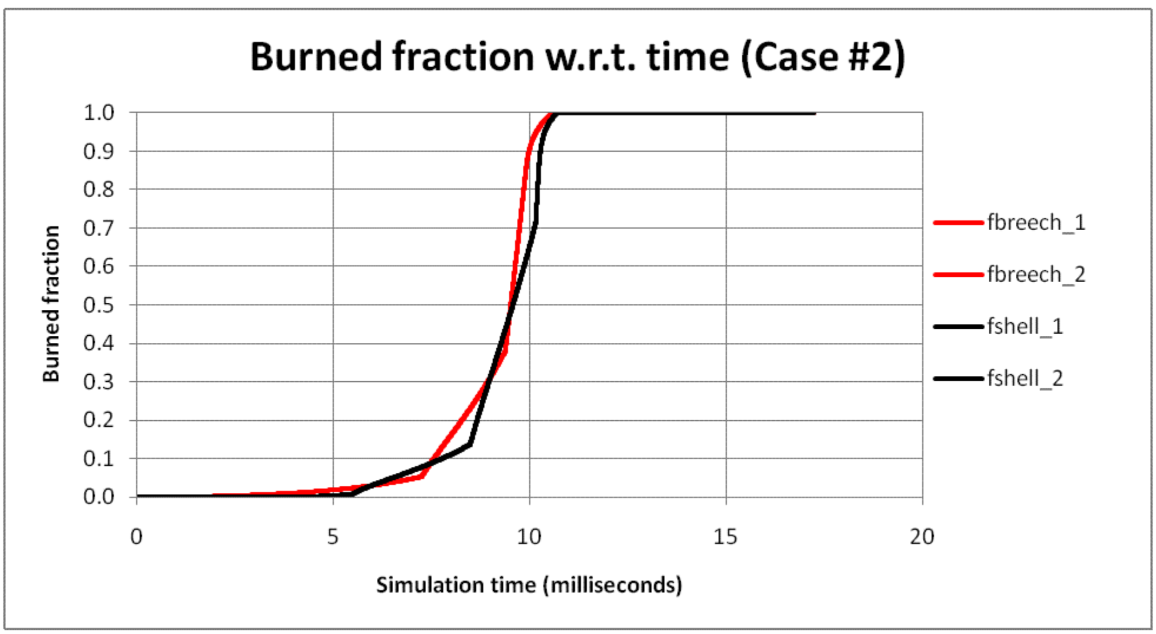


Introducing progressive ignition, rather than instantaneous ignition, did not do what I had anticipated it might do. I had thought (hoped) that delaying the time at which elements were ignited would, through overlapping burning cycles, lead to an overall reduction in the peak pressure experienced. That is definitely not the case. The development of shock waves has led to peak pressures much higher than before. That can be seen more clearly in the following graph, which shows the shell's speed (right-hand vertical axis) and the pressure at three element locations in the active chamber (left-hand vertical axis).



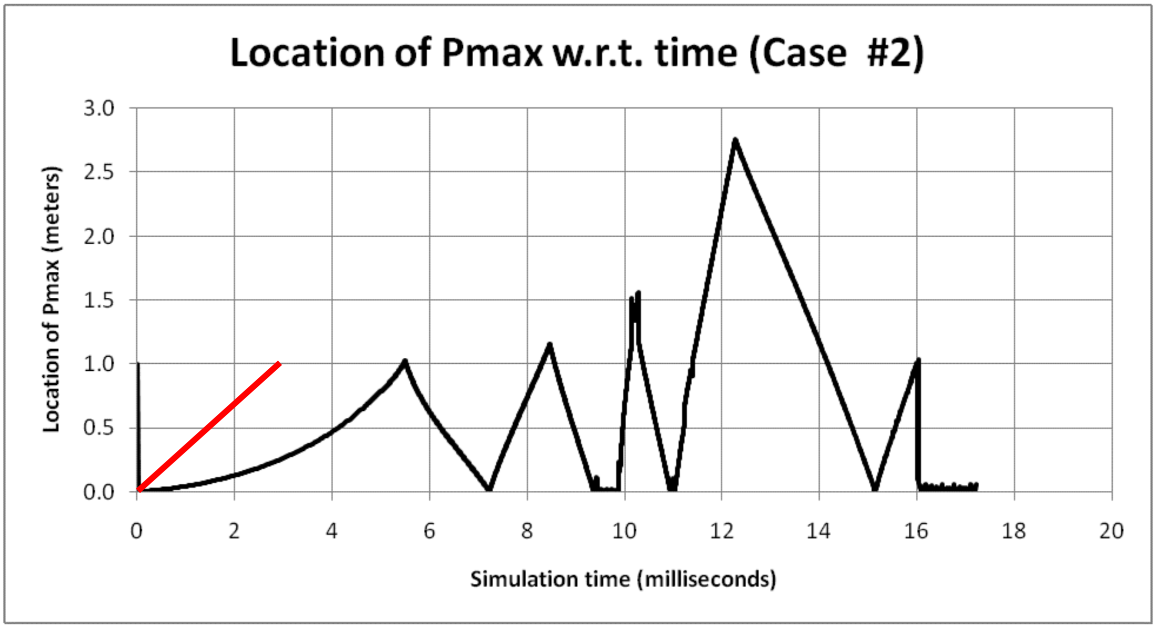
The peak pressure has increased to more than 15,000 atmospheres[1]. The effect of the shock waves has even carried over into the shell's "jerk". (The jerk is the rate-of-change of the shell's <u>acceleration</u>. The discontinuities in the curve of the shell's speed with respect to time are periods of non-zero jerk.)

The following graph shows how quickly the burning progressed at the breech end and next to the shell.

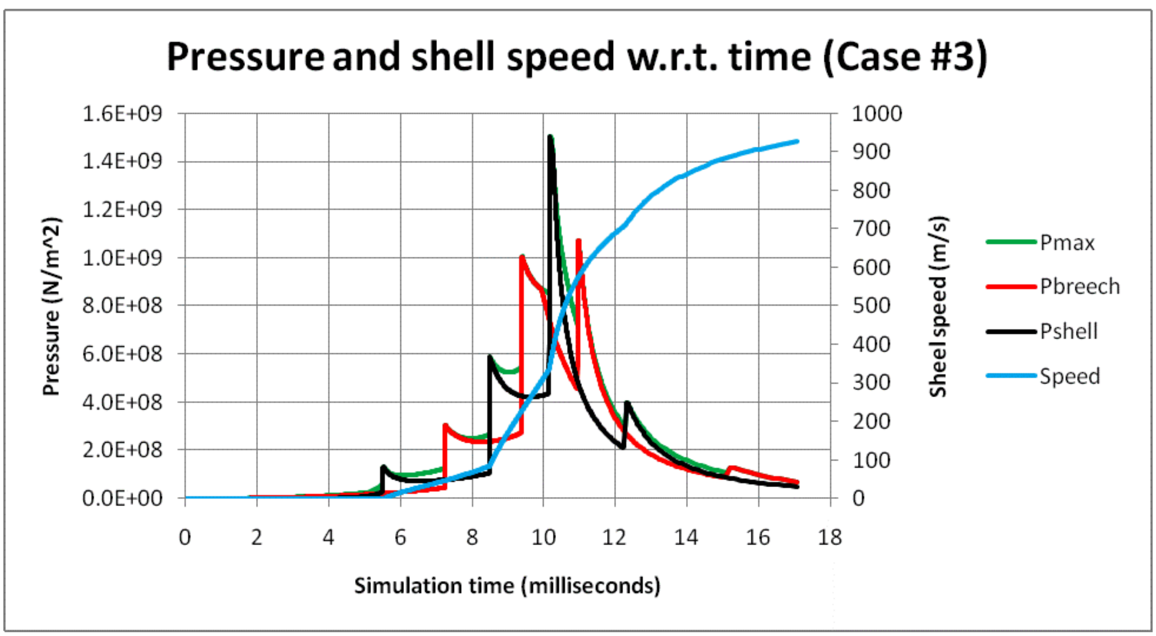


[1] Unlike the previous graph of this type, the legend lists two series for the maximum pressure, two series for the pressure at the breech, and so on. There is no significance to this. Excel limits the number of data points per series to 32,000. Since there were more than 32,000 data points, more than one series needed to be defined in order to construct the graph. Excel gives each series its own line in the legend.

The following graph shows the location (in meters from the breech) of the element having the highest pressure in the active chamber. This is one way to represent the shock waves as they bounce back and forth between the breech and the shell. I have overlaid on the graph a red line segment whose slope is the 343.2 meter per second speed at which the ignition point propagates through the original bags of propellant. It seems that the initial peak pressure lags the ignition point at about one half of this speed.



In order to determine if the use of artificial viscosity has a significant effect on the results obtained from the numerical integration, I made another run (Case #3). The only difference from Case #2 was that I increased the amount of artificial viscosity used, by setting the first viscosity coefficient to fifty, $C_1 = 50$. The following graph shows the pressures and shell speed with respect to time. These curves are almost identical to those obtained with $C_1 = 5$. Conclusion: artificial viscosity does not have any meaningful impact on the results of the integration.



Averaging pressure over surface area and time

I have stated already that the pressures being calculated are unrealistically high. I have also stated my belief that this occurs because the propellant grains for this gun should not have seven holes, but some lesser number, perhaps four or six. Before re-doing the analysis for a lesser number of holes, though, I want to look into a different issue.

The pressure-data for this gun that I have been looking at was recorded shortly after World War II. I suspect that the pressure sensors used were some type of mechanical analogue devices. I have no idea how quickly they could respond to rapid changes in pressure, such as the passage of a shock wave. Like all sensors, their use would have introduced some degree of averaging over time and averaging over area (the size of the sensor's intake port).

In this section, I want to take a look at how "big", physically and temporally, the peak pressures in the graphs above really are.

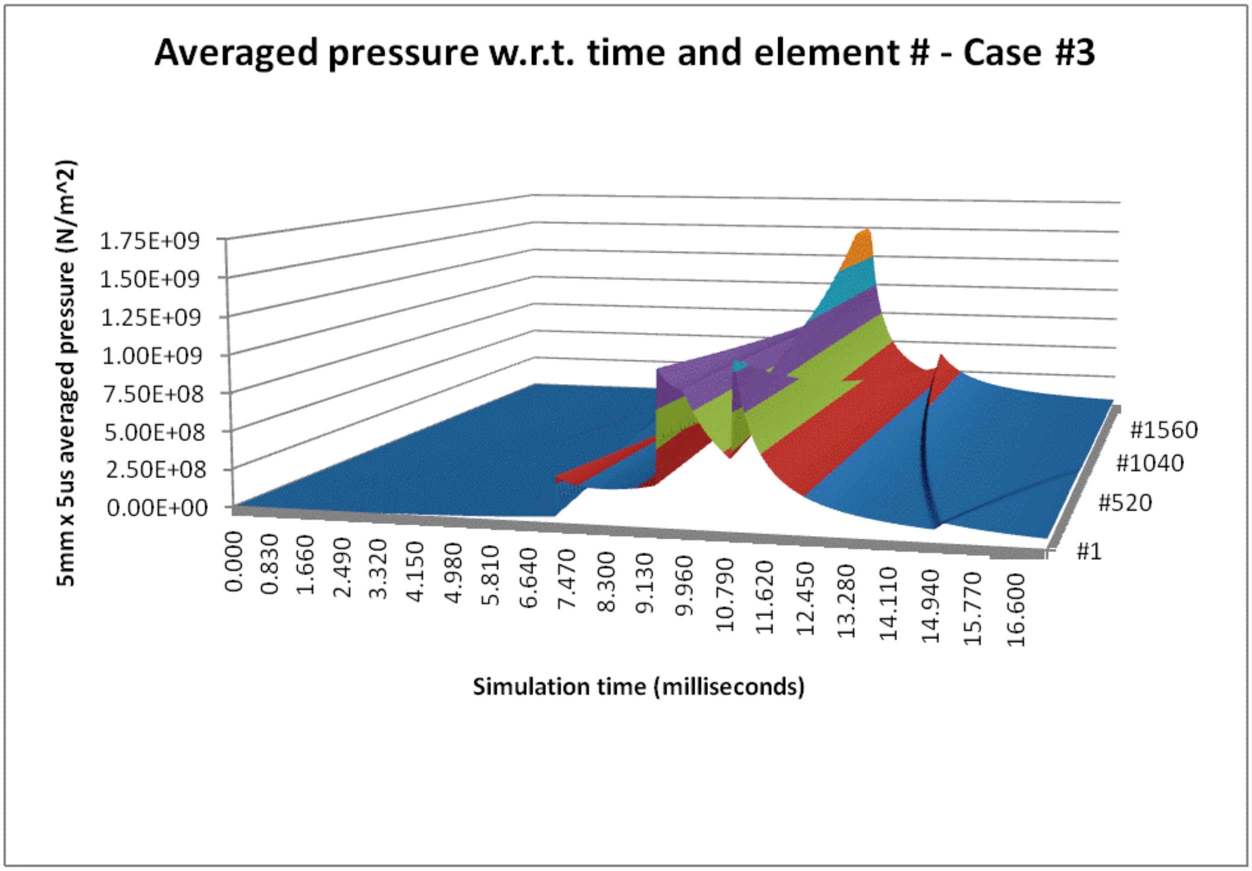
Discretization of distance
In the numerical simulation, the chamber was discretized into 2000 elements. Since the chamber is approximately one meter long, the elements start off being one-half millimeter long. By the time the shell reaches the end of the barrel, the active length of the gas region is about seven meters and the average element length has increased to about 3½ millimeters. The highest pressures are not experienced when the shell is in either of these extreme locations; it is experienced when the shell is between a third and a half of the way down the barrel, when the element lengths are about two millimeters.

Discretization of time
In the numerical simulation, the integration time steps are variable. They are shortest at times when the rates-of-change of pressure are highest, which happen to be the very times when the pressures are passing through their highest peak values. The results of the simulation show that, a those times, the time step gets down to 100 picoseconds.

What we can conclude from the discretization parameters is that the peak pressures shown in the graphs above represent the highest values which occur within distances of two millimeters and time periods of 100 picoseconds.

Peak pressures measured on such small scales of distance and time could turn out to be localized events, and not representative of the pressures in their neighbourhoods in space and time. In order to gauge how representative the reported peaks are, I ran the Case #3 simulation again, with modifications to report each element's "pressure" in a different way. In the modified version, an element's pressure (call this the "central" element) at any time step is defined as the average of the pressures inside certain elements at certain times. To determine which pressure values would be included in the average, the routine looked backwards in time (from the given time step) by five microseconds and forwards in time by five microseconds. At each time step during that 10 microsecond period, the routine looked through the locations of all 2000 elements. If any element was within a distance of five millimeters of the central element at that time, its pressure was added to the average. In short, the pressure for each element at each time is an average of the pressures which exist within five millimeters and five milliseconds.

The following graph shows the pressures measured in this way with respect to the simulation time and the element's index number.



This is a nice-looking graph; the averaging process highlights the sharpness of the shock waves. The peak pressures are certainly not localized phenomena. They are significant on the scale of a centimeter and ten microseconds – large and long enough to burst a barrel. The peak value shown here, after the averaging process, is just slightly less than 15,000 atmospheres. Compare this to the peak pressure (stated on the usual per-element basis) shown in the previous line graph for Case #3. There, the peak pressure was just slightly more than 15,000 atmospheres. Averaging over time and space did not materially affect the values.

I have attached as Appendix ""A" a listing of the VB2010 Express code used for the Case #2 simulation.

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"

Listing of the VB2010 code for the Case #2 simulation

The program consists of a Windows Forms application (Form1) and three modules: Variables, Procedures and HoleCalculations.

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 = "Holes in the propellant grains"
            FormBorderStyle = Windows.Forms.FormBorderStyle.None
            Size = New Drawing.Size(1000, 700)
            CenterToScreen()
            MinimizeBox = True
            MaximizeBox = True
            FormBorderStyle = Windows.Forms.FormBorderStyle.Fixed3D
            With Me
                Controls.Add(buttonSimulateWithHoles)
                buttonSimulateWithHoles.BringToFront()
                Controls.Add(buttonSimulateNoHoles)
                buttonSimulateNoHoles.BringToFront()
                Controls.Add(buttonExit) : buttonExit.BringToFront()
                Controls.Add(labelResult) : labelResult.BringToFront()
            End With
            Visible = True
            PerformLayout()
            BringToFront()
        End With
    End Sub

'/////////////////////////////////////////////////////////////////////////////////////
'/////////////////////////////////////////////////////////////////////////////////////
'// Controls for MainForm.

    Private WithEvents buttonSimulateWithHoles As New Windows.Forms.Button With _
        {.Size = New Drawing.Size(120, 30), _
         .Location = New Drawing.Point(5, 5), _
         .Text = "Simulate with holes", .TextAlign = ContentAlignment.MiddleCenter}

    Private WithEvents buttonSimulateNoHoles As New Windows.Forms.Button With _
        {.Size = New Drawing.Size(120, 30), _
         .Location = New Drawing.Point(5, 40), _
         .Text = "Simulate - No holes", .TextAlign = ContentAlignment.MiddleCenter}

    Private WithEvents buttonExit As New Windows.Forms.Button With _
        {.Size = New Drawing.Size(120, 30), _
         .Location = New Drawing.Point(5, 75), _
         .Text = "Exit", .TextAlign = ContentAlignment.MiddleCenter}
```

```
    Public labelResult As New Windows.Forms.Label With _
        {.Size = New Drawing.Size(950, 600), _
         .Location = New Drawing.Point(125, 5), _
         .Text = "", .TextAlign = ContentAlignment.TopLeft, .Visible = True}

'/////////////////////////////////////////////////////////////////////////////////
'/////////////////////////////////////////////////////////////////////////////////
'// Handlers for controls for MainForm.

    Private Sub buttonSimulateWithHoles_Click() Handles _
        buttonSimulateWithHoles.MouseClick
        SimulateWithHoles = True
        RunCompleteSimulation()
    End Sub

    Private Sub buttonSimulateNoHoles_Click() Handles _
        buttonSimulateNoHoles.MouseClick
        SimulateWithHoles = False
        RunCompleteSimulation()
    End Sub

    Private Sub buttonExit_Click() Handles buttonExit.MouseClick
        Application.Exit()
    End Sub

End Class
```

Module Variables

```
Option Strict On
Option Explicit On

Public Module Variables

    ' Treatment of holes in grain
    ' Set SimulateWithHoles to False to simulate without holes
    Public SimulateWithHoles As Boolean = True

    ' Simulation parameters:
    '   NE = Number of gas elements
    '   MaxSimTime = Maximum length of simulation, in seconds
    '   deltaT = Initial duration of a time step, in seconds
    '   Time = Simulation time, in seconds
    '   deltaTSave = Time interval between writes to output text files
    '   TimeOfNextSave = Time of next write to output text files
    '   deltaNESave = Spatial interval between gas elements which will be saved
    Public NE As Int32 = 2000
    Public MaxSimTime As Double = 0.05
    Public deltaT As Double = 0.0000001
    Public Time As Double
    Public deltaTSave As Double = 0.00001
    Public TimeOfNextSave As Double
    Public deltaNESave As Int32 = 20

    ' Variables used to calculate the length of a time step
    '   MaxChangeAllowed = If per-step change in Pi() or RHOi() exceeds this, reduce TS
    '   MinChangeAllowed = If per-step change in Pi() and RHOi() are less, increase TS
```

```
'   DecreaseInTS = Fraction to decrease time step if pressure change is too high
'   IncreaseInTS = Fraction to increase time step if pressure change is too low
'   MaxPressure = Maximum of pressure in all elements, for display purposes only
'   MaxPressureIndex = Index of element with maximum pressure, for display only
Public MaxChangeAllowed As Double = 0.001
Public MinChangeAllowed As Double = 0.00025
Public DecreaseInTS As Double = 0.9
Public IncreaseInTS As Double = 1.01
Public MaxPressure As Double
Public MaxPressureIndex As Int32

' Important times:
'   TimeOfShellStart = Time at which the shell starts to move
'   TimeOfShellExit = Time at which the shell leaves the barrel
Public TimeofShellStart As Double
Public TimeOfShellExit As Double

' Physical parameters of the gun:
'   Lchamber = Length of chamber
'   Lbarrel = Shell travel distance
'   Abarrel = Area of open barrel
'   Mshell = Mass of the shell
'   Pengband = Engraving band pressure
Public Lchamber As Double = 1.03        ' meters
Public Lbarrel As Double = 5.97         ' meters
Public Abarrel As Double = 0.0127       ' square meters
Public Mshell As Double = 31.8          ' kilograms
Public Pengband As Double = Val("4E7")  ' Newtons per square meter

' Physical parameters of the propellant:
'   Mcharge = Total mass of propellant
'   Dgrain = Diameter of a grain of propellant
'   Lgrain = Original length of a grain of propellant
'   RHOgrain = Crystalline density of solid propellant
'   RHOLoad = Loading density of solid propellant
'   Q0 = Heat released from burning one kilogram of propellant
Public Mcharge As Double = 8.85         ' kilograms
Public Dgrain As Double = 0.0069        ' meters
Public Lgrain As Double = 0.012         ' meters
Public RHOgrain As Double = 1660        ' kilograms per cubic meter
Public RHOLoad As Double = 680          ' kilograms per cubic meter
Public Q0 As Double = 3430000           ' Joules per kilogram

' Physical parameters for the holes in the grains of propellant:
'   Rgrain = Original radius of a grain of propellant
'   Rholes = Original radius of the holes in the webbing
'   RholesEndPhase1 = Hole radius when grain disintegrates
'   RholesEndPhase2 = Hole radius when inner "triangles" stop burning
'   RholesEndPhase3 = Hole radius when outer "triangles" stop burning
'   LgrainEff = Effective length of one grain, per element
'   AgrainEff = Effective cross-sectional area of one grain, per element
Public Rgrain As Double = Dgrain / 2    ' meters
Public Rholes As Double = Rgrain / 20   ' meters
Public RholesEndPhase1 As Double = (Rgrain + Rholes) / 4
Public RholesEndPhase2 As Double = 2 * RholesEndPhase1 / Math.Sqrt(3)
Public RholesEndPhase3 As Double = RholesEndPhase1 * (28 - (6 * Math.Sqrt(3))) / 13
Public LgrainEff As Double
Public AgrainEff As Double
```

```
' Physical parameters of the gas:
'   StoichRatio = Moles of gas produced per kilogram of propellant
'   Ridc = R, the Ideal Gas Constant
'   C1, C2 = Artificial viscosity coefficients
'   Cfr = Mass compensation coefficient for frictional forces
'   Bcovolume = Basic ideal gas co-volume correction "b"
Public StoichRatio As Double = 40       ' moles per kilogram
Public Rigc As Double = 8.31446         ' Joules per mole-degK
Public C1 As Double = 5                 ' Artificial viscosity coefficient #1
Public C2 As Double = C1 / 10           ' Artificial viscosity coefficient #2
Public Cfr As Double = 0.167            ' Mass compensation for frictional forces
Public Bcovolume As Double = 0.00095    ' cubic meters per kilogram

' Initial conditions of the ambient air inside the barrel:
'   Patm = Atmospheric pressure
'   Tatm = Temperature inside the chamber
'   AirMW = Molecular weight of dry air
'   AirNi = Number of moles of original air inside each element
'   AirMi = Mass of original air inside each element
Public Patm As Double = 101300          ' Newtons per square meter
Public Tatm As Double = 100             ' degC
Public AirMW As Double = 0.02897        ' kilograms per mole
Public AirNi As Double                  ' moles
Public AirMi As Double                  ' kilograms

' Ignition parameters
' IgnitionShockWaveSpeeed = Propagation speed of ignition shock wave
'   Use 343.2 m/s for ignition wave
'   Use Val("+1E+20") for simultaneous start all along chamber
Public IgnitionShockWaveSpeed As Double = 343.2

' Gas element variables:
' Values for all elements and for two consecutive time steps are stored
' in the following variables.
'   MiT = Total mass of propellant inside element #i (constant)
'   MiP0 = Original mass of propellant inside element #i
'   MiP(NE) = Mass of unburned propellant inside element #i at time T
'   MiG(NE) = Mass of propellant gas inside element #i at time T
'   Ni(NE) = Total moles of gas inside element #i at time T
'   Xi(NE) = Location of boundary faces at time T (At breech, Xi(0)=0 always)
'   Vi(NE) = Speed of boundary faces at time T-deltaT/2 (Vi(0)= 0 always)
'   Pi(NE) = Static pressure inside element #i at time T
'   Wi(NE) = Artificial viscosity inside element #i at time T
'   VOLi(NE) = Total volume of element #i at time T
'   VOLGasi(NE) = Gas volume of element #i at time T
'   BCoVoli(NE) = Co-volume correction inside element #i at time T
'   Ti(NE) = Temperature inside element #i at time T
'   Ui(NE) = Internal energy of element #i at time T
'   RHOi(NE) = Density inside element #i at time T
'   GAMMAi(NE) = Ratio of specific heats inside element #i at time T
'   Si(NE) = Speed of sound inside element #i at time T
'   deltaQi(NE) = Heat added to element #i during one time step
'   deltaNi(NE) = Moles of gas added to element #i during one time step
Public MiT As Double                                ' kilograms
Public MiP0 As Double                               ' kilograms
Public MiP(NE), MiP_previous(NE) As Double          ' kilograms
Public MiG(NE), MiG_previous(NE) As Double          ' kilograms
Public Ni(NE), Ni_previous(NE) As Double            ' moles
```

```
Public Xi(NE), Xi_previous(NE) As Double            ' meters
Public Vi(NE), Vi_previous(NE) As Double            ' meters per second
Public Pi(NE), Pi_previous(NE) As Double            ' Newtons per square meter
Public Wi(NE), Wi_previous(NE) As Double            ' Newtons per square meter
Public VOLi(NE), VOLi_previous(NE) As Double        ' cubic meters
Public VOLGasi(NE), VOLGasi_previous(NE) As Double  ' cubic meters
Public BCoVoli(NE), BCoVoli_previous(NE) As Double  ' cubic meters
Public Ti(NE), Ti_previous(NE) As Double            ' degK
Public Ui(NE), Ui_previous(NE) As Double            ' Joules
Public RHOi(NE), RHOi_previous(NE) As Double        ' kilograms per cubic meter
Public GAMMAi(NE), GAMMAi_previous(NE) As Double
Public Si(NE), Si_previous(NE) As Double            ' meters per second
Public deltaQi(NE), deltaQi_previous(NE) As Double  ' Joules
Public deltaNi(NE), deltaNi_previous(NE) As Double  ' moles

' Propellant variables:
' Values for all elements and for two consecutive time steps are stored in the
' following variables.
'   Bi(NE) = Burn rate inside element #i at time T
'   Rgraini(NE) = Radius of grains in element #i at time T
'   Rholesi(NE) = Radius of holes in element #i at time T
'   PropVOLi(NE) = Volume of propellant in element #i at time T
'   deltaMi(NE) = Mass of propellant burned during one time step
'   fi(NE) = Fraction of propellant burned in element #i at time T
Public Bi(NE), Bi_previous(NE) As Double             ' meters per second
Public Rgraini(NE), Rgraini_previous(NE) As Double   ' meters
Public Rholesi(NE), Rholesi_previous(NE) As Double   ' meters
Public PropVOLi(NE), PropVOLi_previous(NE) As Double ' cubic meters
Public deltaMi(NE), deltaMi_previous(NE) As Double   ' kilograms
Public fi(NE), fi_previous(NE) As Double

' Projectile variables:
' Values for the projectile for two consecutive time steps are stored in the
' following variables.
'   Pshell = Pressure on rear face of shell at time T
'   Xshell = Location of rear face of shell at time T
'   Vshell = Speed of shell at time T-deltaT/2
'   ACCshell = Acceleration of shell at time T
'   CanShellMove is True when the shell has broken free from its band
Public Pshell, Pshell_previous As Double            ' Newtons per square meter
Public Xshell, Xshell_previous As Double            ' meters
Public Vshell, Vshell_previous As Double            ' meters per second
Public ACCshell, ACCshell_previous As Double        ' meters per second^2
Public CanShellMove As Boolean

' Text file processing
Public ThisDirectory As String = FileSystem.CurDir.ToString
Public TextOutputFileName As String = "Naval_gun_simulation_"
' Log of screen display
Public FileWriterMaster As System.IO.StreamWriter
' Gas element variables
Public FileWriterMiP As System.IO.StreamWriter
Public FileWriterMiG As System.IO.StreamWriter
Public FileWriterNi As System.IO.StreamWriter
Public FileWriterXi As System.IO.StreamWriter
Public FileWriterVi As System.IO.StreamWriter
Public FileWriterPi As System.IO.StreamWriter
Public FileWriterWi As System.IO.StreamWriter
```

```
    Public FileWriterVOLi As System.IO.StreamWriter
    Public FileWriterVOLGasi As System.IO.StreamWriter
    Public FileWriterBCoVoli As System.IO.StreamWriter
    Public FileWriterTi As System.IO.StreamWriter
    Public FileWriterUi As System.IO.StreamWriter
    Public FileWriterRHOi As System.IO.StreamWriter
    Public FileWriterGAMMAi As System.IO.StreamWriter
    Public FileWriterSi As System.IO.StreamWriter
    Public FileWriterdeltaQi As System.IO.StreamWriter
    Public FileWriterdeltaNi As System.IO.StreamWriter
    ' Propellant variables
    Public FileWriterBi As System.IO.StreamWriter
    Public FileWriterRgraini As System.IO.StreamWriter
    Public FileWriterRholesi As System.IO.StreamWriter
    Public FileWriterPropVOLi As System.IO.StreamWriter
    Public FileWriterdeltaMi As System.IO.StreamWriter
    Public FileWriterfi As System.IO.StreamWriter
    ' Projectile variables
    Public FileWriterShell As System.IO.StreamWriter

End Module
```

Module Procedures

```
Option Strict On
Option Explicit On

' List of subroutines:
'   InitializeForSimulation()
'   RunFullSimulation()
'   ExecuteOneTimeStep()
'   ShiftAllValuesToPreviousVariables()
'   FindMaximumPressure()
'   OpenAllOutputTextFiles()
'   CloseAllOutputTextFiles()
'   WriteHeadersToAllOutputTextFiles()
'   WriteDataRowToAllOutputTextFiles()

Public Module Procedures

    Public Sub InitializeForSimulation()
        ' This subroutine sets all quantities to their initial values, so
        ' everything is ready to start the first time step.
        '
        ' -------------------------------------------------------
        ' Deal with the ambient air in the chamber prior to firing
        Dim TotalVolumeOfAir As Double = _
            Mcharge * ((1 / RHOLoad) - (1 / RHOgrain))
        Dim TotalMolesOfAir As Double = _
            Patm * TotalVolumeOfAir / (Rigc * (Tatm + 273.15))
        Dim TotalMassOfAir As Double = AirMW * TotalMolesOfAir
        AirNi = TotalMolesOfAir / NE
        AirMi = TotalMassOfAir / NE
        ' -------------------------------------------------------
        '
        ' Set the initial MASS of propellant inside each element
        MiP0 = Mcharge / NE
```

```
    MiT = MiP0 + AirMi
    For J As Int32 = 1 To NE Step 1
        MiP(J) = MiP0
        MiG(J) = 0
    Next J
    '
    ' Set the initial VOLUME of propellant inside each element
    For J As Int32 = 1 To NE Step 1
        PropVOLi(J) = MiP(J) / RHOgrain
        Rgraini(J) = Rgrain
        If (SimulateWithHoles = True) Then
            Rholesi(J) = Rholes
        Else
            Rholesi(J) = 0
        End If
    Next J
    '
    ' Set the effective area and length of grains in each element
    ' This is done for the sole purpose of calculating the volume of propellant
    If (SimulateWithHoles = True) Then
        AgrainEff = Math.PI * ((Rgrain ^ 2) - (7 * (Rholes ^ 2)))
    Else
        AgrainEff = Math.PI * (Rgrain ^ 2)
    End If
    LgrainEff = ((Mcharge / NE) / RHOgrain) / AgrainEff
    '
    ' Set the unburned fractions
    For J As Int32 = 1 To NE Step 1
        fi(J) = 1
    Next J
    '
    ' Set initial locations of boundary faces
    Xi(0) = 0
    For J As Int32 = 1 To NE Step 1
        Xi(J) = Lchamber * J / NE
    Next J
    Xshell = Lchamber
    '
    ' Set the initial speeds
    For J As Int32 = 1 To NE Step 1
        Vi(J) = 0
    Next J
    Vshell = 0
    '
    ' Set the initial pressure, volume, density and number of moles of air
    For J As Int32 = 1 To NE Step 1
        Pi(J) = Patm
        VOLi(J) = Abarrel * (Xi(J) - Xi(J - 1))
        RHOi(J) = AirMi / (VOLi(J) - PropVOLi(J))
        Ni(J) = AirNi
    Next J
    '
    ' Set the internal energies
    For J As Int32 = 1 To NE Step 1
        Ui(J) = (Ni(J) * Rigc * (Tatm + 273.15)) / (1.333 - 1)
    Next J
End Sub
```

```
Public Sub RunCompleteSimulation()
    Form1.labelResult.Text = "Starting simulation ..."
    Form1.labelResult.Refresh()
    '
    ' Initialize the vectors
    InitializeForSimulation()
    '
    ' Open the text output files
    OpenAllOutputTextFiles()
    WriteHeadersToAllOutputTextFiles()
    '
    ' Save the starting values
    Time = 0
    WriteDataRowToAllOutputTextFiles()
    TimeOfNextSave = Time + deltaTSave
    '
    ' Set the control parameters
    CanShellMove = False
    Dim ScreenUpdateInterval As Int32 = 250
    Dim ScreenUpdateCounter As Int32 = 250
    Do
        Time = Time + deltaT
        ' Test to see if the simulation has timed out
        If (Time >= MaxSimTime) Then
            Exit Do
        End If
        ' Shift the present variables into their "previous" variants
        ShiftAllValuesToPreviousVariables()
        ' Run one time step
        ExecuteOneTimeStep()
        ' Find the change in maximum pressure
        FindMaximumPressure(MaxPressure, MaxPressureIndex)
        ' Test to see if it is time to write to the output text files
        If (Time >= TimeOfNextSave) Then
            WriteDataRowToAllOutputTextFiles()
            TimeOfNextSave = Time + deltaTSave
        End If
        ' Test to see if the shell can start moving
        If ((CanShellMove = False) And _
            (Pi(NE) >= Pengband)) Then
            CanShellMove = True
            TimeofShellStart = Time
            Form1.labelResult.Text = Strings.Right( _
                Form1.labelResult.Text & vbCrLf & _
                "Shell starts moving at time = " & Trim(Str(Time)), 6000)
            Form1.labelResult.Refresh()
            FileWriterMaster.WriteLine( _
                "Shell starts moving at time = " & Trim(Str(Time)))
            Threading.Thread.Sleep(2000)
        End If
        ' Test to see if the shell has left the barrel
        If (Xshell > (Lchamber + Lbarrel)) Then
            TimeOfShellExit = Time
            Form1.labelResult.Text = Strings.Right( _
                Form1.labelResult.Text & vbCrLf & _
                "Shell exits at time = " & Trim(Str(Time)), 6000)
            Form1.labelResult.Refresh()
            FileWriterMaster.WriteLine( _
```

```
            "Shell exits at time = " & Trim(Str(Time)))
        ' Write the final values to the output text files
        WriteDataRowToAllOutputTextFiles()
        Exit Do
    End If
    ' Update the screen display
    ScreenUpdateCounter = ScreenUpdateCounter + 1
    If (ScreenUpdateCounter >= ScreenUpdateInterval) Then
        Form1.labelResult.Text = Strings.Right( _
            Form1.labelResult.Text & vbCrLf & _
            "T(us)= " & FormatNumber(Time * 1000000, 2) & _
            "  dT(us)= " & FormatNumber(deltaT * 1000000, 6) & _
            "  MaxP= " & FormatNumber(MaxPressure, 0, , TriState.True) & _
            "  MaxPindex= " & Trim(Str(MaxPressureIndex)) & _
            "  P1= " & FormatNumber(Pi(1), 0, , , TriState.True) & _
            "  Pne= " & FormatNumber(Pi(NE), 0, , , TriState.True) & _
            "  Bne= " & FormatNumber(Bi(NE), 4) & _
            "  fne= " & FormatNumber(fi(NE), 5) & _
            "  f1= " & FormatNumber(fi(1), 5) & _
            "  Vs= " & FormatNumber(Vshell, 2) & _
            "  Wne= " & FormatNumber(Wi(NE), 0, , , TriState.True), _
            6000)
        Form1.labelResult.Refresh()
        ' Save the new line to the Master text output file as well
        FileWriterMaster.WriteLine( _
            "Time= ," & Trim(Str(Time)) & _
            ", with deltaT= ," & Trim(Str(deltaT)) & _
            ", MaxP= ," & Trim(Str(MaxPressure)) & _
            ", MaxPIndex= ," & Trim(Str(MaxPressureIndex)) & _
            ", P1= ," & Trim(Str(Pi(1))) & _
            ", Pne= ," & Trim(Str(Pi(NE))) & _
            ", Wne= ," & Trim(Str(Wi(NE))) & _
            ", Une= ," & Trim(Str(Ui(NE))) & _
            ", Bne= ," & Trim(Str(Bi(NE))) & _
            ", fne= ," & Trim(Str(fi(NE))) & _
            ", f1= ," & Trim(Str(fi(1))) & _
            ", Xs= ," & Trim(Str(Xshell)) & _
            ", Vs= ," & Trim(Str(Vshell)))
        ScreenUpdateCounter = 0
    End If
    ' Based on change in maximum pressure, change the time step if necessary
    ' Equation (U)
    Dim lRelChangeInP As Double
    Dim lMaxRelChangeInP As Double
    Dim lRelChangeInRHO As Double
    Dim lMaxRelChangeInRHO As Double
    lMaxRelChangeInP = Val("-1E+20")
    lMaxRelChangeInRHO = Val("-1E+20")
    For J As Int32 = 1 To NE Step 1
        lRelChangeInP = _
            Math.Abs((Pi(J) - Pi_previous(J)) / Pi_previous(J))
        lRelChangeInRHO = _
            Math.Abs((RHOi(J) - RHOi_previous(J)) / RHOi_previous(J))
        If (lRelChangeInP > lMaxRelChangeInP) Then
            lMaxRelChangeInP = lRelChangeInP
        End If
        If (lRelChangeInRHO > lMaxRelChangeInRHO) Then
            lMaxRelChangeInRHO = lRelChangeInRHO
```

```
            End If
        Next J
        If ((lMaxRelChangeInP > MaxChangeAllowed) Or _
            (lMaxRelChangeInRHO > MaxChangeAllowed)) Then
            deltaT = DecreaseInTS * deltaT
        End If
        If ((lMaxRelChangeInP < MinChangeAllowed) And _
            (lMaxRelChangeInRHO < MinChangeAllowed)) Then
            deltaT = IncreaseInTS * deltaT
        End If
    Loop
    ' Close the output text file
    CloseAllOutputTextFiles()
End Sub

Public Sub ExecuteOneTimeStep()
    ' This subroutine advances the simulation from time t to time t+deltaT once the
    ' speeds of the boundary faces at time t+deltaT/2 have been calculated.
    ' For example, Vi(j) is the speed of boundary face #j at time t+deltaT/2
    '              Pi(j) is the pressure in element #j at time t+deltaT
    ' If the Boolean flag CanShellMove is False, then the shell, and the RHS
    ' boundary face of element #NE, are not permitted to move.
    '
    ' Step #1: Equation (A)
    Xi(0) = 0
    For J As Int32 = 1 To (NE - 1) Step 1
        Xi(J) = Xi_previous(J) + (Vi(J) * deltaT)
    Next J
    If (CanShellMove = True) Then
        Xi(NE) = Xi_previous(NE) + (Vshell * deltaT)
    Else
        Xi(NE) = Xi_previous(NE)
    End If
    Xshell = Xi(NE)
    '
    ' Step #2: Equation (B)
    For J As Int32 = 1 To NE Step 1
        VOLi(J) = (Xi(J) - Xi(J - 1)) * Abarrel
    Next J
    '
    ' Step #3: Burning equations
    For J As Int32 = 1 To NE Step 1
        If ((IgnitionShockWaveSpeed * Time) >= Xi_previous(J)) Then
            ' Equation (C)
            Bi(J) = Math.Exp(( _
                (0.046696597 * ((Math.Log(Pi_previous(J) / Patm)) ^ 2)) + _
                (0.34808898 * Math.Log(Pi_previous(J) / Patm)) + _
                -0.572295873)) / 1000
        Else
            Bi(J) = 0
        End If

        ' ///////////////////////////////////////////////////////////////////////////
        ' // Use different routines to calculate the mass of propellant burned
        ' ///////////////////////////////////////////////////////////////////////////
        If (SimulateWithHoles = True) Then
            VolumeOfPropellantBurned_WithHoles( _
                Rgraini_previous(J), _
```

```
                Rholesi_previous(J), _
                Bi(J), _
                deltaT, _
                Rgraini(J), _
                Rholesi(J), _
                PropVOLi(J))
        Else
            VolumeOfPropellantBurned_NoHoles( _
                Rgraini_previous(J), _
                Bi(J), _
                deltaT, _
                Rgraini(J), _
                PropVOLi(J))
            Rholesi(J) = 0
        End If

        Dim DecreaseInVolume As Double = PropVOLi_previous(J) - PropVOLi(J)
        deltaMi(J) = RHOgrain * DecreaseInVolume
        '
        ' Check that burning does not consume more than 100% of the remaining mass
        If (deltaMi(J) > MiP_previous(J)) Then
            Rgraini(J) = 0
            PropVOLi(J) = 0
            deltaMi(J) = MiP_previous(J)
        End If
        ' Equation (E)
        MiP(J) = MiP_previous(J) - deltaMi(J)
        MiG(J) = MiG_previous(J) + deltaMi(J)
        fi(J) = MiG(J) / MiP0
    Next J
    '
    ' Step #4: Equation (F)
    For J As Int32 = 1 To NE Step 1
        deltaQi(J) = Q0 * deltaMi(J)
    Next J
    '
    ' Step #5: Density
    For J As Int32 = 1 To NE Step 1
        'Equation (G)
        VOLGasi(J) = VOLi(J) - (MiP(J) / RHOgrain)
        ' Equation (H)
        RHOi(J) = (MiG(J) + AirMi) / VOLGasi(J)
    Next J
    '
    ' Step #6: Co-volume correction
    For J As Int32 = 1 To NE Step 1
        ' Equation (I)
        BCoVoli(J) = Bcovolume * (MiG(J) + AirMi) / (1 + (2 * RHOi(J) / 500))
        ' Equation (J)
        deltaNi(J) = 40 * deltaMi(J)
        ' Equation (K)
        Ni(J) = Ni_previous(J) + deltaNi(J)
    Next J
    '
    ' Step #7: Adiabatic Index
    For J As Int32 = 1 To NE Step 1
        ' Equation (L)
        GAMMAi(J) = 1.333 + (0.567 * RHOi(J) / 1200)
```

```
Next J
'
' Step #8: Internal energy and pressure
For J As Int32 = 1 To NE Step 1
    Dim MaxNumOfIterations As Double = 1000
    Dim NumOfIterations As Double = 0
    ' Starting guess
    Pi(J) = Pi_previous(J)
    ' Loop
    Do
        ' Equation (M)
        Si(J) = Math.Sqrt(GAMMAi(J) * Pi(J) / RHOi(J))
        ' Carry out the relative speed test
        Dim deltaV As Double
        If (J = 1) Then
            deltaV = -Vi(J)
        Else
            deltaV = Vi(J - 1) - Vi(J)
        End If
        ' Equation (N)
        If (deltaV > 0) Then
            Wi(J) = RHOi(J) * ( _
                (C1 * deltaV * deltaV) + _
                (C2 * Si(J) * Math.Abs(deltaV)))
        Else
            Wi(J) = 0
        End If
        ' Equation (O)
        Dim lFactor1 As Double
        Dim lFactor2 As Double
        lFactor1 = Pi(J) + Wi(J) + Pi_previous(J) + Wi_previous(J)
        lFactor2 = VOLi(J) - VOLi_previous(J)
        Ui(J) = Ui_previous(J) + deltaQi(J) - (0.5 * lFactor1 * lFactor2)
        ' Equation (P)
        Dim lFactor3 As Double
        Dim P_New As Double
        lFactor3 = VOLGasi(J) - BCoVoli(J)
        P_New = (GAMMAi(J) - 1) * Ui(J) / lFactor3
        ' Test for negative pressure
        If (P_New <= 0) Then
            MsgBox("Error: Negative pressure.  Reduce time step.")
            Exit Sub
        End If
        ' Test for convergence
        If (Math.Abs((P_New - Pi(J)) / Pi(J)) < 0.000000001) Then
            Pi(J) = P_New
            Exit Do
        End If
        ' Restrict the per-iteration adjustment in pressure
        Dim MaxAbsChange As Double
        MaxAbsChange = Math.Min(0.1 * Pi(J), Math.Abs(P_New - Pi(J)))
        If (P_New > Pi(J)) Then
            Pi(J) = Pi(J) + MaxAbsChange
        Else
            Pi(J) = Pi(J) - MaxAbsChange
        End If
        ' Test for too many iterations
        NumOfIterations = NumOfIterations + 1
```

```
            If (NumOfIterations > MaxNumOfIterations) Then
                MsgBox("Error: Too many iterations.")
                Exit Sub
            End If
        Loop
    Next J
    '
    ' Step #9: Temperature
    ' Equation (Q)
    For J As Int32 = 1 To NE Step 1
        Ti(J) = Pi(J) * (VOLGasi(J) - BCoVoli(J)) / (Ni(J) * Rigc)
    Next J
    '
    ' Step #10: Acceleration of the shell
    ' Equation (R)
    Dim EffectiveMshell As Double = Mshell / (1 - Cfr)
    Pshell = Pi(NE)
    ACCshell = (Pshell - Patm) * Abarrel / EffectiveMshell
    '
    ' Step #11: Advance the speeds
    ' Equation (S)
    For J As Int32 = 1 To (NE - 1) Step 1
        Dim lFactor1 As Double
        lFactor1 = Pi(J + 1) + Wi(J + 1) - Pi(J) - Wi(J)
        Vi(J) = Vi_previous(J) - (deltaT * lFactor1 * Abarrel / MiT)
    Next J
    ' Equation (T)
    If (CanShellMove = True) Then
        Vshell = Vshell_previous + (ACCshell * deltaT)
    Else
        Vshell = 0
    End If
    Vi(NE) = Vshell
End Sub

Public Sub ShiftAllValuesToPreviousVariables()
    ' This subroutine is called at the end of every time step.  It moves the values
    ' just calculated into their "previous" variants, in preparation for the next
    ' time step.
    Xi_previous(0) = Xi(0)
    For J As Int32 = 1 To NE Step 1
        ' Gas element variables
        MiP_previous(J) = MiP(J)
        MiG_previous(J) = MiG(J)
        Ni_previous(J) = Ni(J)
        Xi_previous(J) = Xi(J)
        Vi_previous(J) = Vi(J)
        Pi_previous(J) = Pi(J)
        Wi_previous(J) = Wi(J)
        VOLi_previous(J) = VOLi(J)
        VOLGasi_previous(J) = VOLGasi(J)
        BCoVoli_previous(J) = BCoVoli(J)
        Ti_previous(J) = Ti(J)
        Ui_previous(J) = Ui(J)
        RHOi_previous(J) = RHOi(J)
        GAMMAi_previous(J) = GAMMAi(J)
        Si_previous(J) = Si(J)
        deltaQi_previous(J) = deltaQi(J)
```

```
            deltaNi_previous(J) = deltaNi(J)
            ' Propellant variables
            Bi_previous(J) = Bi(J)
            Rgraini_previous(J) = Rgraini(J)
            Rholesi_previous(J) = Rholesi(J)
            PropVOLi_previous(J) = PropVOLi(J)
            deltaMi_previous(J) = deltaMi(J)
            fi_previous(J) = fi(J)
        Next J
        ' Projectile variables
        Pshell_previous = Pshell
        Xshell_previous = Xshell
        Vshell_previous = Vshell
        ACCshell_previous = ACCshell
    End Sub

    Public Sub FindMaximumPressure( _
        ByRef MaxP As Double, ByRef MaxPIndex As Int32)
        ' This function is called at the end of every time step.  It looks through the
        ' current values of the pressure in all elements.  It returns the maximum value
        ' and the index of the element with the maximum pressure.
        MaxP = Val("-1E+20")
        For J As Int32 = 1 To NE Step 1
            If (Pi(J) > MaxP) Then
                MaxP = Pi(J)
                MaxPIndex = J
            End If
        Next J
    End Sub

    Public Sub OpenAllOutputTextFiles()
        ' Log of screen display
        FileWriterMaster = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "Master.txt")
        ' Gas element variables
        FileWriterMiP = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "MiP.txt")
        FileWriterMiG = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "MiG.txt")
        FileWriterNi = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "Ni.txt")
        FileWriterXi = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "Xi.txt")
        FileWriterVi = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "Vi.txt")
        FileWriterPi = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "Pi.txt")
        FileWriterWi = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "Wi.txt")
        FileWriterVOLi = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "VOLi.txt")
        FileWriterVOLGasi = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "VOLGasi.txt")
        FileWriterBCoVoli = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "BCoVoli.txt")
        FileWriterTi = New System.IO.StreamWriter( _
            ThisDirectory & "\" & TextOutputFileName & "Ti.txt")
        FileWriterUi = New System.IO.StreamWriter( _
```

```
        ThisDirectory & "\" & TextOutputFileName & "Ui.txt")
    FileWriterRHOi = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "RHOi.txt")
    FileWriterGAMMAi = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "GAMMAi.txt")
    FileWriterSi = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "Si.txt")
    FileWriterdeltaQi = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "deltaQi.txt")
    FileWriterdeltaNi = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "deltaNi.txt")
    ' Propellant variables
    FileWriterBi = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "Bi.txt")
    FileWriterRgraini = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "Rgraini.txt")
    FileWriterRholesi = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "Rholesi.txt")
    FileWriterPropVOLi = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "PropVOLi.txt")
    FileWriterdeltaMi = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "deltaMi.txt")
    FileWriterfi = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "fi.txt")
    ' Projectile variables
    FileWriterShell = New System.IO.StreamWriter( _
        ThisDirectory & "\" & TextOutputFileName & "Shell.txt")
End Sub

Public Sub CloseAllOutputTextFiles()
    ' Log of screen display
    FileWriterMaster.Close()
    ' Gas element variables
    FileWriterMiP.Close()
    FileWriterMiG.Close()
    FileWriterNi.Close()
    FileWriterXi.Close()
    FileWriterVi.Close()
    FileWriterPi.Close()
    FileWriterWi.Close()
    FileWriterVOLi.Close()
    FileWriterVOLGasi.Close()
    FileWriterBCoVoli.Close()
    FileWriterTi.Close()
    FileWriterUi.Close()
    FileWriterRHOi.Close()
    FileWriterGAMMAi.Close()
    FileWriterSi.Close()
    FileWriterdeltaQi.Close()
    FileWriterdeltaNi.Close()
    ' Propellant variables
    FileWriterBi.Close()
    FileWriterRgraini.Close()
    FileWriterRholesi.Close()
    FileWriterPropVOLi.Close()
    FileWriterdeltaMi.Close()
    FileWriterfi.Close()
    ' Projectile variables
```

```
    FileWriterShell.Close()
End Sub

Public Sub WriteHeadersToAllOutputTextFiles()
    ' Log of screen display
    FileWriterMiP.Write("Time, ")
    ' Gas element variables
    FileWriterMiG.Write("Time, ")
    FileWriterNi.Write("Time, ")
    FileWriterXi.Write("Time, ")
    FileWriterVi.Write("Time, ")
    FileWriterPi.Write("Time, ")
    FileWriterWi.Write("Time, ")
    FileWriterVOLi.Write("Time, ")
    FileWriterVOLGasi.Write("Time, ")
    FileWriterBCoVoli.Write("Time, ")
    FileWriterTi.Write("Time, ")
    FileWriterUi.Write("Time, ")
    FileWriterRHOi.Write("Time, ")
    FileWriterGAMMAi.Write("Time, ")
    FileWriterSi.Write("Time, ")
    FileWriterdeltaQi.Write("Time, ")
    FileWriterdeltaNi.Write("Time, ")
    ' Propellant variables
    FileWriterBi.Write("Time, ")
    FileWriterRgraini.Write("Time, ")
    FileWriterRholesi.Write("Time, ")
    FileWriterPropVOLi.Write("Time, ")
    FileWriterdeltaMi.Write("Time, ")
    FileWriterfi.Write("Time, ")
    ' Projectile variables
    FileWriterShell.WriteLine("Time, Xshell, Vshell, ACCshell, Pshell")
    For K As Int32 = 0 To (NE - 1) Step deltaNESave
        Dim ElementIndex As Int32
        If (K = 0) Then
            ElementIndex = 1
        Else
            ElementIndex = K
        End If
        ' Gas element variables
        FileWriterMiP.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterMiG.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterNi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterXi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterVi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterPi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterWi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterVOLi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterVOLGasi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterBCoVoli.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterTi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterUi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterRHOi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterGAMMAi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterSi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterdeltaQi.Write(Trim(Str(ElementIndex)) & ", ")
        FileWriterdeltaNi.Write(Trim(Str(ElementIndex)) & ", ")
        ' Propellant variables
```

```
            FileWriterBi.Write(Trim(Str(ElementIndex)) & ", ")
            FileWriterRgraini.Write(Trim(Str(ElementIndex)) & ", ")
            FileWriterRholesi.Write(Trim(Str(ElementIndex)) & ", ")
            FileWriterPropVOLi.Write(Trim(Str(ElementIndex)) & ", ")
            FileWriterdeltaMi.Write(Trim(Str(ElementIndex)) & ", ")
            FileWriterfi.Write(Trim(Str(ElementIndex)) & ", ")
        Next K
        ' Gas element variables
        FileWriterMiP.WriteLine(Trim(Str(NE)))
        FileWriterMiG.WriteLine(Trim(Str(NE)))
        FileWriterNi.WriteLine(Trim(Str(NE)))
        FileWriterXi.WriteLine(Trim(Str(NE)))
        FileWriterVi.WriteLine(Trim(Str(NE)))
        FileWriterPi.WriteLine(Trim(Str(NE)))
        FileWriterWi.WriteLine(Trim(Str(NE)))
        FileWriterVOLi.WriteLine(Trim(Str(NE)))
        FileWriterVOLGasi.WriteLine(Trim(Str(NE)))
        FileWriterBCoVoli.WriteLine(Trim(Str(NE)))
        FileWriterTi.WriteLine(Trim(Str(NE)))
        FileWriterUi.WriteLine(Trim(Str(NE)))
        FileWriterRHOi.WriteLine(Trim(Str(NE)))
        FileWriterGAMMAi.WriteLine(Trim(Str(NE)))
        FileWriterSi.WriteLine(Trim(Str(NE)))
        FileWriterdeltaQi.WriteLine(Trim(Str(NE)))
        FileWriterdeltaNi.WriteLine(Trim(Str(NE)))
        ' Propellant variables
        FileWriterBi.WriteLine(Trim(Str(NE)))
        FileWriterRgraini.WriteLine(Trim(Str(NE)))
        FileWriterRholesi.WriteLine(Trim(Str(NE)))
        FileWriterPropVOLi.WriteLine(Trim(Str(NE)))
        FileWriterdeltaMi.WriteLine(Trim(Str(NE)))
        FileWriterfi.WriteLine(Trim(Str(NE)))
    End Sub

    Public Sub WriteDataRowToAllOutputTextFiles()
        ' Gas element variables
        FileWriterMiP.Write(Trim(Str(Time)) & ", ")
        FileWriterMiG.Write(Trim(Str(Time)) & ", ")
        FileWriterNi.Write(Trim(Str(Time)) & ", ")
        FileWriterXi.Write(Trim(Str(Time)) & ", ")
        FileWriterVi.Write(Trim(Str(Time)) & ", ")
        FileWriterPi.Write(Trim(Str(Time)) & ", ")
        FileWriterWi.Write(Trim(Str(Time)) & ", ")
        FileWriterVOLi.Write(Trim(Str(Time)) & ", ")
        FileWriterVOLGasi.Write(Trim(Str(Time)) & ", ")
        FileWriterBCoVoli.Write(Trim(Str(Time)) & ", ")
        FileWriterTi.Write(Trim(Str(Time)) & ", ")
        FileWriterUi.Write(Trim(Str(Time)) & ", ")
        FileWriterRHOi.Write(Trim(Str(Time)) & ", ")
        FileWriterGAMMAi.Write(Trim(Str(Time)) & ", ")
        FileWriterSi.Write(Trim(Str(Time)) & ", ")
        FileWriterdeltaQi.Write(Trim(Str(Time)) & ", ")
        FileWriterdeltaNi.Write(Trim(Str(Time)) & ", ")
        ' Propellant variables
        FileWriterBi.Write(Trim(Str(Time)) & ", ")
        FileWriterRgraini.Write(Trim(Str(Time)) & ", ")
        FileWriterRholesi.Write(Trim(Str(Time)) & ", ")
        FileWriterPropVOLi.Write(Trim(Str(Time)) & ", ")
```

```
FileWriterdeltaMi.Write(Trim(Str(Time)) & ", ")
FileWriterfi.Write(Trim(Str(Time)) & ", ")
For K As Int32 = 0 To (NE - 1) Step deltaNESave
    Dim ElementIndex As Int32
    If (K = 0) Then
        ElementIndex = 1
    Else
        ElementIndex = K
    End If
    ' Gas element variables
    FileWriterMiP.Write(Trim(Str(MiP(ElementIndex))) & ", ")
    FileWriterMiG.Write(Trim(Str(MiG(ElementIndex))) & ", ")
    FileWriterNi.Write(Trim(Str(Ni(ElementIndex))) & ", ")
    FileWriterXi.Write(Trim(Str(Xi(ElementIndex))) & ", ")
    FileWriterVi.Write(Trim(Str(Vi(ElementIndex))) & ", ")
    FileWriterPi.Write(Trim(Str(Pi(ElementIndex))) & ", ")
    FileWriterWi.Write(Trim(Str(Wi(ElementIndex))) & ", ")
    FileWriterVOLi.Write(Trim(Str(VOLi(ElementIndex))) & ", ")
    FileWriterVOLGasi.Write(Trim(Str(VOLGasi(ElementIndex))) & ", ")
    FileWriterBCoVoli.Write(Trim(Str(BCoVoli(ElementIndex))) & ", ")
    FileWriterTi.Write(Trim(Str(Ti(ElementIndex))) & ", ")
    FileWriterUi.Write(Trim(Str(Ui(ElementIndex))) & ", ")
    FileWriterRHOi.Write(Trim(Str(RHOi(ElementIndex))) & ", ")
    FileWriterGAMMAi.Write(Trim(Str(GAMMAi(ElementIndex))) & ", ")
    FileWriterSi.Write(Trim(Str(Si(ElementIndex))) & ", ")
    FileWriterdeltaQi.Write(Trim(Str(deltaQi(ElementIndex))) & ", ")
    FileWriterdeltaNi.Write(Trim(Str(deltaNi(ElementIndex))) & ", ")
    ' Propellant variables
    FileWriterBi.Write(Trim(Str(Bi(ElementIndex))) & ", ")
    FileWriterRgraini.Write(Trim(Str(Rgraini(ElementIndex))) & ", ")
    FileWriterRholesi.Write(Trim(Str(Rholesi(ElementIndex))) & ", ")
    FileWriterPropVOLi.Write(Trim(Str(PropVOLi(ElementIndex))) & ", ")
    FileWriterdeltaMi.Write(Trim(Str(deltaMi(ElementIndex))) & ", ")
    FileWriterfi.Write(Trim(Str(fi(ElementIndex))) & ", ")
Next K
' Gas element variables
FileWriterMiP.WriteLine(Trim(Str(MiP(NE))))
FileWriterMiG.WriteLine(Trim(Str(MiG(NE))))
FileWriterNi.WriteLine(Trim(Str(Ni(NE))))
FileWriterXi.WriteLine(Trim(Str(Xi(NE))))
FileWriterVi.WriteLine(Trim(Str(Vi(NE))))
FileWriterPi.WriteLine(Trim(Str(Pi(NE))))
FileWriterWi.WriteLine(Trim(Str(Wi(NE))))
FileWriterVOLi.WriteLine(Trim(Str(VOLi(NE))))
FileWriterVOLGasi.WriteLine(Trim(Str(VOLGasi(NE))))
FileWriterBCoVoli.WriteLine(Trim(Str(BCoVoli(NE))))
FileWriterTi.WriteLine(Trim(Str(Ti(NE))))
FileWriterUi.WriteLine(Trim(Str(Ui(NE))))
FileWriterRHOi.WriteLine(Trim(Str(RHOi(NE))))
FileWriterGAMMAi.WriteLine(Trim(Str(GAMMAi(NE))))
FileWriterSi.WriteLine(Trim(Str(Si(NE))))
FileWriterdeltaQi.WriteLine(Trim(Str(deltaQi(NE))))
FileWriterdeltaNi.WriteLine(Trim(Str(deltaNi(NE))))
' Propellant variables
FileWriterBi.WriteLine(Trim(Str(Bi(NE))))
FileWriterRgraini.WriteLine(Trim(Str(Rgraini(NE))))
FileWriterRholesi.WriteLine(Trim(Str(Rholesi(NE))))
FileWriterPropVOLi.WriteLine(Trim(Str(PropVOLi(NE))))
```

```
        FileWriterdeltaMi.WriteLine(Trim(Str(deltaMi(NE))))
        FileWriterfi.WriteLine(Trim(Str(fi(NE))))
        ' Projectile variables
        FileWriterShell.WriteLine(Trim(Str(Time)) & ", " & Trim(Str(Xshell)) & ", " & _
            Trim(Str(Vshell)) & ", " & Trim(Str(ACCshell)) & ", " & Trim(Str(Pshell)))
    End Sub

End Module
```

Module HoleCalculations

```
Option Strict On
Option Explicit On

Public Module HoleCalculations

    ' List of subroutines:
    '   VolumeOfPropellantBurned_WithHoles()
    '   VolumeOfPropellantBurned_NoHoles()

    Public Sub VolumeOfPropellantBurned_WithHoles( _
        ByVal lRgrainStart As Double, _
        ByVal lRholesStart As Double, _
        ByVal lBurnRate As Double, _
        ByVal lTimeStep As Double, _
        ByRef lRgrainEnd As Double, _
        ByRef lRholesEnd As Double, _
        ByRef lVolumeEnd As Double)
        '
        ' ///////////////////////////////////////////////////////////////////////////
        ' // Phase 1 burning - Grain is intact
        ' ///////////////////////////////////////////////////////////////////////////
        Dim lBurnDepth As Double
        If (lRholesStart < RholesEndPhase1) Then
            ' Calculate the depth burned during this time step
            lBurnDepth = lBurnRate * lTimeStep
            ' Calculate new radii for the grain and the holes
            lRgrainEnd = lRgrainStart - lBurnDepth
            lRholesEnd = lRholesStart + lBurnDepth
            ' Calculate the new volume
            lVolumeEnd = Math.PI * _
                ((lRgrainEnd ^ 2) - (7 * (lRholesEnd ^ 2))) * LgrainEff
            Exit Sub
        End If
        '
        ' ///////////////////////////////////////////////////////////////////////////
        ' // Calculate new hole radii for Phase 2 and Phase 3 burning
        ' ///////////////////////////////////////////////////////////////////////////
        ' Calculate the depth burned during this time step
        lBurnDepth = lBurnRate * lTimeStep
        ' Calculate new radius for the holes
        lRholesEnd = lRholesStart + lBurnDepth
        '
        ' ///////////////////////////////////////////////////////////////////////////
        ' // Phase 2 burning - Grain has disintegrated; 12 "triangles" are burning
        ' ///////////////////////////////////////////////////////////////////////////
        Dim lFactor1, lFactor2, lFactor3, lFactor4, lFactor5 As Double
```

```
Dim lAlphaEnd, lLengthADEnd As Double
Dim lAreaStraightTriangle As Double
Dim lAreaCircularSegment As Double
Dim lAreaInnerTriangle As Double
If (lRholesEnd < RholesEndPhase2) Then
    ' Calculate angle alpha
    lFactor1 = Math.Sqrt((lRholesEnd ^ 2) - (RholesEndPhase1 ^ 2))
    lFactor2 = (Math.Sqrt(3) * RholesEndPhase1) - lFactor1
    lFactor3 = RholesEndPhase1 + (Math.Sqrt(3) * lFactor1)
    lAlphaEnd = _
        Math.Atan2(lFactor2, lFactor3) - Math.Atan2(lFactor1, RholesEndPhase1)
    ' Calculate the length of line segment AD
    lFactor4 = RholesEndPhase1 - (0.5 * lFactor3)
    lFactor5 = lFactor1 - (0.5 * lFactor2)
    lLengthADEnd = Math.Sqrt((lFactor4 ^ 2) + (lFactor5 ^ 2))
    ' Calculate the area of the straight-sided triangle
    lAreaStraightTriangle = Math.Sqrt(3) * (lLengthADEnd ^ 2) / 4
    ' Calculate the area of the circular segments
    lFactor2 = 0.5 * lAlphaEnd * (lRholesEnd ^ 2)
    lFactor3 = 0.5 * lLengthADEnd * lRholesEnd * Math.Cos(lAlphaEnd / 2)
    lAreaCircularSegment = lFactor2 - lFactor3
    ' Calculate the area of one inner "triangle"
    lAreaInnerTriangle = lAreaStraightTriangle - (3 * lAreaCircularSegment)
Else
    lAreaInnerTriangle = 0
End If
'
' ////////////////////////////////////////////////////////////////////////
' // Phase 3 burning - Only six outer "triangles" are burning
' ////////////////////////////////////////////////////////////////////////
Dim lBetaEnd, lThetaEnd As Double
Dim lLengthGIEnd, lLengthOZEnd, lLengthOEEnd, lLengthEZEnd As Double
Dim lAreaGreen, lAreaViolet As Double
Dim lAreaOuterTriangle As Double
If (lRholesEnd < RholesEndPhase3) Then
    ' Calculate angle beta
    lFactor1 = Math.PI / 3
    lFactor2 = Math.Sqrt( _
        ((3 * RholesEndPhase1) - lRholesEnd) * (lRholesEnd - RholesEndPhase1))
    lFactor3 = (5 * RholesEndPhase1) - (2 * lRholesEnd)
    lBetaEnd = lFactor1 - (2 * Math.Atan2(lFactor2, lFactor3))
    ' Calculate angle theta
    lFactor1 = _
        (Math.Sqrt(3) * RholesEndPhase1) + _
        Math.Sqrt((lRholesEnd ^ 2) - (RholesEndPhase1 ^ 2))
    lFactor2 = _
        -RholesEndPhase1 + _
        Math.Sqrt(3 * ((lRholesEnd ^ 2) - (RholesEndPhase1 ^ 2)))
    lFactor3 = Math.Sqrt( _
        ((3 * RholesEndPhase1) - lRholesEnd) * (lRholesEnd - RholesEndPhase1))
    lFactor4 = (3 * RholesEndPhase1) - (2 * lRholesEnd)
    lThetaEnd = Math.Atan2(lFactor1, lFactor2) - Math.Atan2(lFactor3, lFactor4)
    ' Calculate the length of line segment GI
    lLengthGIEnd = _
        2 * ((4 * RholesEndPhase1) - lRholesEnd) * Math.Sin(lBetaEnd / 2)
    ' Calculate the length of line segment OZ
    lLengthOZEnd = ((4 * RholesEndPhase1) - lRholesEnd) * Math.Cos(lBetaEnd / 2)
    ' Calculate the length of line segment OE
```

```
            lFactor1 = _
                (3 * RholesEndPhase1) + _
                Math.Sqrt(3 * ((lRholesEnd ^ 2) - (RholesEndPhase1 ^ 2)))
            lFactor2 = _
                (Math.Sqrt(3) * RholesEndPhase1) + _
                Math.Sqrt((lRholesEnd ^ 2) - (RholesEndPhase1 ^ 2))
            lLengthOEEnd = 0.5 * Math.Sqrt((lFactor1 ^ 2) + (lFactor2 ^ 2))
            ' Calculate the length of line segment EZ
            lLengthEZEnd = lLengthOZEnd - lLengthOEEnd
            ' Calculate the area of the straight-sided triangle
            lAreaStraightTriangle = 0.5 * lLengthGIEnd * lLengthEZEnd
            ' Calculate the area of the green circular segments
            lFactor1 = ((4 * RholesEndPhase1) - lRholesEnd) ^ 2
            lFactor2 = lBetaEnd - Math.Sin(lBetaEnd)
            lAreaGreen = 0.5 * lFactor1 * lFactor2
            ' Calculate the area of the violet circular segment
            lFactor1 = lRholesEnd ^ 2
            lFactor2 = lThetaEnd - Math.Sin(lThetaEnd)
            lAreaViolet = 0.5 * lFactor1 * lFactor2
            ' Calculate the area of one outer "triangle"
            lAreaOuterTriangle = lAreaStraightTriangle + lAreaGreen - (2 * lAreaViolet)
        Else
            lAreaOuterTriangle = 0
        End If
        '
        ' ///////////////////////////////////////////////////////////////////////////////
        ' Calculate total volume of all "triangles" in Phases 2 and 3
        ' ///////////////////////////////////////////////////////////////////////////////
        ' Calculate the new volume
        lVolumeEnd = 6 * (lAreaInnerTriangle + lAreaOuterTriangle) * LgrainEff
    End Sub

    Public Sub VolumeOfPropellantBurned_NoHoles( _
        ByVal lRgrainStart As Double, _
        ByVal lBurnRate As Double, _
        ByVal lTimeStep As Double, _
        ByRef lRgrainEnd As Double, _
        ByRef lVolumeEnd As Double)
        Dim lBurnDepth As Double
        ' Calculate the depth burned during this time step
        lBurnDepth = lBurnRate * lTimeStep
        ' Calculate new radius for the grain
        lRgrainEnd = lRgrainStart - lBurnDepth
        ' Calculate the new volume
        lVolumeEnd = Math.PI * (lRgrainEnd ^ 2) * LgrainEff
    End Sub

End Module
```