A One-Dimensional, Noniterative Trajectory Model
(With a C++ Implementation)

by Robert J. Yager and Benjamin J. Flanders

ARL-TN-598
March 2014

 Approved for public release; distribution is unlimited.
NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer’s or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.
A One-Dimensional, Noniterative Trajectory Model
(With a C++ Implementation)

Robert J. Yager and Benjamin J. Flanders
Weapons and Materials Research Directorate, ARL
This report defines algorithms that can be used to calculate speed, distance, and time variables as functions of initial and/or final projectile states. Examples include speed as a function of initial speed and distance traveled, distance traveled as a function of initial and final speeds, and time of flight as a function of initial and final speeds. Derivations for the algorithms are based exclusively on force due to air resistance and allow for drag coefficients that have a functional relationship to Mach number.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>v</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>vi</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Scaled Variables</td>
<td>1</td>
</tr>
<tr>
<td>3. Derivation of Equations for Linear Drag Coefficients</td>
<td>2</td>
</tr>
<tr>
<td>3.1 Drag Coefficient as a Function of Mach Number</td>
<td>2</td>
</tr>
<tr>
<td>3.2 Scaled Distance as a Function of Mach Number</td>
<td>3</td>
</tr>
<tr>
<td>3.3 Mach Number as a Function of Scaled Distance</td>
<td>4</td>
</tr>
<tr>
<td>3.4 Scaled Time as a Function of Mach Number</td>
<td>5</td>
</tr>
<tr>
<td>4. Generalization to an Arbitrary Drag-Mach Relationship</td>
<td>6</td>
</tr>
<tr>
<td>4.1 Drag Coefficient as a Piecewise, Linear Function of Mach Number</td>
<td>6</td>
</tr>
<tr>
<td>4.2 Scaled Distance as a Function of Mach Number</td>
<td>7</td>
</tr>
<tr>
<td>4.3 Mach Number as a Function of Scaled Distance</td>
<td>8</td>
</tr>
<tr>
<td>4.4 Scaled Time as a Function of Mach Number</td>
<td>9</td>
</tr>
<tr>
<td>5. State Variables and Functional Notation</td>
<td>10</td>
</tr>
<tr>
<td>5.1 State Variables</td>
<td>10</td>
</tr>
<tr>
<td>5.2 Functional Notation</td>
<td>10</td>
</tr>
<tr>
<td>6. Solving for Unknowns</td>
<td>11</td>
</tr>
<tr>
<td>7. Example: The Lapua Scenar GB528 Rifle Bullet</td>
<td>12</td>
</tr>
<tr>
<td>7.1 Creating a Table of Unitless Constants</td>
<td>12</td>
</tr>
<tr>
<td>7.2 Calculating Speed as a Function of Initial Speed and Distance Traveled</td>
<td>13</td>
</tr>
<tr>
<td>7.3 Calculating Time of Flight as a Function of Initial Speed and Distance Traveled</td>
<td>14</td>
</tr>
</tbody>
</table>
8. **C++ Implementation**

8.1 Storing Drag-Table Coefficients: The DRAG Struct ....................................................14

8.2 Creating a Pointer to an Array of DRAG Structs: The Drags() Function .................15

8.3 Calculating Scaled Distance as a Function of Mach Number: The SofM() Function...16

8.4 Calculating Mach Number as a Function of Scaled Distance: The MofS() Function...16

8.5 Calculating Scaled Time: The TofM() Function ...........................................................17

8.6 Example: Calculating Speed and Time of Flight for the Lapua Scenar GB528 Rifle
   Bullet ......................................................................................................................................18

8.7 Example: Calculating Scaled Time of Impact for Fragment Trajectories..................19

9. **Code Summary**

   Distribution List 22
List of Figures

Figure 1. Drag coefficient, $C_D$, as a function of Mach number, $M$. ................................................2
Figure 2. Scaled distance, $s$, as a function of Mach number, $M$. .....................................................4
Figure 3. Mach number, $M$, as a function of scaled distance, $s$. .........................................................4
Figure 4. Scaled time, $\tau$, as a function of Mach number, $M$. .............................................................6
Figure 5. Drag coefficient, $C_D$, as a piecewise function of Mach number, $M$. ........................................7
Figure 6. Scaled distance, $s$, as a piecewise function of Mach number, $M$. .............................................8
Figure 7. Mach number, $M$, as a piecewise function of scaled distance, $s$. .............................................9
Figure 8. Scaled time, $\tau$, as a piecewise function of Mach number, $M$. ................................................10
Figure 9. Scaled time of impact, $\tau$, for fragments striking a plate. .......................................................20

List of Tables

Table 1. Unitless parameters for a Lapua Scenar GB528 rifle bullet. ......................................................12
Acknowledgments

The authors would like to thank Dr. Paul Weinacht of the U.S. Army Research Laboratory’s Weapons and Materials Research Directorate. Dr. Weinacht provided technical and editorial recommendations that improved the quality of this report.
1. Introduction

This report defines algorithms that can be used to calculate speed, distance, and time variables as functions of initial and/or final projectile states. Examples include speed as a function of initial speed and distance traveled, distance traveled as a function of initial and final speeds, and time of flight as a function of initial and final speeds. Derivations for the algorithms are based exclusively on force due to air resistance and allow for drag coefficients that have a functional relationship to Mach number.

All algorithms are written in terms of scaled variables (speed, distance, and time) that are independent of units. This approach allows for the precalculation of a set of parameters that depend only on the characteristics of a particular drag versus Mach curve (i.e., the parameters are independent of projectile mass, projectile cross-sectional area, and air density). A practical benefit of this approach is a reduction in computation time.

The algorithms presented in this report are particularly useful for situations where total distance traveled is approximately equal to the distance between starting and ending locations. Examples include flat-fire trajectories and fragment trajectories.

JTCG/ME-79-1-2\(^1\) presents a method for calculating speed as a function of initial speed and total distance traveled. It relies upon an analytic solution that is similar to what is presented in sections 3.1 and 3.2 of this report. The key difference is that the method presented in this report provides a solution to the equations of motion in terms of variables that are independent of units.

2. Scaled Variables

Recall that Mach number, \(M\), is defined to be the ratio of the speed, \(v\), of an object moving through a fluid to the local speed of sound, \(v_s\). That is,

\[
M \equiv \frac{v}{v_s}
\]  

(1)

Scaled distance, \(s\), is defined as a function of measured distance, \(x\):

---

\[ s = \frac{\rho A}{2m} x \]  

(2)

where \( \rho \) is the density of the air through which a projectile travels, \( A \) is the projectile’s cross-sectional area, and \( m \) is its mass.

Scaled time, \( \tau \), is defined as a function of measured time, \( t \):

\[ \tau = \frac{\rho A v}{2m} t \]  

(3)

### 3. Derivation of Equations for Linear Drag Coefficients

Begin with the equation for the force acting on a projectile that is due to air resistance:

\[ F = -\frac{1}{2} C_D \rho A v^2 \]  

(4)

where \( C_D \) is a projectile’s drag coefficient, and \( \rho \), \( A \), and \( v \) are as defined in section 2.

#### 3.1 Drag Coefficient as a Function of Mach Number

Assume that \( C_D \) is linearly dependent on Mach number (figure 1). Then

\[ C_D = \alpha M + \beta \]  

(5)

where \( \alpha \) and \( \beta \) are constants representing slope and intercept, respectively.

![Figure 1. Drag coefficient, \( C_D \), as a function of Mach number, \( M \).](image)
3.2 Scaled Distance as a Function of Mach Number

Using Newton’s second law, equation 4 can be used to solve for acceleration:

\[ F = ma \]  \hspace{1cm} (6)

\[ \Rightarrow a = -\frac{1}{2m} C_D \rho A v^2 \]  \hspace{1cm} (7)

Next, the definitions of speed and acceleration can be used to find acceleration as a function of speed and position:

\[ v = \frac{dx}{dt} \]  \hspace{1cm} (8)

and \[ a = \frac{dv}{dt} = \frac{dv}{dx} \frac{dx}{dt} \]  \hspace{1cm} (9)

\[ \Rightarrow a = v \frac{dv}{dx} \]  \hspace{1cm} (10)

Combining equations 7 and 10,

\[ \frac{dv}{dx} = -\frac{1}{2m} C_D \rho A v \]  \hspace{1cm} (11)

Next, equations 1 and 2 can be used to rewrite equation 11 in terms of scaled variables:

\[ \frac{dM}{ds} = -C_D M \]  \hspace{1cm} (12)

Substituting equation 5 into equation 12,

\[ \frac{dM}{ds} = -(\alpha M + \beta) M \]  \hspace{1cm} (13)

Equation 13 can be solved for \( s \) by using the method of separation of variables:

\[ s = -\int \frac{1}{(\alpha M + \beta) M} dM \]  \hspace{1cm} (14)

The solution to the integral in equation 14 can be found in integral tables:\(^2\)

\[ s = \frac{1}{\beta} \ln \left( \frac{\alpha + \frac{\beta}{M}}{M} \right) + c \]  \hspace{1cm} (15)

where \( c \) is a constant of the integration and is determined by initial values for \( s \) and \( M \).

\[ c = s_0 - \frac{1}{\beta} \ln \left( \alpha + \frac{\beta}{M_0} \right) \]  

(16)

Note that equations 15 and 16 are only valid for \( \alpha + \beta / M > 0 \), \( M \neq 0 \), and \( \beta \neq 0 \). Since, by definition, Mach numbers are non-negative, the conditions \( \alpha + \beta / M > 0 \) and \( M \neq 0 \) are assured if we assume that \( C_D > 0 \) for all values of \( M \) and \( M_0 > 0 \). The case where \( \beta = 0 \) will occur occasionally. To be exact, equation 14 should be solved for the special case where \( \beta = 0 \). In practice, however, when encountering a situation where \( \beta = 0 \), it is sufficient to change \( \beta \) to some very small value.

Figure 2 presents scaled distance, \( s \), as a function of Mach number, \( M \).

![Figure 2](image)

**Figure 2.** Scaled distance, \( s \), as a function of Mach number, \( M \).

### 3.3 Mach Number as a Function of Scaled Distance

Solving equation 15 for \( M \) yields Mach number as a function of scaled distance (figure 3):

\[ M = \frac{\beta}{e^{\beta (s-c)} - \alpha} \]  

(17)

![Figure 3](image)

**Figure 3.** Mach number, \( M \), as a function of scaled distance, \( s \).
3.4 Scaled Time as a Function of Mach Number

Begin by using equations 1 and 7 to find acceleration as a function of Mach number:

\[ a = -\frac{1}{2m} C_D \rho A v_s^2 M^2 \]  \hspace{1cm} (18)

Next, use equations 1 and 9 to relate acceleration to the time derivative of Mach number:

\[ a = v_s \frac{dM}{dt} \]  \hspace{1cm} (19)

Combining equations 18 and 19,

\[ \frac{dM}{dt} = -\frac{1}{2m} C_D \rho A v_s v M^2 \]  \hspace{1cm} (20)

Equation 3 can be used to rewrite equation 20 in terms of scaled time:

\[ \frac{dM}{d\tau} = -C_D M^2 \]  \hspace{1cm} (21)

Substituting equation 5 into equation 21,

\[ \frac{dM}{d\tau} = -(\alpha M + \beta)M^2 \]  \hspace{1cm} (22)

Equation 22 can be solved for \( \tau \) by using the method of separation of variables:

\[ \tau = -\int \frac{1}{(\alpha M + \beta)M^2} dM \]  \hspace{1cm} (23)

The solution to the integral in equation 23 can be found in integral tables:

\[ \tau = \frac{1}{\beta M} - \frac{\alpha}{\beta^2} \ln \left( \frac{\alpha + \beta}{M} \right) + d \]  \hspace{1cm} (24)

where \( d \) is a constant of the integration and is determined by initial values for \( \tau \) and \( M \).

\[ d = \tau_0 - \frac{1}{\beta M_0} + \frac{\alpha}{\beta^2} \ln \left( \frac{\alpha + \beta}{M_0} \right) \]  \hspace{1cm} (25)

As was the case with equations 15 and 16, equations 24 and 25 are only valid for \( \alpha + \beta / M > 0 \), \( M \neq 0 \), and \( \beta \neq 0 \). Since, by definition, Mach numbers are non-negative, the conditions \( \alpha + \beta / M > 0 \) and \( M \neq 0 \) are assured if we assume that \( C_D > 0 \) for all values of \( M \) and \( M_0 > 0 \).

---

The case where $\beta = 0$ will occur occasionally. To be exact, equation 23 should be solved for the special case where $\beta = 0$. In practice, however, when encountering a situation where $\beta = 0$, it is sufficient to change $\beta$ to some very small value.

Figure 4 presents scaled time, $\tau$, as a function of Mach number, $M$.

$$\tau = \frac{1}{\beta M} - \frac{\alpha}{\beta^2} \ln\left(\frac{\alpha + \beta}{M}\right) + d,$$

where $d = \tau_0 - \frac{1}{\beta M_0} - \frac{\alpha}{\beta^2} \ln\left(\frac{\alpha + \beta}{M_0}\right)$.

and $\alpha$, $\beta$, $\tau_0$, and $M_0$ are free parameters ($M_0 > 0$ and $\beta \neq 0$).

Figure 4. Scaled time, $\tau$, as a function of Mach number, $M$.

4. Generalization to an Arbitrary Drag-Mach Relationship

The equations in section 3 can be generalized for a piecewise, linear function of drag given Mach. This technique allows for any functional relationship between drag coefficient and Mach number to be modeled.

4.1 Drag Coefficient as a Piecewise, Linear Function of Mach Number

Suppose that drag coefficient as a function of Mach number is defined by $n$ matched pairs of values for Mach number and drag coefficient, $(M_k, C_{D,k})$, such that

$$M_{k+1} < M_k \text{ for } 0 \leq k < n - 1$$  \hspace{1cm} (26)

Using the matched pairs, $n - 1$ equations that have the form of equation 5 can be generated:

$$C_D = \alpha_k M + \beta_k$$  \hspace{1cm} (27)

for $k$ such that $M_{k+1} \leq M \leq M_k$ and $0 \leq k < n - 1$.

Figure 5 presents drag coefficient, $C_D$, as a piecewise function of Mach number, $M$. 
\[ C_D = \alpha_k M + \beta_k \quad \text{for } k \text{ such that } M_{k+1} \leq M \leq M_k \]

where
\[
\begin{align*}
\alpha_k &= \frac{C_{D,k} - C_{D,k+1}}{M_k - M_{k+1}} \\
\beta_k &= C_{D,k} - \alpha_k M_k
\end{align*}
\]

\[ C_D = \frac{2F_{drag}}{\rho A v^2} \quad \text{& } M = \frac{v}{v_i} \]

Figure 5. Drag coefficient, \( C_D \), as a piecewise function of Mach number, \( M \).

Equation 27 can be used to determine the values of \( \alpha_k \) and \( \beta_k \):
\[
\alpha_k = \frac{C_{D,k} - C_{D,k+1}}{M_k - M_{k+1}} \quad (28)
\]

and
\[
\beta_k = C_{D,k} - \alpha_k M_k \quad (29)
\]

4.2 Scaled Distance as a Function of Mach Number

Equation 15 can be generalized to a piecewise function of Mach number:
\[
s = \frac{1}{\beta_k} \ln \left( \alpha_k + \frac{\beta_k}{M} \right) + c_k \quad (30)
\]

for \( k \) such that \( M_{k+1} \leq M \leq M_k \).

Values for \( c_k \) can be found by generalizing equation 16:
\[
c_k = s_{0,k} - \frac{1}{\beta_k} \ln \left( \alpha_k + \frac{\beta_k}{M_{0,k}} \right) \quad (31)
\]

Values for initial conditions for each segment of the function are given by equations 32 and 33:
\[
M_{0,k} = M_k \quad (32)
\]

and
\[
s_{0,k} = s_k \quad (33)
\]

Values for \( s_k \) can be obtained from equation 30:
\[ s_k = \frac{1}{\beta_{k-1}} \ln \left( \alpha_{k-1} + \frac{\beta_{k-1}}{M_k} \right) + c_{k-1} \]  

(34)

Since the value of \( s_{k=0} \) is arbitrary, zero is used as a convenient choice:

\[ s_{k=0} = 0 \]  

(35)

Figure 6 presents an overview of the functions necessary to calculate scaled distance as a function of Mach number. Note that for a particular drag coefficient versus Mach table, \( \alpha_k \), \( \beta_k \), \( c_k \), and \( s_k \) are unique (independent of particle mass, air density, etc.). Thus, they can be precalculated and stored in tabulated form.

\[ s = \frac{1}{\beta_k} \ln \left( \alpha_k + \frac{\beta_k}{M} \right) + c_k \]  

for \( M_{k-1} \leq M \leq M_k \)

\[ \alpha_k = \frac{C_{D,k} - C_{D,k+1}}{M_k - M_{k-1}}, \quad \beta_k = C_{D,k} - \alpha_k M_k, \]

where

\[ c_k = s_k - \frac{1}{\beta_k} \ln \left( \alpha_k + \frac{\beta_k}{M_k} \right), \]

\[ s_k = \frac{1}{\beta_{k-1}} \ln \left( \alpha_{k-1} + \frac{\beta_{k-1}}{M_k} \right) + c_{k-1}, \]  

& \( s_{k=0} = 0 \)

Figure 6. Scaled distance, \( s \), as a piecewise function of Mach number, \( M \).

### 4.3 Mach Number as a Function of Scaled Distance

Equation 17 can be generalized to a piecewise function of scaled distance:

\[ M = \frac{\beta_k}{e^{\beta_k (s - c_k)} - \alpha_k} \]  

(36)

for \( k \) such that \( s_k \leq s \leq s_{k+1} \).

Figure 7 presents an overview of the functions necessary to calculate Mach as a function of scaled distance. Note that for a particular drag coefficient versus Mach table, \( \alpha_k \), \( \beta_k \), \( c_k \), and \( s_k \) are unique (independent of particle mass, air density, etc.). Thus, they can be precalculated and stored in tabulated form.
4.4 Scaled Time as a Function of Mach Number

Equation 24 can be generalized to a piecewise function of Mach number:

\[ \tau = \frac{1}{\beta_k M} - \frac{\alpha_k}{\beta^2_k} \ln \left( \frac{\alpha_k + \beta_k}{M} \right) + d_k \]  

(37)

for \( k \) such that \( M_{k+1} \leq M \leq M_k \).

Values for \( d_k \) can be found by generalizing equation 25:

\[ d_k = \tau_{0,k} - \frac{1}{\beta_k M_{0,k}} + \frac{\alpha_k}{\beta^2_k} \ln \left( \frac{\alpha_k + \beta_k}{M_{0,k}} \right) \]  

(38)

Values for initial conditions for each segment of the function are given by equations 32 and 39:

\[ \tau_{0,k} = \tau_k \]  

(39)

Values for \( \tau_k \) can be obtained from equation 37:

\[ \tau_k = \frac{1}{\beta_{k-1} M_k} - \frac{\alpha_{k-1}}{\beta^2_{k-1}} \ln \left( \frac{\alpha_{k-1} + \beta_{k-1}}{M_k} \right) + d_{k-1} \]  

(40)

Since the value of \( \tau_{k=0} \) is arbitrary, zero is used as a convenient choice:

\[ \tau_{k=0} = 0 \]  

(41)

Figure 8 presents an overview of the functions necessary to calculate scaled time as a function of Mach number. Note that given the choice of \( \tau_{k=0} = 0 \), for a particular drag versus Mach table, \( \alpha_k \), \( \beta_k \), \( d_k \), and \( \tau_k \) are unique. Thus, they can be precalculated and stored in tabulated form.
5. State Variables and Functional Notation

For a given set of atmospheric and projectile characteristics, knowledge of any one of the three scaled variables defined in section 2 (i.e., $M$, $s$, or $\tau$) is sufficient to uniquely define the state of a projectile. Applications of the equations presented in section 4 typically involve solving for some unknown state variable given one or two known state variables. The following six equations can make solving for unknowns easier.

5.1 State Variables

Since $M$, $s$, and $\tau$ can each be used to characterize the state of a projectile, the following three definitions can be used to quantify a change in a state variable.

$$\Delta M \equiv M_{\text{final}} - M_{\text{initial}}$$  \hspace{1cm} (42)

$$\Delta s \equiv s_{\text{final}} - s_{\text{initial}}$$  \hspace{1cm} (43)

$$\Delta \tau \equiv \tau_{\text{final}} - \tau_{\text{initial}}$$  \hspace{1cm} (44)

5.2 Functional Notation

Using equation 30, define $s(M)$ to be scaled distance as a function of Mach number:

$$s(M) = \frac{1}{\beta_k} \ln \left( \alpha_k + \frac{\beta_k}{M} \right) + c_k$$  \hspace{1cm} (45)

Using equation 36, define $M(s)$ to be Mach number as a function of scaled distance:
\[ M(s) = \frac{\beta_k}{e^{\beta_k(s-e_k)} - \alpha_k} \]  

(46)

Using equation 37, define \( \tau(M) \) to be scaled time as a function of Mach number:

\[ \tau(M) = \frac{1}{\beta_k M} - \frac{\alpha_k}{\beta_k^2} \ln \left( \frac{\alpha_k + \beta_k}{M} \right) + d_k \]  

(47)

6. Solving for Unknowns

When attempting to solve for some unknown state variable, it helps to break the problem into four sections:

- Identify known and unknown values for unscaled state variables.
- Convert known values for state variables to scaled state variables using the equations presented in section 2.
- Use the equations presented in section 5 to develop an equation that is solved for the unknown scaled state variable in terms of known scaled state variables.
- Use the equations presented in section 2 to convert the unknown scaled state variable to an unscaled value.

For example, suppose that initial speed, \( v_{\text{initial}} \), and distance traveled, \( \Delta x \), are known, and it is wished to determine final speed, \( v_{\text{final}} \). The first step is to use equations 1 and 2 to convert \( v_{\text{initial}} \) and \( \Delta x \) to the scaled variables \( M_{\text{initial}} \) and \( \Delta s \). Next, represent the problem in functional notation:

\[ M_{\text{final}} = M(s_{\text{final}}) \]  

(48)

where \( M(s_{\text{final}}) \) is given by equation 46.

Solve equation 42 for the final scaled speed and substitute into equation 46:

\[ M_{\text{final}} = M(\Delta s + s_{\text{initial}}) \]  

(49)

Next, note that \( s_{\text{initial}} \) can be calculated using equation 45. Thus,

\[ M_{\text{final}} = M(\Delta s + s(M_{\text{initial}})) \]  

(50)

Finally, use equation 1 to find \( v_{\text{final}} \) given \( M_{\text{final}} \). A wide variety of problems can be solved in this manner.
7. Example: The Lapua Scenar GB528 Rifle Bullet

7.1 Creating a Table of Unitless Constants

Wikipedia’s External Ballistics page\(^4\) lists drag coefficient versus Mach number values for the Lapua Scenar GB528 rifle bullet. Using those values, equations from section 4 can be used to calculate the parameters listed in table 1. Specifically, given \(M_k\) and \(C_{D,k}\) for \(0 \leq k < 28\), the equations summarized in figures 7 and 8 have been used to calculate \(\alpha_k\), \(\beta_k\), \(s_k\), \(c_k\), \(\tau_k\), and \(d_k\).

<table>
<thead>
<tr>
<th>(k)</th>
<th>(M)</th>
<th>(C_{D})</th>
<th>(\alpha)</th>
<th>(\beta)</th>
<th>(s)</th>
<th>(c)</th>
<th>(\tau)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.4</td>
<td>0.27</td>
<td>-0.06</td>
<td>0.414</td>
<td>0</td>
<td>5.273</td>
<td>0</td>
<td>-0.241615</td>
</tr>
<tr>
<td>1</td>
<td>2.2</td>
<td>0.282</td>
<td>-0.05</td>
<td>0.392</td>
<td>0.315209</td>
<td>5.555784</td>
<td>0.137177</td>
<td>-0.353937</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.292</td>
<td>-0.06</td>
<td>0.412</td>
<td>0.647242</td>
<td>5.317506</td>
<td>0.295484</td>
<td>-0.237973</td>
</tr>
<tr>
<td>3</td>
<td>1.8</td>
<td>0.304</td>
<td>-0.085</td>
<td>0.457</td>
<td>1.000723</td>
<td>4.89244</td>
<td>0.481805</td>
<td>-0.01001</td>
</tr>
<tr>
<td>4</td>
<td>1.6</td>
<td>0.321</td>
<td>-0.07</td>
<td>0.433</td>
<td>1.377521</td>
<td>5.087262</td>
<td>0.703845</td>
<td>-0.139846</td>
</tr>
<tr>
<td>5</td>
<td>1.5</td>
<td>0.328</td>
<td>-0.08</td>
<td>0.448</td>
<td>1.576392</td>
<td>4.96971</td>
<td>0.832223</td>
<td>-0.049923</td>
</tr>
<tr>
<td>6</td>
<td>1.4</td>
<td>0.336</td>
<td>-0.07</td>
<td>0.434</td>
<td>1.784183</td>
<td>5.072469</td>
<td>0.975621</td>
<td>-0.13983</td>
</tr>
<tr>
<td>7</td>
<td>1.3</td>
<td>0.343</td>
<td>-0.05</td>
<td>0.408</td>
<td>2.002448</td>
<td>5.268108</td>
<td>1.137427</td>
<td>-0.347739</td>
</tr>
<tr>
<td>8</td>
<td>1.2</td>
<td>0.348</td>
<td>0</td>
<td>0.348</td>
<td>2.234102</td>
<td>5.791212</td>
<td>1.32293</td>
<td>-0.017106</td>
</tr>
<tr>
<td>9</td>
<td>1.15</td>
<td>0.348</td>
<td>0.02</td>
<td>0.325</td>
<td>2.3564</td>
<td>6.034291</td>
<td>1.427044</td>
<td>-1.474873</td>
</tr>
<tr>
<td>10</td>
<td>1.1</td>
<td>0.347</td>
<td>0.08</td>
<td>0.259</td>
<td>2.48432</td>
<td>6.938917</td>
<td>1.54079</td>
<td>-3.345151</td>
</tr>
<tr>
<td>11</td>
<td>1.075</td>
<td>0.345</td>
<td>0.16</td>
<td>0.173</td>
<td>2.550765</td>
<td>9.120311</td>
<td>1.601894</td>
<td>-9.851053</td>
</tr>
<tr>
<td>12</td>
<td>1.05</td>
<td>0.341</td>
<td>0.28</td>
<td>0.047</td>
<td>2.619369</td>
<td>26.548368</td>
<td>1.666471</td>
<td>-161.15269</td>
</tr>
<tr>
<td>13</td>
<td>1.025</td>
<td>0.334</td>
<td>1.12</td>
<td>-0.814</td>
<td>2.690775</td>
<td>1.313248</td>
<td>1.735305</td>
<td>-1.038474</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0.306</td>
<td>2.8</td>
<td>-2.494</td>
<td>2.768002</td>
<td>2.293195</td>
<td>1.811601</td>
<td>1.679499</td>
</tr>
<tr>
<td>15</td>
<td>0.975</td>
<td>0.236</td>
<td>2.36</td>
<td>-2.065</td>
<td>2.862002</td>
<td>2.175026</td>
<td>1.906853</td>
<td>1.618416</td>
</tr>
<tr>
<td>16</td>
<td>0.95</td>
<td>0.177</td>
<td>0.92</td>
<td>-0.697</td>
<td>2.988737</td>
<td>0.577959</td>
<td>2.038621</td>
<td>0.366765</td>
</tr>
<tr>
<td>17</td>
<td>0.925</td>
<td>0.154</td>
<td>0.48</td>
<td>-0.29</td>
<td>3.150184</td>
<td>-3.032027</td>
<td>2.210906</td>
<td>-2.938533</td>
</tr>
<tr>
<td>18</td>
<td>0.9</td>
<td>0.142</td>
<td>0.2</td>
<td>-0.038</td>
<td>3.353448</td>
<td>-45.258439</td>
<td>2.413998</td>
<td>-224.10354</td>
</tr>
<tr>
<td>19</td>
<td>0.875</td>
<td>0.137</td>
<td>0</td>
<td>0.137</td>
<td>3.537428</td>
<td>17.072049</td>
<td>2.641631</td>
<td>-5.700392</td>
</tr>
<tr>
<td>20</td>
<td>0.85</td>
<td>0.137</td>
<td>-0.16</td>
<td>0.273</td>
<td>3.749016</td>
<td>10.434934</td>
<td>2.886984</td>
<td>2.496055</td>
</tr>
<tr>
<td>21</td>
<td>0.825</td>
<td>0.141</td>
<td>-0.12</td>
<td>0.24</td>
<td>3.963785</td>
<td>11.324716</td>
<td>3.143445</td>
<td>1.773405</td>
</tr>
<tr>
<td>22</td>
<td>0.8</td>
<td>0.144</td>
<td>-0.2</td>
<td>0.304</td>
<td>4.179723</td>
<td>9.820507</td>
<td>3.409242</td>
<td>3.008442</td>
</tr>
<tr>
<td>23</td>
<td>0.7</td>
<td>0.164</td>
<td>-0.07</td>
<td>0.213</td>
<td>5.046777</td>
<td>11.859988</td>
<td>4.567078</td>
<td>0.099253</td>
</tr>
<tr>
<td>24</td>
<td>0.6</td>
<td>0.171</td>
<td>-0.29</td>
<td>0.345</td>
<td>5.96672</td>
<td>9.605173</td>
<td>5.987225</td>
<td>4.214716</td>
</tr>
<tr>
<td>25</td>
<td>0.5</td>
<td>0.2</td>
<td>-0.29</td>
<td>0.345</td>
<td>6.949257</td>
<td>9.605173</td>
<td>7.79309</td>
<td>4.214716</td>
</tr>
<tr>
<td>26</td>
<td>0.4</td>
<td>0.229</td>
<td>-0.0025</td>
<td>0.23</td>
<td>7.988528</td>
<td>10.413495</td>
<td>10.102174</td>
<td>-0.741033</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>0.23</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

All of the parameters listed in table 1, though specific to the Lapua Scenar GB528 rifle bullet, are independent of air density and initial speed.

7.2 Calculating Speed as a Function of Initial Speed and Distance Traveled

For the Lapua Scenar GB528 rifle bullet, Wikipedia’s External Ballistics page lists mass ($m$) as 19.44 g, diameter as 8.59 mm, and initial speed ($v_{initial}$) as 830 m/s.

Assuming standard sea-level atmospheric conditions (air density ($\rho$) equals 1.225 kg/m$^3$ and speed of sound ($v_{s}$) equals 340.3 m/s)$^5$ the speed of a bullet that has traveled 300 m ($\Delta x$) can be calculated using the equations presented in sections 1 and 5.

First use equation 1 to find initial Mach number ($M_{initial}$):

$$M_{initial} = \frac{v_{initial}}{v_{sound}} = \frac{830 \text{ m/s}}{340.3 \text{ m/s}} = 2.43902$$

(51)

Next, use the equation for the area of a circle to find the cross-sectional area ($A$) of the projectile:

$$A = \pi r^2 = 3.14159 \left( \frac{.00859 \text{ m}}{2} \right)^2 = 5.7953 \times 10^{-5} \text{ m}^2$$

(52)

Use equation 2 to find the scaled distance traveled ($\Delta s$):

$$\Delta s = \frac{\rho A}{2m} \Delta x = \frac{(1.225 \text{ kg/m}^3)(5.7953 \times 10^{-5} \text{ m}^2)}{2(0.1944 \text{ kg})}(300 \text{ m}) = .54778$$

(53)

Use equation 45 to find the initial scaled distance ($s_{initial}$):

$$s_{initial} = \frac{1}{\beta k=0} \left( \alpha k=0 + \frac{\beta k=0}{M_{initial}} \right) + c_{k=0} = \frac{1}{.414} \ln \left( - .06 + \frac{.414}{2.43902} \right) + 5.2773 = -.059991$$

(54)

Use equation 43 to find the final scaled distance ($s_{final}$):

$$s_{final} = s_{initial} + \Delta s = -.059991 + .54778 = .487789$$

(55)

Use equation 46 to find the final Mach number ($M_{final}$):

$$M_{final} = \frac{\beta k=1}{e^{\beta k=1}(v_{final}-v_{s}) - \alpha k=1} = \frac{.392}{e^{.392(487789-5.5557845) + .05}} = 2.09454$$

(56)

Finally, use equation 1 to find the final speed ($v_{final}$):

$$v_{final} = M_{final}v_{sound} = 2.09454(340.3 \text{ m/s}) = 713 \text{ m/s}$$

(57)

7.3 Calculating Time of Flight as a Function of Initial Speed and Distance Traveled

Use equation 47 to find $\tau_{\text{initial}}$ and $\tau_{\text{final}}$:

$$
\tau_{\text{initial}} = \frac{1}{\beta_{k=0} M_{\text{initial}}} - \frac{\alpha_{k=0}}{\beta_{k=0}^2} \ln \left( \frac{\alpha_{k=0}}{\beta_{k=0}} + \frac{\beta_{k=0}}{M_{\text{initial}}} \right) + d_{k=0}
$$

$$
= \frac{1}{.414(2.439)} - \frac{(-.06)}{.414^2} \ln \left( -.06 + \frac{.414}{2.439} \right) + (.2416)
$$

$$
= -.024768,
$$

$$
\tau_{\text{final}} = \frac{1}{\beta_{k=1} M_{\text{final}}} - \frac{\alpha_{k=1}}{\beta_{k=1}^2} \ln \left( \frac{\alpha_{k=1}}{\beta_{k=1}} + \frac{\beta_{k=1}}{M_{\text{initial}}} \right) + d_{k=1}
$$

$$
= \frac{1}{(.392)(2.095)} - \frac{(-.05)}{.392^2} \ln \left( -.05 + \frac{.392}{2.095} \right) + (-.3539)
$$

$$
= .217245.
$$

Next, calculate the change in scaled time ($\Delta \tau$):

$$
\Delta \tau = \tau_{\text{final}} - \tau_{\text{initial}} = .217245 - (-.024768) = .242013
$$

Finally, use equation 3 to find time of flight ($\Delta t$)

$$
\Delta t = \frac{2m\Delta \tau}{\rho A v_s} = \frac{2(.01944 \text{ kg})(.242013)}{(1.225 \text{ kg/m}^3)(5.795 \times 10^{-5} \text{ m}^2)(340.3 \text{ m/s})} = .390 \text{ s}
$$

8. C++ Implementation

8.1 Storing Drag-Table Coefficients: The DRAG Struct

DRAG structs are used to store the parameters associated with a single linear drag profile. Multiple DRAG structs can be used to describe a nonlinear drag profile.

**DRAG Code**

```cpp
struct DRAG{//<=============A LINEAR DRAG PROFILE (CREATE A SET USING Drags())
    double M;//<-------------------THE MAXIMUM MACH VALUE FOR THE DRAG PROFILE
    double a;//<---------------------------------THE SLOPE OF THE DRAG PROFILE
    double b;//<---------------------------THE Y-INTERCEPT OF THE DRAG PROFILE
    double s;//<--------------THE MINIMUM SCALED DISTANCE FOR THE DRAG PROFILE
    double c;//<-------------------THE SCALED-DISTANCE CONSTANT OF INTEGRATION
    double T;//<------------------THE MINIMUM SCALED TIME FOR THE DRAG PROFILE
    double d;//<-----------------------THE SCALED-TIME Constant OF INTEGRATION
};~~~YAGENAUT@GMAIL.COM~~~~~~~~~LAST-UPDATED-09DEC2013~~~~~~
```
DRAG Parameters

M  M specifies the maximum Mach value for the drag profile.
a  a specifies the slope, $\alpha$, of the drag profile (see equation 28).
b  b specifies the y-intercept, $\beta$, of the drag profile (see equation 29).
s  s specifies the minimum scaled distance, $s_0$, for the drag profile (see equations 33–35).
c  c specifies $c$, the integration constant from equation 31.
T  T specifies the minimum scaled time, $\tau$, for the DRAG profile (see equations 39–41).
d  d specifies $d$, the integration constant from equation 38.

8.2 Creating a Pointer to an Array of DRAG Structs: The Drags() Function

The Drags() function can be used to create a pointer to an array of DRAG structs. The code uses equation 32 to determine $M_{0,k}$; equation 28 to determine $\alpha_k$; equation 29 to determine $\beta_k$; equations 33–35 to determine $s_{0,k}$; equation 31 to determine $c_k$; equations 39–41 to find $\tau_k$; and equation 38 to determine $d_k$. If $\beta_k$ is found to be very close to zero, it is replaced with the user-definable value $b_{\text{min}}$.

Note that the Drags() function uses the “new” command to allocate memory for the array of DRAGs that is pointed to by the return value. Thus, to avoid memory leaks, each use of the Drags() function should be accompanied by a use of the “delete[]” operator.

Drags() Code

```cpp
inline DRAG*Drags( //<===================================CREATES A SET OF DRAGS
    const double*M, //<--------------MACH VALUES (MUST BE UNIQUE & DECREASING)
    const double*Cd, //<------CORRESPONDING Cd VALUES (FROM F=-.5*rho*A*Cd*v^2)
    int n, //<----------------------------------------THE NUMBER OF MACH VALUES
    double b_min=1E-8) //<----------------------------------MINIMUM BETA VALUE
    DRAG*D=new DRAG[n-1];
    for(int k=0,j=k-1;k<n-1;++k,++j){
        D[k].M=M[k];
        D[k].a=(Cd[k]-Cd[k+1])/(M[k]-M[k+1]);
        D[k].b=Cd[k]-D[k].a*M[k];/*<-*/if(fabs(D[k].b)<b_min)D[k].b=b_min;
        D[k].s=!k?0:1/D[j].b*log(D[j].a+D[j].b/M[k])+D[j].c;
        D[k].c=D[k].s-log(D[k].a+D[k].b/M[k])/D[k].b;
        D[k].T=!k?0:(1/M[k]-D[j].a/D[j].b*log(D[j].a+D[j].b/M[k]))/D[j].b+D[j].d;
        D[k].d=D[k].T-(1/M[k]-D[k].a/D[k].b*log(D[k].a+D[k].b/M[k]))/D[k].b;}
    return D; //.....................note that D points to newly allocated memory
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST-UPDATED~09DEC2013~~~~~~
Drags() Parameters

\( M \)  \( M \)  \( M \) points to an array of Mach values. The Mach values must be unique and listed in decreasing order.

\( Cd \)  \( Cd \)  \( Cd \) points to an array of drag coefficients. There must be a drag coefficient for each Mach value.

\( n \)  \( n \)  \( n \) specifies the number of elements contained in the array that is pointed to by \( M \).

\( b_{\text{min}} \)  \( b_{\text{min}} \)  \( b_{\text{min}} \) specifies the smallest value that \( \beta_s \) is allowed to have (see the discussion that follows equation 25).

Drags() Return Value

The Drags() function returns a pointer to an array of DRAG structs.

8.3 Calculating Scaled Distance as a Function of Mach Number: The SofM() Function

The SofM() function uses equation 45 to calculate scaled distance as a function of Mach number.

SofM() Code

```c
inline double SofM(  
    const DRAG*D,  
    int n,  
    double M)  
{  
    int k;  
    for(k=n-2;k>0&&M>D[k].M;--k);  
    return 1/D[k].b*log(D[k].a+D[k].b/M)+D[k].c;  
}
```

SofM() Parameters

\( D \)  \( D \)  \( D \) points to an array of DRAG structs. The CreateDrags() function can be used to create a pointer to an array of DRAG structs.

\( n \)  \( n \)  \( n \) specifies the number of Mach elements used to create \( D \). Note that \( n \) is one less than the size of \( D \).

\( M \)  \( M \)  \( M \) specifies a Mach number. Use equation 1 to convert from speed to Mach number.

SofM() Return Value

The SofM() function returns a scaled distance. Use equation 2 to convert from scaled distance to measured distance.

8.4 Calculating Mach Number as a Function of Scaled Distance: The MofS() Function

The MofS() function uses equation 46 to calculate Mach number as a function of scaled distance.
MofS() Code

```c
inline double MofS(  
    const DRAG*D,//<----------------------DRAG PROFILES (CREATE USING Drags())
    int n,//<----------NUMBER OF MACH VALUES USED TO CREATE D (NOT SIZE OF D!)
    double s){//<----------------------------SCALED-DISTANCE (s=rho*A/(2*m)*x)
  int k;/*<-*/for(k=n-2;k>0&&s<D[k].s;--k);
  return D[k].b/(exp(D[k].b*(s-D[k].c))-D[k].a);//.......................M=v/c
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~
```

MofS() Parameters

**D**  
Points to an array of DRAG structs. The CreateDrags() function can be used to create a pointer to an array of DRAG structs.

**n**  
Specifies the number of Mach elements used to create **D**. Note that **n** is one less than the size of **D**.

**s**  
Specifies a scaled-distance. Use equation 2 to convert from measured distance to scaled distance.

MofS() Return Value

The MofS() function returns a Mach number. Use equation 1 to convert from Mach number to speed.

8.5 Calculating Scaled Time: The TofM() Function

The TofM() function uses equation 47 to calculate scaled time as a function of Mach number.

TofM() Code

```c
inline double TofM(  
    const DRAG*D,//<----------------------DRAG PROFILES (CREATE USING Drags())
    int n,//<----------NUMBER OF MACH VALUES USED TO CREATE D (NOT SIZE OF D!)
    double M){//<------------------------------------------MACH NUMBER (M=v/c)
  int k;/*<-*/for(k=n-2;k>0&&M>D[k].M;--k);//          T=rho*A/(2*m)*c*t ----.
  return(1/M-D[k].a/D[k].b*log(D[k].a+D[k].b/M))/D[k].b+D[k].d;//<-----------'
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~~
```

TofM() Parameters

**D**  
Points to an array of DRAG structs. The CreateDrags() function can be used to create a pointer to an array of DRAG structs.

**n**  
Specifies the number of Mach elements used to create **D**. Note that **n** is one less than the size of **D**.

**M**  
Specifies a Mach number. Use equation 1 to convert from speed to Mach number.
TofM() Return Value

The TofM() function returns a scaled time. Use equation 3 to convert from scaled time to measured time.

8.6 Example: Calculating Speed and Time of Flight for the Lapua Scenar GB528 Rifle Bullet

The following example uses information from Wikipedia’s External Ballistics page to calculate speed and time of flight as functions of total distance traveled for a Lapua Scenar GB528 rifle bullet. Calculated values are compared to Doppler radar measurements, which were also obtained from Wikipedia’s External Ballistics page.

```c
#include <cstdio>
#include "y_traj_1d.h"  // yTraj1D, cmath\{fabs()\}

int main() {
    double M[] = {2.4, 2.2, 2, 1.8, 1.6, 1.5, 1.4, 1.3, 1.2, 1.15, 1.1, 1.075, 1.05, 1.025, 1, .975, .95, .925, .9, .875, .85, .825, .8, .75, .7, .65, .6, .5, .4, 0};
    double A = 3.14159 * pow(8.59/2000, 2);  // cross-sectional area of projectile (m^2)
    double m = 19.44/1000;  // mass of projectile (kg)
    double v0 = 830;  // initial speed of projectile (m/s)
    double rho = 1.225;  // density of air at sea level (kg/m^3)
    double c = 340.3;  // local speed of sound (m/s)
    double v_r[] = {830, 711, 604, 507, 422, 349, 311, 288, 267, 247, 227};
    double t_r[] = {0, .3918, .8507, 1.3937, 2.0435, 2.8276, 3.748, 4.7522, 5.8254, 7.0095, 8.2909};

    printf("%13s       | calc.    ref.         |   calc.    ref.    
", "");
    printf("%13s range | speed   speed     %%   |   time     time     %%
", "");
    printf("%13s  (m)  | (m/s) , (m/s) , diff. |   (s)  ,   (s)  , diff.
", "");
    printf("%13s======================================================
", "");

    for (int i=0; i<11; ++i) {
        double v = yTraj1D::MofS(D, 28, yTraj1D::SofM(D, 28, v0/c) + 300*i*rho*A/(2*m)*c;
        double t = (yTraj1D::TofM(D, 28, v/c) - yTraj1D::TofM(D, 28, v0/c)) * 2*m/(rho*A*c);
        printf("%19.1f |%6.1f ,%6.1f ,%5.1f  |%7.4f ,%7.4f ,%5.1f
", 300.0*i, v, v_r[i], fabs(v_r[i]-v)/(v_r[i]+v)/2)*100, t, t_r[i], i==0?0:fabs(t_r[i]-t)/(t_r[i]+t)/2)*100;}
}
```

// YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~09DEC2013~~~~~
8.7 Example: Calculating Scaled Time of Impact for Fragment Trajectories

The following example simulates the trajectories of fragments produced by a fragmentation weapon. The code generates 1,000,000 fragments, all with the same starting location. Each fragment is given a randomly chosen velocity. Direction is uniformly distributed across all possible directions. Speed is uniformly distributed with values from Mach 1.0 to Mach 3.0.

A rectangular surface that is 80.0 × 40.0 scaled length units is placed with its center 10.0 scaled length units away from the fragment source. The surface is oriented such that, at its center, it is perpendicular to a ray originating from the fragment source.

Distance to point of impact is calculated using functions from the y3DOps namespace.6 The MofS() function is used to find each fragment’s speed at the impact location. The TofM() function is used to find the scaled time of impact.

```c
#include <cstdio> // fclose(), FILE, freopen(), printf(), stdout
#include <cstdlib> // rand(), RAND_MAX
#include "y_3d_ops.h" // drags(), y3DOps
#include "y_traj_1d.h" // yTraj1D, <cmath>{cos(), sin(), sqrt()}
int main(){
    FILE*f=fopen("impacts.txt","w",stdout); // redirect output to a file
    double M[]={3.2,1.5,1.5,1.5,1.5,1.5},C[]={.45,.6,.6,.6,.6,.6}; // drag coefficient
    yTraj1D::DRAG*D=yTraj1D::Drags(M,C,6);
    double T[]={10,40,20,-40,20,20,-10,40,-20}; // rectangular target
    for(int i=0;i<1000000;++i){
        double M0=2*rand()/RAND_MAX+1; // initial speed
        double phi=2*3.14159265358979*rand()/RAND_MAX; // random azimuthal angle
        double theta=acos((2.*rand())/RAND_MAX-1); // random inclination angle
```

The results are shown in figure 9. Since all time and length units are scaled, the image presented in figure 9 is independent of fragment mass, fragment cross-sectional area, and air density. Note the color mixing that results from selecting random initial speeds.

![Figure 9. Scaled time of impact, $\tau$, for fragments striking a plate.](image)

9. **Code Summary**

A summary sheet is provided at the end of this report. It presents the yTraj1D namespace, which contains the struct and four functions that are described in sections 8.1–8.5. Also presented are the examples found in sections 8.6 and 8.7.
In this paper, we focus on the following tasks:

1. **Impacts Example**: The main function `main` is shown, which initializes parameters such as `M`, `C`, `x`, `theta`, and `t`. The function `yTraj1D::TofM` is used to calculate the trajectory. The code contains multiple nested loops and conditional statements to iterate through different scenarios.

2. **Validating Example**: This section validates the model by comparing the calculated trajectory with expected results. A table is provided to compare the calculated and expected values, along with a graph showing the output from the example.

3. **yTraj1D Summary**: This summary provides a high-level overview of the class `yTraj1D`, which includes methods for calculating trajectory based on various parameters.

The code snippet includes a variety of C++ features such as `#include`, `inline`, `struct`, and `namespace`. The use of `printf` for output and `for` loops for iterations is also evident.
<table>
<thead>
<tr>
<th>NO. OF COPIES</th>
<th>ORGANIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEFENSE TECHNICAL INFORMATION CTR DTIC OCA</td>
</tr>
<tr>
<td>2</td>
<td>DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL &amp; RECORDS MGMT</td>
</tr>
<tr>
<td>1</td>
<td>GOVT PRINTG OFC A MALHOTRA</td>
</tr>
<tr>
<td>1</td>
<td>DIR USARL RDRL WML A R YAGER</td>
</tr>
</tbody>
</table>