PDHonline Course G294 (4 PDH)

Explosion Standoff Distances

Instructor: Marvin Liebler, P.E.

2012

PDH Online | PDH Center

5272 Meadow Estates Drive
Fairfax, VA 22030-6658
Phone & Fax: 703-988-0088
www.PDHonline.org
www.PDHcenter.com

An Approved Continuing Education Provider

1. INTRODUCTION

1.1 EXPLOSIVES -- DEFINITION

The dictionary defines an explosion as a rapid chemical or nuclear reaction with production of heat and violent expansion of gas. For blast loads under consideration here, the explosive must be detonated to produce an explosion, rather than a fire. Detonation is a rapid and stable reaction which proceeds through the explosive, at speeds of approximately 25000 ft/sec (hypersonic).
This detonation rapidly converts the solid or liquid explosive into hot, dense, high pressure gas and this gas volume is the source of strong blast waves in air.

1.2 EXPLOSIVE MATERIALS

Explosive materials are characterized, for study purposes, as TNT equivalents, as shown in the following charts in Reference 5.1,showing single and multi-component explosives.

1.3 BLAST LOADING CATEGORIES

The three unconfined (as contrasted with confined explosions which are not covered here) loading categories are free air burst, air burst, and surface burst.
A free air burst is one in which the shock wave proceeds outward from its origin with no intermediate amplification.
Air bursts occur such that the ground reflections impact the structure prior to the arrival of the blast wave.
When the explosion is located close to the ground surface such that the initial shock is amplified at the point of detonation, this is called a surface burst. The surface burst will be the type considered here, as it is the most common.

1.4 SURFACE BLAST SCALING LAWS

The pertinent properties of surface blast explosions are given for a range of standoff distances and explosive weight by the dimensionless scaled distance parameter Z=R/W^(1/3).
Pertinent parameters are defined on the page 90 of Reference 5.2

TERM	DEFINITION	UNITS
Pr	peak positive reflected normal pressure	psi
Pso	peak positive incident pressure. This is the basic incident loading characteristic. It is not uniform over the surface, but varies with vertical and horizontal distance.	psi
Pt	Pr+Pso	psi
R	standoff distance from explosion	ft

to element
to duration of positive phase of ms
blast wave
W explosive weight, converted to TNT lbf
Z scaled ground distance=R/W^(1/3) ft/lbf^(1/3)

It is very convenient to automate the selection of Pt=Pr+Pso and to, using a polynomial approximation.
This is done, in general, by:

(1) Carefully select points of (x,y) where x is the independent variable and y the dependent variable.
(2) Express each dependent variable in the form:
y = cn*x^n + c(n-1)*x^(n-1) + …… +c1*x + c0
There will be n+1 equations in n+1 unknowns.
Eight points at Z=.5,.9,2,5,10,20,50,and 100 were chosen. So that the equations are:
y = c7*x^7 + c6*x^6 + …… +c1*x + c0.
Note that for each dependent variable there are eight equations in 8 unknowns.
In this case, x = log10(Z) and y = Pt=Pr+Pso, or log10(to/W^(1/3)).
(3) Solve for the 'c' coefficients, enabling solution of The dependent variable at any point. See the heading of source code "interpolation.c" for details of solution.

PROCEDURE FOR FINDING BLAST WAVE PARAMETERS

1. Find point of interest on ground → R.
2. Assume charge weight W (TNT or TNT equivalent)
3. Apply safety factor to W, if required.
4. Find scaled standoff distance Z.
5. Find desired scaled parameter values from chart.
6. Multiply scaled values by W^(1/3) to obtain absolute values for to, but not Pt, which is Given directly as Pr+Pso

Using these methods, construct a table for a weight of 10 000 lbf of TNT for standoff distances of 50,100, and 200 feet. Find Pt, to, and ½ Pt*to for each case
Here W^(1/3) = 21.544 lbf^(1/3)

R	Z	Pt(psi)	to(sec)	½ Pt*to (psi-sec)
50	2.321	1744.439	.015164	26.453
100	4.642	249.526	.028212	7.040
200	9.283	38.611	.051168	1.976

Notice how Pt falls off with increasing standoff distance. This is somewhat balanced by the increase in to. ½ Pt*to is the area of the triangle formed by coordinates (0,0),(to,0), and (0,Pt).

2. MATHEMATICAL MODELS OF EXPLOSION

2.1 IMPULSE INPUT -- This model uses an input of finite area and zero width. It has the distinct advantage of being the easiest to solve, for a given struck element. It depends on to << 1/wn , wn =natural frequency of element = square root of stiffness divided by mass of element, for elastic action.
It is given numerically by ½ Pt*to, as shown above.

2.2 TRIANGULAR INPUT -- This model uses a right triangular input with initial value of Pt psi and final value of zero at t = to, with a straight line decreasing pressure variation.
This is more precise than the impulse method, but less precise than the Friedlander equation following. It is more conservative, in general, than the Friedlander equation, if the same Pt and to are used.

2.3 FRIEDLANDER EQUATION

p = Pt*(1-t/to)*e^(-bt/to) where p = pressure at any time t, b affects the duration of the negative phase and e = base of natural logarithms. * denotes multiplication. t is measured from the start of the pulse.
Some authors (Reference 5.3) recommend using b=1 for each case. That approach will examined later.

3. SDOF (SINGLE DEGREE OF FREEDOM) ANALYSIS

A SDOF model is used for first analyses of buildings, walls, columns, beams and slabs.
The basic equation of this analysis, in the elastic range, is:
m*x''+c*x'+k*x=f(t) where:
m = mass in lbm (lbf/(in./sec^2))
c = viscous friction (lbf/(in./sec))
k = stiffness (lbf/in.)
x''= acceleration (in./sec^2)
x' = velocity (in./sec)
x = displacement (in.)
f(t) = applied force (lbf) vs. time

3.1 NUMERICAL SOLUTION METHODS

3.1.1 LAPLACE TRANSFORM METHOD

The Laplace transform method is used to solve linear differential equations, such as the one above. It substitutes algebraic equations in frequency for differential equations in time. Solutions for displacement, velocity, and acceleration at any point in time may be obtained. Nonlinear equations, such as elastic-plastic may only be solved by application of different differential equations at different points in time. The process is

expedited by tables of direct (time → frequency) and inverse (frequency → time) transform pairs.

A short table useful for the equations here is:

F(s)	f(t)
1/s	1
1/(s+a)	e^(-at)
1/(s+a)^2	te^(-at)
1/((s(s+a))	(1/a)(1-e^(-at))
1/(s^2(s+a))	(1/a^2)(at-1+e^(-at))
1/(s(s+a)^2)	(1/a^2)(1-e^(-at)-ate^(-at))
1/(s^2+2ζwns+wn^2)	(1/z)e^(-ζwnt)sin(zt) z=wn*sqrt(1-ζζ)
1/(s(s^2+2ζwns+wn^2))	1/wn^2-(1/wnz)e^(-ζwnt)* sin(zt+cos^(-1)(ζ)
1/(s^2(s^2+2ζwns+wn^2))	t/wn^2-2ζ/wn^3+(1/(wn^2*z))* e^(-ζwnt)* sin(zt+cos^(-1)(2ζζ-1))
1/(s(s+a)(s+b))	(1/ab)(1-(b/(b-a))e^(-at) +(a/(b-a))e^(-bt)
1/(s(s+a)(s+b)^2)	(1/ab^2)-(1/(a(b-a)^2)e^(-at) +((2b-a)/(b^2(b-a)^2)e^(-bt) +(t/(b(b-a)))e^(-bt)

Inversion methods include partial fraction expansion and the convolution integral.

3.1.2 THE RUNGE--KUTTA METHOD

The Runge--Kutta method is iterative, requiring the six (6)solutions, in sequence shown,at each time interval, where u''=f(u,u',t) :

1. k1 = h*f(un,un',tn)
2. k2 = h*f(un+h*un'/2+h*k1/8,un'+k1/2,tn+h/2)
3. k3 = h*f(un+h*un'/2+h*k1/8,un'+k2/2,tn+h/2)
4. k4 = h*f(un+h*un' +h*k3/2,un'+k3, tn+h)
5. u(n+1) = un+h*(un'+(k1+k2+k3)/6)
6. u(n+1)'= un'+(k1+2*(k2+k3)+k1)/6

where :

h	= time interval, seconds
*	= multiplication
un	= displacement at nth interval
un'	= velocity at nth interval
u(n+1)	= displacement at (n+1)th interval
u(n+1)'	= velocity at (n+1)th interval

Starting conditions are generally u(0) = u(0)' = 0

3.2 LINEAR (ELASTIC) RANGE

3.2.1 IMPULSE INPUT -- ELASTIC ANALYSIS

This case is the most straightforward, as well as being, in general, conservative. The general equation of motion is:

mu''+cu'+ku = ½ Pt*A*to*δ(t) where δ(t) is the Dirac delta function, an infinitely sharp peak with zero duration and unit area. Dividing the above equation by m and taking Laplace transforms, with u(0) = u'(0) = 0 and

k/m = wn^2, c = 2*ζ*wn*m

s^2*U(s)+2*ζ*wn*s+wn^2 = Pt*A*to/2*m

The inverse transform, i.e., the time displacement of the SDOF is :

```
u(t) = (Pt*A*to/(2*m))*(1/(wn*(1-ζζ)^½ )*e^(-ζ*wn*t)*
       sin(wn*(1-ζζ)^½*t)
```

The maximum occurs when u'(t) = 0. It can be shown that this occurs when :

tmax = θ/(wn*(1-ζζ)^½) and θ = tan^(-1)((1-ζζ)^½/ζ)

A program implementing this response is dirac.c.

3.2.2 RIGHT TRIANGLE INPUT

A new function is given here, namely 1(expression). This equals zero when expression evaluates to < zero, and equals one when expression evaluates to > = zero.

f(t)= applied force = Pt*A*1(t)*(1-t/to) + 1(t-to)*(-1+t/to)

Notice how the delayed function cancels the input after t=to. Taking Laplace transforms,

```
U(s) = (Pt*A/m)*(1/(s*(s^2+2*ζ*wn*s+wn^2))-
         (1/to)*(1/(s^2*(s^2+2*ζ*wn*s+wn^2))+
         (1/to)*(1/(s^2*(s^2+2*ζ*wn*s+wn^2)))*e^(-to*s))
```

After a fair amount of work, this transform is inverted to :

```
u(t) = (Pt*A/m)*(1/z)*(e^(-ζ*wn*t)*sin(z*t+cos^(-1)(ζ))
       -(1/to)*(t/wn^2-2*ζ/wn^3+(1/z)*
        (sin(z*t+cos^(-1)(2ζζ-1)))
       +(1/to)*1(t-to)*((t-to)/wn^2)-2*ζ/wn^3+(1/z)*
        sin(z*(t-to)+cos^(-1)(2*ζ*ζ-1))))
```

This function is evaluated in triangle1.c

A second method of solution is iteration of the equation for acceleration. If carried out to small enough intervals, it should match the transform method to any desired number of places. In general, it is much easier to implement than the transform method, and its comparison with the transform method helps cut down on arithmetic errors.

x''(t) = -wn^2*un(t) - 2*ζ*wn*un'(t) + f(t)/m

where f(t) = Pt*A*(1-t/to) for t < 0
= 0 for t>= to

The approach is outlined in section 3.1.2 above and in program triangle2.c.

3.2.3 FRIEDLANDER EQUATION

First examine solution by the transform method.
The basic equation may be written as:
u''+(c/m)u'+(k/m)u =f(t)/m = (1/m)pA
Using the Friedlander equation,
u''+(c/m)u'+(k/m)u = (1/m)PtA(1-t/to)e^(-bt/to)
Let wn^2 = k/m, square of natural frequency of element, and
c/m =2ζwn, and ζ = damping ratio (generally less than 1)
u''+2ζwnu'+wn^2u = (1/m)PsoA(1-t/to)e^(-bt/to)

Taking Laplace transforms of each side,
s^2*U(s)+2*ζ*wn*s*U(s)+wn^2*U(s)=(1/m)*Pt*A*(1/(s+b/to)
-(1/to)*(1/(s+b/to)^2
U(s)=(1/m)*Pt*A*[1/(s+b/to)-(1/to)*(1/(s+b/to)^2}*
|<----------------F(s)---------------------->|
1/(s^2+2*ζ*wn*s+wn^2)
|←--------G(s)----→|
This multiplication in the frequency domain (F(s)*G(s))
corresponds to convolution in the time domain.
Let f(u) = (1/m)*Pt*A*(e^(-b*u/to)-(1/to)*u*e^(-b*u/to)) =
inverse transform of F(s) and
g(u) = (1/b1)*e^(-ζ*wn*u)*sin(b1*u) = inverse transform
of G(s), where b1=wn*sqrt(1-ζζ).
The convolution theorem states

$$u(t) = \int_0^t f(t-u)*g(u)du$$

After a fair amount of work we have :

$$u(t) = c0*[e^(-ζ*wn*t)*(c1*sin(b1*t)+c2*cos(b1*t))+ e^(-b*t/to)*(c3*(1-t/to)+c4)]$$

where : c0 = Pt*A/(b1*m)
c1 = (to*(a^2+b1^2)*a-(a^2-b1^2))/
(to*(a^2+b1^2)^2)
a = b/to-ζ*wn
c2 = (to*(a^2+b1^2)*(-b1)+2*a*b1)/
(to*(a^2+b1^2)^2)
c3 = b1/(a^2+b1^2)
c4 = -2*a*b1/(to*(a^2+b1^2)^2

$$u'(t) = c0*[e^(-ζ*wn*t)*(b1*c1*cos(b1*t)-b1*c2*sin(b1*t))- e^(-ζ*wn*t)*ζ*wn*(c1*sin(b1*t)+c2*cos(b1*t)) + e^(-b1*t/to)*(-c3/to) - e^(-b1*t/to)*(b1/to)*(c3*(1-t/T)+c4)]$$

See program frdlndr1.c for source code for above, as
region1.

The second method to solve this linear SDOF problem is
an iterative method, here being the Runge-Kutta method. It
is advisable to solve each problem by both methods, to
reduce the chance of error.

The iterative equations, solved in sequence shown, are :
(1) k1=delta*(-wn*wn*un0-2*ζ*wn*un0'+
Pso*A/m)*(1-t/to)*e^(-b*t/to))
(2) k2=delta*(-wn*wn*(un0+delta*un0'/2+delta*k1/8)-

```
      2*ζ*wn*(un0'+k1/2)+
      (Pso*A/m)*(1-(t+delta/2)/to)*e^((-b/to)*(t+delta/2)))
(3)   k3=delta*(-wn*wn*(un0+delta*un0'/2+delta*k1/8)-
      2*ζ*wn*(un0'+k2/2)+
      (Pso*A/m)*(1-(t+delta/2)/to)*e^((-b/to)*(t+delta/2)))
(4)   k4=delta*(-wn*wn*(un0+delta*un0'+delta*k3/2)
      -2*ζ*wn*(un0'+k3)+
      +(Pso*A/m)*(1-(t+delta)/to)*e^((-b/to)*(t+delta)))
(5)   un0 =un0+delta*(un0'+(k1+k2+k3)/6)
(6)   un0'=un0'+(k1+2*(k2+k3)+k4)/6
```

Iterative methods have the advantage of not requiring explicit solutions, but have the disadvantage of generally requiring a number of iterations for accuracy.
It is recommended that delta be <= 0.001 times either to or 1/wn, whichever is less. This source code for this program is frdlndr2.c as region 1.

3.3 NONLINEAR (ELASTIC-PLASTIC) RANGE -- FRIEDLANDER EQUATION

A simplified two part resistance function is used here to model elastic-plastic behavior, described as :

(1) Linear response from initial conditions (u=u'=0) to elastic limit. The equation describing this region is: u(t)''+2*ζ*wn*u(t)'+wn^2*u(t)=p(t)/m
R = resistance = k*u(t). This is called region 1. This region continues until R = Rmax, if it occurs. The only addition to the previous linear equation is continuing calculation of u and u' and monitoring if u' goes to zero, which signals the end of this region (region 2).

(2) Plastic region with R=Rmax and described by: u(t)''+(c/m)*u(t)'+Rm=p(t)/m.
This continues until u(t)'=0, indicating maximum displacement.

(3) The crossover time from linear to plastic regions is found by solving u(t) = umax for t as shown in program crossover1.c, where u(t) is the unknown displacement. The program uses the Newton-Raphson iteration method to solve that equation. It is advisable to get a starting value for the iteration from either frdlndr1 or frdlndr2, noting where u(t) crosses umax. If u(t) does not cross umax, then the solution is entirely linear.

The goals here are to first, determine if element reaction stays within the linear region, and secondly, If not, how deep into nonlinearity the element is forced.

3.3.1 TRANSFORM APPROACH TO REGION 2

```
u(t)''+(c/m)*u(t)'+Rmax/m=
(Pt*A/m)*(1-t1/to)*e^(-b*t1/to), using Friedlander
Equation, and t1 = t+crossover time
Now take Laplace transforms of each side of the
equation.
U(s)(s^2+sc/m) =
u(0)/(s+c/m)+(u'(0)+(c/m)u(0))/(s(s+c/m))-
     (Rm/m)/(s^2(s+c/m))+d0/(s(s+c/m)(s+b/to))-
     d1/(s(s+c/m)(s+b/to)^2)
where   d0 = (PtA/m)(e^(-btcr/to))(1-tcr/to)
 and  d1 = (PtA/m)(e^(-btcr/to))/to, tcr = crossover time
```

After a fair amount of work, the following equations are obtained :

```
t       =       t-cro;
d0      =       (Pt*A/m)*exp(-b*cro/to)*(1-cro/to);
d1      =       (Pt*A/m)*exp(-b*cro/to)*(1/to);
e1      =       exp(-(c/m)*t);
e2      =       exp(-(b/to)*t);
f1      =       1/((b/to)-(c/m));
f2      =       (b/to)/((b/to)-(c/m));
f3      =       1/((b/to)*((b/to)-(c/m)));
f4      =       (c/m)/(b/to-c/m);

*un0    =       *u0*e1+(*du0+(c/m)*(*u0))*(m/c)*(1-e1)-
                (Rm/m)*(m/c)*(m/c)*((c/m)*t-1+e1)+
                d0*(m*to/(b*c))*(1+f4*e2-f2*e1)-
                d1*(1/((c/m)*(b/to)*(b/to))-
                (m/c)*f1*f1*e1+
                (2*b/to-c/m)*f3*f3*e2+
                (to/b)*f3*t*e2);
*dun0   =       -(c/m)*(*u0)*e1+(*du0+(c/m)*(*u0))*e1-
                (Rm/c)*(1-e1)+
                d0*(m*to/(b*c))*(-f4*b*e2/to+f2*c*e1/m)-
                d1*(f1*f1*e1-(2*b/to-c/m)*f3*f3*b*e2/to+
                          f3*(e2+t*(-b/to)*e2));
```

3.3.2 ITERATIVE APPROACH TO PLASTIC REGION (REGION 2)

```
In this case,

u''(t)      =     (-c/m)u'(t)-Rmax/m +
                  (PtA/m)(1-t1/to)e^(-(b/to)t1)
where t1 = local t + crossover time.
No reference is made to the displacement value
u(t) in the acceleration calculations (k1-k4),
and time resets to zero at t=cro, except for the
Friedlander equation.
```

```
t    = t-cro;
d0   = (Pt*A/m)*exp(-b*cro/to)*(1-cro/to);
d1   = (Pt*A/m)*exp(-b*cro/to)*(1/to);
k1   = delta*(-(c/m)*(*dun0)-Rm/m+d0*exp(-b*t/to)-
       d1*t*exp(-b*t/to));
k2   = delta*(-(c/m)*(*dun0+k1/2)-Rm/m+d0*exp(-
       b*(t+delta/2)/to)-
```

```
        d1*(t+delta/2)*exp(-b*(t+delta/2)/to));
  k3   = delta*(-(c/m)*(*dun0+k2/2)-Rm/m+d0*
        exp(-b*(t+delta/2)/to)-
        d1*(t+delta/2)*exp(-b*(t+delta/2)/to));
  k4   = delta*(-(c/m)*(*dun0+k3)-Rm/m+d0*exp(-b*(t+delta)/to)-
        d1*(t+delta)*exp(-b*(t+delta)/to));
  *un0 = *un0+delta*(*dun0+(k1+k2+k3)/6);
  *dun0= *dun0+(k1+2*(k2+k3)+k4)/6;
```

3.4 Selection of Constant 'b' in the Friedlander Equation

Choice of 'b' is difficult to get from the literature. Reference 5.3 gives a range from .1 to 10, with a recommended value of 1. Given no other information, a conservative approach is to use the value of b which maximizes the area of t<=to minus the area of t>to. This should provide for maximum deflection for a given Pt and to.

The first area is $\int_{o}^{to} (1-t/to)e^{(-bt/to)}dt = to/b$

and the second $\int_{to}^{infinity} (1-t/to)e^{(-bt/to)}dt = to/b^2$

To maximize this function, find the first derivative and set equal to zero.

$f(b) = 1/b-1/b^2$

$f'(b)=-1/b^2+2/b^3 = 0 \rightarrow 2b^2 = b^3 \rightarrow b = 2$

As a check on this result, calculate f(b) from b = .1 to 10.

b	f(b)
.1	-490.110
.5	-11.049
1	0.079
1.5	1.230
2	1.384
2.5	1.328
3	1.229
4	1.037
5	.885
6	.768
7	.678
8	.605
9	.547
10	.498

It is seen that the maximum occurs at b=2, which will be used in further calculations.

4. Example

4.1 GIVEN

Concrete Beam with:
span = 20'
tributary height = 10'
pinned end connections

As=As'= 3.60 in.^2 (6#7)
fc' = 4000 psi
dy = 60000 psi
b = 18"
d' = 1.5625" (¾" cover)
h = 12"
d = 10.4375"
weak axis bending, remainder
top + bottom bars not listed

4.2 FIND

Standoff distances for W = 100,1000,and 10 000 pounds of TNT for moment Mn.

4.3 SOLUTION

4.3.1 Concrete Beam Strength Parameters

a = βc, depth of stress block, in.
As = tension steel area, in.^2
As' = compression steel area, in.^2
As1 = balances force of compression concrete, in.^2
As2 = balances force of compression steel, in.^2
b = width of section, in.
c = distance from outermost compression fiber to neutral axis, in.
d = distance from outermost compression fiber to centroid of As, in.
d' = distance from outermost compression fiber to centroid of comp. steel, in.
d1 = distance from outermost compression fiber to centroid of compression concrete, in.
Es = steel modulus of elasticity, psi
fc' = specified compressive strength of concrete
fs' = compression steel stress, psi
fy = steel yield stress, psi
Mn = nominal flexural strength of section, lbf-in.
Mn1 = nominal flexural strength due to As1 and compression concrete, lbf-in.
Mn2 = nominal flexural strength due to As2 and compression steel, lbf-in.

In this case, since As = As', fs' must be less than fy, and the following equations hold :

(1) .003(c-d') = fs'/Es equality of strains
(2) As2 = As'fs'/fy As2 balances As' force
(3) .85fc'(ab-As') = fy(As-As2)
As1 balances compression concrete
(4) a = βc stress block assumption

Equations 1-4 (unknowns in a, As2, c, fs') may be solved for a as:

(.85fc'b)a^2+(.003As'Es-fyAs-.85fc'As')a
-.003As'Esβd'=0
3400(18)a^2+23600(3.6)a-73950(3.6)1.5625 = 0
This solves to a = 2.003784 in.

d1 = depth of concrete stress block
d1 = (ab(c/2)-As'd')/(ab-As')
d1 = .939733005 in.

M1n = .85fc'(ab-As')(d-d1)
M1n = 979725.4 lbf-in.
fs' = (fyAs-.85fc'(ab-As'))/As'
fs' = 29335.7 psi
M2n = 937274.6 lbf-in.
Mn = Mn1+Mn2
Mn = 1917000 lbf-in.

Check:
FyAs1 = .85fc'(ab-As') → As1 = 1.83986 in.^2
FyAs2 = fs'As' → As2 = 1.76014 in.^2
As = As1+As2 = 3.60000 in.^2, Q.E.D.

4.3.2 Cracking Moment of Inertia

The following terms are defined to specify the cracked moment of inertia :

Icr = cracked moment of inertia, in.^4
n = Es/Ec, dimensionless
ycr = distance from outermost compression fiber to neutral axis

Ec = 57000(fc')^½
Ec = 3604996 psi
Es = 29000000 psi
n = 8.044391

Find ycr by summing first moments of area about neutral axis, and equating sum to zero.

(½)bycr^2+(n-1)As'(ycr-d')-nAs(d-ycr) = 0
ycr = 3.844818 in.

Icr = Icomp.conc.+Icomp.steel+Itensionsteel
Icr = bycr^3/3+(n-1)As'(ycr-d')^2+nAs(d-ycr)^2
Icr = 227.346 +132.099 +1258.693
Icr = 1618.138 in.^4

4.3.3 Program Parameters

m = mass = (20x12x18x12/1728)x150 = 4500 lbm
umax = displacement at Mn, assumed end of elastic range
find w (distributed load in lbf/in.) for m = Mn
Mn = 1/8 wL^2
w = 8 Mn/L^2

w = 1917000/240^2 = 266.5 lbf/in.
total load = 266.5x240 = 63900 lbf
umax = 5wL^4/384EIcr
umax = 5(266.5)240^4/384(3604997)1618.138
umax = 1.971755 in.
k = stiffness = total load/Δ
k = 32407.674 lbf/in.
wn = natural frequency = (k/m)^½ = 2.683599 rad/sec
A = 20(10) = 200 ft^2 = 28800 in.^2
2ζwn = c/m → c = 2ζ(wn)m
let ζ =0.1
C = 2(.1)2.683599(4500)
C = 2415.239 lbf/in.

4.3.4 Calculation of Standoff Parameters
Figure 2-5 on page 90 of UFC 3-340-02 gives Pt and to/W^1/3 verse Z/W^1/3, the normalized standoff distance. We use 'interpolation to seek the value of Z which gives the closest displacement to umax = 1.971755 inches.

input form	W (lbf)	Z (ft/lbf^1/3)	Pt (psi)	to (sec)	standoff (ft)
impulse	100	4.14	348.452	.005491	19.216
	1000	6.09	114.564	.016733	60.900
	10000	9.45	37.052	.051821	203.594
triangle	100	4.14	348.452	.005491	19.216
	1000	6.09	114.564	.016733	60.900
	10000	9.45	37.052	.051821	203.594
Friedlander	100	2.92	946.189	.004050	13.553
	1000	4.29	314.049	.012208	42.900
	10000	6.34	102.465	.037374	136.591

The following conclusions are drawn from this example.

1. There is little difference between the impulse and triangle models.
2. The product ½*Pt*to = .9585 ± .0015 for impulse and triangle, with the last case being 1.916 ± .001, units being psi-sec = lbf*sec/in.^2

5. REFERENCES

5.1 "The Physics and Mechanisms of Primary Blast Injury", Stuhmiller,etal., Chapter 7 "Conventional Warfare Ballistics, Blast and Burn Injuries". To access this paper, go to www.google.com and type in the title.

5.2 "Structures to Resist the effects of Accidental Explosions", UFC 3-340-02, 12-05-08 www.hnd.usace.Mil/technology/engpubs.htm

5.3 "Elastic-Plastic Response Spectra for Exponential Blast Loading",Gantes, etal., International Journal of Impact Engineering, 2004, www.elsevier.com/locate/ijimpeng

6. SOURCE CODES

```
/* interpolation.c : x_vek    = logs of Z, Z=R/W^(1/3)
                     y_vek    = Pt = Pso+Pr
                     z_vek    = logs of to/W^(1/3)
                     unk1_vek = coefficient vector,Pt = sum vek1^i*z^i,i=0-7
                     unk2_vek = coefficient vector,to = sum vek2^i*z^i,i=0-7
                     pwr_mtrx = z7,.....,z1,1 for each data point, one data
                                point per row, 8 data points
                     inputs   = x_vek,y_vek,z_vek
                     outputs  = unk1_vek,unk2_vek
                     take values from left hand side of graph



                    _                          _   _       _     _  _
                   |  x0^N x0^(N-1).....x0 1 | | unk1+N |     | y0 |
                   |  .     .         .  .  . | |    .    |  = | .  |
                   |  .     .         .  .  . | |    .    |    | .  |
                   |  xN^N xN^(N-1).....xN 1 | | unk1+0 |     | yN |
                    -                          -   -       -     -  -

                   <-------logZ values------->  <-coeff->  <-Pt,logto->

                            pwr_mtrx          for 1 variable,
                                              i.e., c7,c6,…,c0,c1,c0

                   ynew=c7*xnew^7+c6*xnew^6+.....c1*xnew^1+c0
*/
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
        int i,j;
        double *x_vek,*y_vek,*z_vek,*unk1_vek,*unk2_vek,*pwr_mtrx;
        double base,exp,jay,wrkf;
        double logPt,logto,logZ,Pt,to,W,Z;
        double frak = 1.0/3.0;
        FILE *inn;
        FILE *out;
        void invert(double *);
        inn = fopen("interpolation.in","r");
        out = fopen("interpolation.out","w+");
        x_vek    = calloc(8,sizeof(double));
        y_vek    = calloc(8,sizeof(double));
        z_vek    = calloc(8,sizeof(double));
        unk1_vek = calloc(8,sizeof(double));
        unk2_vek = calloc(8,sizeof(double));
        pwr_mtrx = calloc(64,sizeof(double));
        for(i=0;i<=7;i++)
        {
                fscanf(inn,"%lf",&wrkf);
                *(x_vek+i) = wrkf;
        }
        for(i=0;i<=7;i++)
        {
                fscanf(inn,"%lf",&wrkf);
                *(y_vek+i) = wrkf;
        }
        for(i=0;i<=7;i++)
        {
                fscanf(inn,"%lf",&wrkf);
                *(z_vek+i) = wrkf;
        }
        fclose(inn);
        for(i=0;i<=7;i++)
        {
                *(pwr_mtrx+8*i+7) = 1;
        }
```

```
        for(i=0;i<=7;i++)
        {
                for(j=0;j<=6;j++)
                {
                        jay = j;
                        *(pwr_mtrx+8*i+j) = pow(*(x_vek+i),7-jay);
                }
        }
        invert(pwr_mtrx);
        for(i=0;i<=7;i++)
        {
                *(unk1_vek+i) = 0;
                *(unk2_vek+i) = 0;
        }
        for(i=0;i<=7;i++)
        {
                for(j=0;j<=7;j++)
                {
                        *(unk1_vek+i) += (*(pwr_mtrx+8*i+j))*(*(y_vek+j));
                        *(unk2_vek+i) += (*(pwr_mtrx+8*i+j))*(*(z_vek+j));
                }
        }
        for(i=0;i<=7;i++)
        {
                fprintf(out,"%14.6f\n",*(unk1_vek+i));
        }
        fprintf(out,"\n\n");
        for(i=0;i<=7;i++)
        {
                fprintf(out,"%14.6f\n",*(unk2_vek+i));
        }
        fclose(out);
        printf("enter W,Z\n");
        scanf("%lf %lf",&W,&Z);
        logZ = log10(Z);
        for(i=0;i<=6;i++)
        {
                logPt += (*(unk1_vek+i))*pow(logZ,7-i);
                logto += (*(unk2_vek+i))*pow(logZ,7-i);
        }
        logPt += *(unk1_vek+7);
        logto += *(unk2_vek+7);
        Pt = pow(10,logPt);
        to = pow(10,logto);
        to = to*pow(W,frak);
        printf("Pt= ");printf("%14.6f",Pt);printf(" psi \n");
        printf("to= ");printf("%14.6f",to);printf(" msec\n");
        return 0;
}
void invert(double *mtrx)
{
    int i,j,k;
    double *old_kol,*old_row,pivco;
    old_kol=calloc(8,sizeof(double));
    old_row=calloc(8,sizeof(double));
    for(i=0;i<=7;i++)
    {
        for(j=0;j<=7;j++)
        {
            *(old_kol+j)=*(mtrx+8*j+i);
            *(old_row+j)=*(mtrx+8*i+j);
        }
        pivco=*(mtrx+8*i+i);
        *(mtrx+8*i+i)=1/pivco;
        for(j=0;j<=7;j++)
        {
            if(j!=i)
            {
                *(mtrx+8*j+i)=+*(mtrx+8*j+i)/pivco;*(mtrx+8*i+j)=-*(mtrx+8*i+j)/pivco;
            }
```

```
        }
        for(j=0;j<=7;j++)
        {
            for(k=0;k<=7;k++)
            {
                if((j!=i)&&(k!=i))
                {
                    *(mtrx+8*j+k)=
                    *(mtrx+8*j+k)-*(old_kol+j)*(*(old_row+k))/pivco;
                }
            }
        }
    }
}


/* dirac.c : solution of damped SDOF driven by impulse : 11-30-09 : ml
*/
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
        int i;
        double A,b1,b2,delta,du,m,Pt,t,theta,tmax,to,ttotal,u,wn,zilch;
        double arg1,arg2,arg3,*input_data,wrkf;
        FILE *inn;
        FILE *out;
        input_data=calloc(8,sizeof(double));
        inn=fopen("dirac.in","r");
        out=fopen("dirac.out","w+");
        for(i=0;i<=7;i++)
        {
                fscanf(inn,"%lf",&wrkf);
                *(input_data+i)=wrkf;
        }
        fprintf(out,"  DIRAC.OUT\n\n");
        A      =       *(input_data+0);
        delta  =       *(input_data+1);
        m      =       *(input_data+2);
        Pt     =       *(input_data+3);
        to     =       *(input_data+4);
        ttotal =       *(input_data+5);
        wn     =       *(input_data+6);
        zilch  =       *(input_data+7);
        fclose(inn);
        fprintf(out,"  A     = ");fprintf(out,"%14.6f",A     );fprintf(out," in.^2\n");
        fprintf(out,"  delta = ");fprintf(out,"%14.6f",delta );fprintf(out," sec\n");
        fprintf(out,"  m     = ");fprintf(out,"%14.6f",m     );fprintf(out," lbm\n");
        fprintf(out,"  Pt    = ");fprintf(out,"%14.6f",Pt    ); fprintf(out,"lbf/in.^2\n");
        fprintf(out,"  to    = ");fprintf(out,"%14.6f",to    );fprintf(out," sec\n");
        fprintf(out,"  ttotal= ");fprintf(out,"%14.6f",ttotal);fprintf(out," sec\n");
        fprintf(out,"  wn    = ");fprintf(out,"%14.6f",wn    );fprintf(out," rad/sec\n");
        fprintf(out,"  zilch = ");fprintf(out,"%14.6f",zilch );fprintf(out,"\n\n");
        fprintf(out,"     sec         u           du     \n");
        fprintf(out,"  --------    --------    --------\n");
        arg1    =       1-zilch*zilch;
        b1      =       wn*sqrt(arg1);
        b2      =       sqrt(arg1);
        for(t=0;t<=ttotal+delta/2;t+=delta)
        {
                arg2    =       -zilch*wn*t;
                arg3    =       b1*t;
                u       =       (Pt*A*to/(2*m))*(1/b1)*(exp(arg2))*sin(arg3);
                du      =       (Pt*A*to/(2*m))*(1/b1)*(exp(arg2))*
                                (b1*cos(arg3)-zilch*wn*sin(arg3));
                fprintf(out,"%10.6f %10.6f %10.6f\n",t,u,du);
        }
        theta   =       atan(b2/zilch);
        tmax    =       theta/b1;
        fprintf(out,"\n\n");
```

```
        fprintf(out,"  tmax =  ");
        fprintf(out,"%10.6f",tmax);
        fprintf(out," sec\n\n");
        fclose(out);
        return 0;
}
```

```
/* triangle1.c : solution of equations of motion with three point input
   by Laplace transforms :  11-30-09 : points = (0,0),(0,Pso),and(to,0)
   A            =       affected area, in.^2
   delta        =       time step, seconds
   du           =       velocity, in./sec
   m            =       member mass, lbm
   Pt           =       magnitude of driving pressure,Pso+Pr, lbf/in.^2
   to           =       duration of positive driving force, seconds
   ttotal       =       total time of run, seconds
   u            =       displacement, in.
   wn           =       natural frequency of member, rad/sec =(k/m)^(1/2)
   zilch        =       fraction of critical damping
*/
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
        int i;
        double A,delta,Pt,m,to,ttotal,wn,zilch;
        double t,u,du;
        double c1,c2,c3,c4,c5,c6,c7,v1,v2,v3,v4;
        double *input_data,wrkf;
        FILE *inn;
        FILE *out;
        input_data=calloc(8,sizeof(double));
        inn=fopen("triangle1.in","r");
        out=fopen("triangle1.out","w+");
        for(i=0;i<=7;i++)
        {
                fscanf(inn,"%lf",&wrkf);
                *(input_data+i)=wrkf;
        }
        fprintf(out,"  TRIANGLE1.OUT\n\n");
        A      =       *(input_data+0);
        delta  =       *(input_data+1);
        m      =       *(input_data+2);
        Pt    =        *(input_data+3);
        to     =       *(input_data+4);
        ttotal =       *(input_data+5);
        wn     =       *(input_data+6);
        zilch  =       *(input_data+7);
        fprintf(out,"  A     = ");fprintf(out,"%14.6f",A     );fprintf(out," in.^2\n");
        fprintf(out,"  delta = ");fprintf(out,"%14.6f",delta );fprintf(out," sec\n");
        fprintf(out,"  m     = ");fprintf(out,"%14.6f",m     );fprintf(out," lbm\n");
        fprintf(out,"  Pt    = ");fprintf(out,"%14.6f",Pt    );fprintf(out," lbf\n");
        fprintf(out,"  to    = ");fprintf(out,"%14.6f",to    );fprintf(out," sec\n");
        fprintf(out,"  ttotal= ");fprintf(out,"%14.6f",ttotal);fprintf(out," sec\n");
        fprintf(out,"  wn    = ");fprintf(out,"%14.6f",wn    );fprintf(out," rad/sec\n");
        fprintf(out,"  zilch = ");fprintf(out,"%14.6f",zilch );fprintf(out,"   \n\n");
        fprintf(out,"     sec        u          du   \n");
        fprintf(out,"  --------   --------   --------\n");
        c1      =       Pt*A/m;
        c2      =       1/(wn*wn);
        c3      =       1/(wn*wn*sqrt(1-zilch*zilch));
        c4      =       1.470628906; //acos(zilch);
        c5      =       2.941257811; //acos(2*zilch*zilch-1);
        c6      =       wn*sqrt(1-zilch*zilch);
```

```
        c7      =       1/(wn*wn*sqrt(1-zilch*zilch));

        for(t=0;t<=ttotal+delta/2;t+=delta)
        {
                v1      =       exp(-zilch*wn*t);
                v2      =       wn*(sqrt(1-zilch*zilch))*t;
                v3      =       exp(-zilch*wn*(t-to));
                v4      =       wn*(sqrt(1-zilch*zilch))*(t-to);
                if(t<=to)
                {
                        u       =       c1*(c2-c3*v1*sin(v2+c4))-
                                        (c1/to)*(c2*t-2*zilch/(wn*wn*wn)+
                                        c7*v1*sin(v2+c5));
                        du      =       c1*(-c3*(v1*c6*cos(v2+c4)-
                                        zilch*wn*v1*sin(v2+c4)))-
                                        (c1/to)*(c2+c7*(v1*c6*cos(v2+c5)-
                                        zilch*wn*v1*sin(v2+c5)));
                }
                else
                {
                        u       =       c1*(c2-c3*v1*sin(v2+c4))-
                                        (c1/to)*(c2*t-2*zilch/(wn*wn*wn)+
                                        c7*v1*sin(v2+c5))+
                                        (c1/to)*(c2*(t-to)-
                                        2*zilch/(wn*wn*wn)+
                                        c7*v3*sin(v4+c5));
                        du      =       c1*(-c3*(v1*c6*cos(v2+c4)-
                                        zilch*wn*v1*sin(v2+c4)))-
                                        (c1/to)*(c2+c7*(v1*c6*cos(v2+c5)-
                                        zilch*wn*v1*sin(v2+c5)))+
                                        (c1/to)*(c2+c7*(c6*v3*cos(v4+c5)-
                                        zilch*wn*v3*sin(v4+c5)));
                }
                fprintf(out,"%10.6f %10.6f %10.6f \n",t,u,du);
        }
        return 0;
}



/* triangle2.c : solution of equations of motion with triangle input
   by runge-kutta iteration :
   12-01-09 : choice of iterations vs. printing by variable arg : ml
   A            =       affected area, in.^2
   arg          =       every 'argth' value printed
   delta        =       time step, seconds
   dun0         =       velocity of element, in.sec
   m            =       element mass, lbm
   Pt           =       magnitude of incident pressure,Pso+Pr, lbf/in.^2
   t            =       elapsed time, seconds
   to           =       duration of positive incident force, sec
   ttotal       =       total time of run, sec
   un0          =       displacement of element, in.
   wn           =       natural frequency of element, rad/sec =(k/m)^(1/2)
   zilch        =       fraction of critical damping
*/
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
        int arg,i,j,wrki;
        double gA,gdelta,gdun0,gm,gPt,gt,gto,gttotal,gun0,gwn,gzilch;
        double wrkf;
        void region1(double,double,double *,double,double,double,double,
                     double *,double,double);
        void region2(double,double,double *,double,double,
                     double,double,double *,double,double);
        double *input_data;
        FILE *inn;
```

```
        FILE *out;
        input_data=calloc(14,sizeof(double));
        inn=fopen("triangle2.in","r");
        out=fopen("triangle2.out","w+");
        for(i=0;i<=7;i++)
        {
                fscanf(inn,"%lf",&wrkf);
                *(input_data+i)=wrkf;
        }
        fprintf(out,"  TRIANGLE2.OUT\n\n");
        gA      =       *(input_data+0);
        gdelta  =       *(input_data+1);
        gm      =       *(input_data+2);
        gPt     =       *(input_data+3);
        gto     =       *(input_data+4);
        gttotal =       *(input_data+5);
        gwn     =       *(input_data+6);
        gzilch  =       *(input_data+7);
        fscanf(inn,"%d",&wrki);
        arg     =       wrki;
        fclose(inn);
        fprintf(out,"  A     = ");fprintf(out,"%14.6f",gA     );fprintf(out," in.^2\n");
        fprintf(out,"  delta = ");fprintf(out,"%14.6f",gdelta );fprintf(out," sec\n");
        fprintf(out,"  m     = ");fprintf(out,"%14.6f",gm     );fprintf(out," lbm\n");
        fprintf(out,"  Pt    = ");fprintf(out,"%14.6f",gPt    );fprintf(out," psi\n");
        fprintf(out,"  to    = ");fprintf(out,"%14.6f",gto    );fprintf(out," sec\n");
        fprintf(out,"  ttotal= ");fprintf(out,"%14.6f",gttotal);fprintf(out," sec\n");
        fprintf(out,"  wn    = ");fprintf(out,"%14.6f",gwn    );fprintf(out," rad/s\n");
         fprintf(out,"  zilch = ");fprintf(out,"%14.6f",gzilch );fprintf(out,"\n");
        fprintf(out,"  arg   = ");fprintf(out,"%14d"  ,arg    );fprintf(out,"\n\n");
        fprintf(out,"     sec        u          du    \n");
        fprintf(out,"  --------   --------   --------\n");
        gun0   = 0;
        gdun0  = 0;
        j      = 0;
        for(gt=0;gt<=gttotal+gdelta/2;gt+=gdelta)
        {
                if(gt<=gto)
                {
                        if(((j+arg)%arg)==0)
                        {
                                fprintf(out,"%10.6f %10.6f %10.6f\n",gt,
                                        gun0,gdun0);
                        }
                        else
                        {
                                ;
                        }
                        region1(gA,gdelta,&gdun0,gm,gPt,gt,gto,&gun0,gwn,gzilch);
                }
                else
                {
                        if(((j+arg)%arg)==0)
                        {
                                fprintf(out,"%10.6f %10.6f %10.6f\n",gt,gun0,gdun0);
                        }
                        else
                        {
                                ;
                        }
                        region2(gA,gdelta,&gdun0,gm,gPt,gt,gto,&gun0,gwn,gzilch);
                }
                j++;
        }
        fclose(out);
        return 0;
}
void region1(double A,double delta,double *dun0,double m,
             double Pt,double t,double to,double *un0,double wn,double zilch)
{
        double k1,k2,k3,k4;
```

```
        k1   = delta*(-wn*wn*(*un0)-2*zilch*wn*(*dun0)+
               (Pt*A/m)*(1-t/to));
        k2   = delta*(-wn*wn*(*un0+delta*(*dun0)/2+delta*k1/8)-2*zilch*wn*
               (*dun0+k1/2)+(Pt*A/m)*(1-(t+delta/2)/to));
        k3   = delta*(-wn*wn*(*un0+delta*(*dun0)/2+delta*k1/8)-2*zilch*wn*
               (*dun0+k2/2)+(Pt*A/m)*(1-(t+delta/2)/to));
        k4   = delta*(-wn*wn*(*un0+delta*(*dun0)+delta*k3/2)-2*zilch*wn*
               (*dun0+k3)+(Pt*A/m)*(1-(t+delta)/to));
        *un0 = *un0+delta*(*dun0+(k1+k2+k3)/6);
        *dun0= *dun0+(k1+2*(k2+k3)+k4)/6;
}
void region2(double A,double delta,double *dun0,
             double m,double Pt,double t,double to,
             double *un0,double wn,double zilch)
{
        double k1,k2,k3,k4;
        t    = t-to;
        k1   = delta*(-wn*wn*(*un0)-2*zilch*wn*(*dun0));
        k2   = delta*(-wn*wn*(*un0+delta*(*dun0)/2+delta*k1/8)-2*zilch*wn*
               (*dun0+k1/2));
        k3   = delta*(-wn*wn*(*un0+delta*(*dun0)/2+delta*k1/8)-2*zilch*wn*
               (*dun0+k2/2));
        k4   = delta*(-wn*wn*(*un0+delta*(*dun0)+delta*k3/2)-2*zilch*wn*
               (*dun0+k3));
        *un0 = *un0+delta*(*dun0+(k1+k2+k3)/6);
        *dun0= *dun0+(k1+2*(k2+k3)+k4)/6;
}



/* crossover1.c : calculation of crossover time from elastic to plastic
   regions : 12-13-09 : ml
*/
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
        int i,no_it;
        double A,b,m,Pt,strt,to,umax,wn,zilch;          /* external */
        double a,b1,c0,c1,c2,c3,c4,f0,fp0,t;            /* internal */
        FILE *inn;
        FILE *out;
        inn=fopen("crossover1.in","r");
        out=fopen("crossover1.out","w+");
        fscanf(inn,"%lf",&A);
        fscanf(inn,"%lf",&b);
        fscanf(inn,"%lf",&m);
        fscanf(inn,"%d ",&no_it);
        fscanf(inn,"%lf",&Pt);
        fscanf(inn,"%lf",&strt);
        fscanf(inn,"%lf",&to);
        fscanf(inn,"%lf",&umax);
        fscanf(inn,"%lf",&wn);
        fscanf(inn,"%lf",&zilch);
        fprintf(out,"crossover1.out\n\n");
        fprintf(out,"A     = ");fprintf(out,"%14.6f",A     );fprintf(out," in.^2\n");
        fprintf(out,"b     = ");fprintf(out,"%14.6f",b     );fprintf(out,"   \n");
        fprintf(out,"m     = ");fprintf(out,"%14.6f",m     );fprintf(out," lbm\n");
        fprintf(out,"no_it = ");fprintf(out,"%14d",no_it   );fprintf(out,"\n");
        fprintf(out,"Pt    = ");fprintf(out,"%14.6f",Pt    );fprintf(out," psi\n");
        fprintf(out,"strt  = ");fprintf(out,"%14.6f",strt  );fprintf(out," sec\n");
        fprintf(out,"to    = ");fprintf(out,"%14.6f",to    );fprintf(out," sec\n");
        fprintf(out,"umax  = ");fprintf(out,"%14.6f",umax  );fprintf(out," in.\n");
        fprintf(out,"wn    = ");fprintf(out,"%14.6f",wn    );fprintf(out," rad/sec\n");
        fprintf(out,"zilch = ");fprintf(out,"%14.6f",zilch );fprintf(out," \n\n");
        fprintf(out,"         t              f0             fp0   \n");
        fprintf(out,"     ---------      ---------       ---------\n");
        a       =       b/to-zilch*wn;
        b1      =       wn*sqrt(1-zilch*zilch);
```

```
        c0      =       Pt*A/(b1*m);
        c1      =       (to*a*(a*a+b1*b1)-(a*a-b1*b1))/(to*(a*a+b1*b1)*(a*a+b1*b1));
        c2      =       (to*(a*a+b1*b1)*(-b1)+2*a*b1)/(to*(a*a+b1*b1)*(a*a+b1*b1));
        c3      =       b1/(a*a+b1*b1);
        c4      =       -2*a*b1/(to*(a*a+b1*b1)*(a*a+b1*b1));
        t       =       strt;
        for(i=0;i<=no_it-1;i++)
        {
                f0 = c0*(exp(-zilch*wn*t)*(c1*sin(b1*t)+c2*cos(b1*t))+
                     exp(-b*t/to)*(c3*(1-t/to)+c4))-umax;
                fp0      =       c0*(exp(-zilch*wn*t)*
                                (b1*c1*cos(b1*t)-b1*c2*sin(b1*t)-
                                zilch*wn*(c1*sin(b1*t)+c2*cos(b1*t)))+
                                exp(-b*t/to)*(-c3/to-(b/to)*(c3*(1-t/to)+c4)));
                t = t-f0/fp0;
                fprintf(out,"%14.6f %14.6f %14.6f \n",t,f0,fp0);
        }
        return 0;
}



/* frdlndr1.c : solution of equations of motion with friedlander equation
   by transform method : elastic range = region1 and plastic range =
   region2 : 12-11-09 : set cro > ttotal for linear only : ml
   A            =       affected area, in.^2
   b            =       parameter containing rate of wave amplitude decay
   c            =       multiplier of du/dt to obtain viscous force,
                        lbf/(in./sec)
   cro          =       crossover time, seconds
   delta        =       time step, seconds
   dun0         =       velocity of element, in.sec
   k            =       multplier of u to obtain deflection force, lbf/in.
   m            =       element mass, lbm
   Pt           =       magnitude of total pressure, Pso+Pr, lbf/in.^2
   m            =       member mass, lbm
   Rm           =       maximum resisting force, lbf
   t            =       elapsed time, seconds
   to           =       duration of positive incident force, seconds
   ttotal       =       total time of run, seconds
   um           =       maximum linear displacement of element, in.
   un0          =       displacement of element, in.
   wn           =       natural frequency of element, rad/sec =(k/m)^(1/2)
   zilch        =       fraction of critical damping
*/
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
        int i;
        double gA,gb,gc,gcro,gdelta,gdun0,gk,gm,gPt,gRm,gt,gto,gttotal,gum,gun0,
               gwn,gzilch;
        double gu0,gdu0,wrkf;
        void region1(double,double,double *,double,double,double,double,
                     double *,double,double,double *,double *);
        void region2(double,double,double,double *,double,double,double,
                     double,double,double *,double *,double *,double);
        double *input_data;
        FILE *inn;
        FILE *out;
        input_data=calloc(14,sizeof(double));
        inn=fopen("frdlndr1.in","r");
        out=fopen("frdlndr1.out","w+");
        for(i=0;i<=13;i++)
        {
                fscanf(inn,"%lf",&wrkf);
                *(input_data+i)=wrkf;
        }
        fprintf(out,"NONLINEAR1.OUT\n\n");
```

```
        gA      =       *(input_data+0);
        gb      =       *(input_data+1);
        gc      =       *(input_data+2);
        gcro    =       *(input_data+3);
        gdelta  =       *(input_data+4);
        gk      =       *(input_data+5);
        gm      =       *(input_data+6);
        gPt     =       *(input_data+7);
        gRm     =       *(input_data+8);
        gto     =       *(input_data+9);
        gttotal =       *(input_data+10);
        gum     =       *(input_data+11);
        gwn     =       *(input_data+12);
        gzilch  =       *(input_data+13);
        fprintf(out,"A     = ");fprintf(out,"%14.6f",gA     );
        fprintf(out,"in.^2\n");
        fprintf(out,"b     = ");fprintf(out,"%14.6f",gb     );
        fprintf(out,"   \n");
        fprintf(out,"c     = ");fprintf(out,"%14.6f",gc     );fprintf(out,"lbf/ips\n");
        fprintf(out,"cro   = ");fprintf(out,"%14.6f",gcro   );fprintf(out,"seconds\n");
        fprintf(out,"delta = ");fprintf(out,"%14.6f",gdelta );fprintf(out,"sec\n");
        fprintf(out,"k     = ");fprintf(out,"%14.6f",gk     );fprintf(out,"lbf/in.\n");
        fprintf(out,"m     = ");fprintf(out,"%14.6f",gm     );fprintf(out,"lbm\n");
        fprintf(out,"Pt    = ");fprintf(out,"%14.6f",gPt    );fprintf(out,"lbf/in.^2\n");
        fprintf(out,"Rm    = ");fprintf(out,"%14.6f",gRm    );fprintf(out,"lbf/in.\n");
        fprintf(out,"to    = ");fprintf(out,"%14.6f",gto    );fprintf(out,"sec\n");
        fprintf(out,"ttotal= ");fprintf(out,"%14.6f",gttotal);fprintf(out,"sec\n");
        fprintf(out,"um    = ");fprintf(out,"%14.6f",gum    );fprintf(out,"in.\n");
        fprintf(out,"wn    = ");fprintf(out,"%14.6f",gwn    );fprintf(out,"rad/sec\n");
        fprintf(out,"zilch = ");fprintf(out,"%14.6f",gzilch );fprintf(out,"\n\n");
        fprintf(out," seconds      u         du    \n");
        fprintf(out,"--------- ---------  ---------\n");
        gun0   = 0;
        gdun0  = 0;
        region1(gA,gb,&gdun0,gm,gPt,gcro,gto,&gun0,gwn,gzilch,&gdu0,&gu0);
        gu0 = gun0;
        gdu0= gdun0;
        for(gt=0;gt<=gttotal+gdelta/2;gt+=gdelta)
        {
                if(gt<gcro)
                {
                        region1(gA,gb,&gdun0,gm,gPt,gt,gto,&gun0,gwn,gzilch,&gdu0,&gu0);
                        fprintf(out,"%7.3f %10.6f %10.6f\n",gt,gun0,gdun0);
                }
                else
                {
                        region2(gA,gb,gc,&gdun0,gm,gPt,gRm,gt,gto,&gun0,&gdu0,&gu0,gcro);
                        fprintf(out,"%7.3f %10.6f %10.6f\n",gt,gun0,gdun0);
                }
        }
        return 0;
}
void region1(double A,double b,double *dun0,double m,
             double Pt,double t,double to,double *un0,
             double wn,double zilch,double *du0,double *u0)
{
        double          a,b1,c0,c1,c2,c3,c4;
        a       =       b/to-zilch*wn;
        b1      =       wn*sqrt(1-zilch*zilch);
        c0      =       Pt*A/(b1*m);
        c1      =       (to*a*(a*a+b1*b1)-(a*a-b1*b1))/(to*(a*a+b1*b1)*(a*a+b1*b1));
        c2      =       (to*(a*a+b1*b1)*(-b1)+2*a*b1)/(to*(a*a+b1*b1)*(a*a+b1*b1));
        c3      =       b1/(a*a+b1*b1);
        c4      =       -2*a*b1/(to*(a*a+b1*b1)*(a*a+b1*b1));
        *un0    =       c0*(exp(-zilch*wn*t)*(c1*sin(b1*t)+c2*cos(b1*t))+
                        exp(-b*t/to)*(c3*(1-t/to)+c4));
        *dun0   =       c0*(exp(-zilch*wn*t)*
                        (b1*c1*cos(b1*t)-b1*c2*sin(b1*t)-
                        zilch*wn*(c1*sin(b1*t)+c2*cos(b1*t)))+
                        exp(-b*t/to)*(-c3/to-(b/to)*(c3*(1-t/to)+c4)));
}
```

```
void region2(double A,double b,double c,double *dun0,
             double m,double Pt,double Rm,double t,double to,
             double *un0,double *du0,double *u0,double cro)
{
        double          d0,d1,e1,e2,f1,f2,f3,f4;
        t       =       t-cro;
        d0      =       (Pt*A/m)*exp(-b*cro/to)*(1-cro/to);
        d1      =       (Pt*A/m)*exp(-b*cro/to)*(1/to);
        e1      =       exp(-(c/m)*t);
        e2      =       exp(-(b/to)*t);
        f1      =       1/((b/to)-(c/m));
        f2      =       (b/to)/((b/to)-(c/m));
        f3      =       1/((b/to)*((b/to)-(c/m)));
        f4      =       (c/m)/(b/to-c/m);

        *un0    =       *u0*e1+(*du0+(c/m)*(*u0))*(m/c)*(1-e1)-
                        (Rm/m)*(m/c)*(m/c)*((c/m)*t-1+e1)+
                        d0*(m*to/(b*c))*(1+f4*e2-f2*e1)-
                        d1*(1/((c/m)*(b/to)*(b/to))-
                        (m/c)*f1*f1*e1+
                        (2*b/to-c/m)*f3*f3*e2+
                        (to/b)*f3*t*e2);
        *dun0   =       -(c/m)*(*u0)*e1+(*du0+(c/m)*(*u0))*e1-
                        (Rm/c)*(1-e1)+
                        d0*(m*to/(b*c))*(-f4*b*e2/to+f2*c*e1/m)-
                        d1*(f1*f1*e1-(2*b/to-c/m)*f3*f3*b*e2/to+
                                  f3*(e2+t*(-b/to)*e2));
}
```

```
/* frdlndr2.c : solution of equations of motion with fridlander equation
   by runge-kutta iteration : elastic range and plastic range :
   choice of iterations vs. printing by variable arg :
   set cro > ttotal for linear only : ml : 12-11-09
   A            =       affected area, in.^2
   arg          =       every 'argth' value printed
   b            =       parameter containing rate of wave amplitude decay
   c            =       multiplier of du/dt to obtain viscous force, lbf/(in./sec)
   cro          =       crossover ctime,sec
   delta        =       time step, seconds
   dun0         =       velocity of element, in.sec
   k            =       multplier of u to obtain deflection force, lbf/in.
   m            =       element mass, lbm
   Pt           =       magnitude of total pressure, Pso+Pr, lbf/in.^2
   Rm           =       maximum resisting force, lbf
   t            =       elapsed time, seconds
   to           =       duration of positive incident force, sec
   ttotal       =       total time of run, sec
   um           =       maximum linear displacement of element, in.
   un0          =       displacement of element, in.
   wn           =       natural frequency of element, rad/sec =(k/m)^(1/2)
   zilch        =       fraction of critical damping
*/
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
        int arg,i,j,wrki;
        double gA,gb,gc,gcro,gdelta,gdun0,gk,gm,gPt,gRm,gt,gto,gttotal,gum,
               gun0,gwn,gzilch;
        double wrkf;
        void region1(double,double,double,double *,double,double,double,double,
                     double *,double,double);
        void region2(double,double,double,double,double *,double,double,double,
```

```
            double,double,double *,double);
double *input_data;
FILE *inn;
FILE *out;
input_data=calloc(14,sizeof(double));
inn=fopen("nonlinear2.in","r");
out=fopen("nonlinear2.out","w+");
for(i=0;i<=13;i++)
{
        fscanf(inn,"%lf",&wrkf);
        *(input_data+i)=wrkf;
}
fprintf(out,"NONLINEAR2.OUT\n\n");
gA      =       *(input_data+0);
gb      =       *(input_data+1);
gc      =       *(input_data+2);
gcro    =       *(input_data+3);
gdelta  =       *(input_data+4);
gk      =       *(input_data+5);
gm      =       *(input_data+6);
gPt     =       *(input_data+7);
gRm     =       *(input_data+8);
gto     =       *(input_data+9);
gttotal =       *(input_data+10);
gum     =       *(input_data+11);
gwn     =       *(input_data+12);
gzilch  =       *(input_data+13);
fscanf(inn,"%d",&wrki);
arg     =       wrki;
fprintf(out,"A     = ");fprintf(out,"%14.6f",gA     );fprintf(out,"in.^2\n");
fprintf(out,"b     = ");fprintf(out,"%14.6f",gb     );fprintf(out,"   \n");
fprintf(out,"c     = ");fprintf(out,"%14.6f",gc     );fprintf(out,"lbf/ips\n");
fprintf(out,"cro   = ");fprintf(out,"%14.6f",gcro   );fprintf(out,"sec\n");
fprintf(out,"delta = ");fprintf(out,"%14.6f",gdelta );fprintf(out,"sec\n");
fprintf(out,"k     = ");fprintf(out,"%14.6f",gk     );fprintf(out,"lbf/in.\n");
fprintf(out,"m     = ");fprintf(out,"%14.6f",gm     );fprintf(out,"lbm\n");
fprintf(out,"Pt    = ");fprintf(out,"%14.6f",gPt    );fprintf(out,"lbf/in.^2\n");
fprintf(out,"Rm    = ");fprintf(out,"%14.6f",gRm    );fprintf(out,"lbf/in.\n");
fprintf(out,"to    = ");fprintf(out,"%14.6f",gto    );fprintf(out,"sec\n");
fprintf(out,"ttotal= ");fprintf(out,"%14.6f",gttotal);fprintf(out,"sec\n");
fprintf(out,"um    = ");fprintf(out,"%14.6f",gum    );fprintf(out,"in.\n");
fprintf(out,"wn    = ");fprintf(out,"%14.6f",gwn    );fprintf(out,"rad/sec\n");
fprintf(out,"zilch = ");fprintf(out,"%14.6f",gzilch );fprintf(out,"\n");
fprintf(out,"arg   = ");fprintf(out,"%14d"  ,arg    );fprintf(out,"\n\n");
fprintf(out," seconds      u         du    \n");
fprintf(out,"--------- ---------  ---------\n");
gun0   = 0;
gdun0  = 0;
j      = 0;
for(gt=0;gt<=gttotal+gdelta/2;gt+=gdelta)
{
        if(gt<gcro)
        {
                if(((j+arg)%arg)==0)
                {
                        fprintf(out,"%7.3f %10.6f %10.6f\n",gt,gun0,gdun0);

                }
                else
                {
                        ;
                }
                region1(gA,gb,gdelta,&gdun0,gm,gPt,gt,gto,&gun0,gwn,gzilch);
        }
        else
        {
                if(((j+arg)%arg)==0)
                {
                        fprintf(out,"%7.3f %10.6f %10.6f\n",gt,
                                gun0,gdun0);
                }
```

```
                        else
                        {
                                ;
                        }
                        region2(gA,gb,gc,gdelta,&gdun0,gm,gPt,gRm,gt,
                                gto,&gun0,gcro);
                }
                j++;
        }
        return 0;
}
void region1(double A,double b,double delta,double *dun0,double m,
             double Pt,double t,double to,double *un0,double wn,double zilch)
{
        double k1,k2,k3,k4;
        k1   = delta*(-wn*wn*(*un0)-2*zilch*wn*(*dun0)+
               (Pt*A/m)*(1-t/to)*exp(-b*t/to));
        k2   = delta*(-wn*wn*(*un0+delta*(*dun0)/2+delta*k1/8)-2*zilch*wn*
               (*dun0+k1/2)+(Pt*A/m)*(1-(t+delta/2)/to)*
               exp((-b/to)*(t+delta/2)));
        k3   = delta*(-wn*wn*(*un0+delta*(*dun0)/2+delta*k1/8)-2*zilch*wn*
               (*dun0+k2/2)+(Pt*A/m)*(1-(t+delta/2)/to)*
               exp((-b/to)*(t+delta/2)));
        k4   = delta*(-wn*wn*(*un0+delta*(*dun0)+delta*k3/2)-2*zilch*wn*
               (*dun0+k3)+(Pt*A/m)*(1-(t+delta)/to)*exp(-(b/to)*(t+delta)));
        *un0 = *un0+delta*(*dun0+(k1+k2+k3)/6);
        *dun0= *dun0+(k1+2*(k2+k3)+k4)/6;
}

void region2(double A,double b,double c,double delta,double *dun0,
             double m,double Pt,double Rm,double t,double to,
             double *un0,double cro)
{
        double d0,d1,e1,e2,k1,k2,k3,k4;
        t    = t-cro;
        d0   = (Pt*A/m)*exp(-b*cro/to)*(1-cro/to);
        d1   = (Pt*A/m)*exp(-b*cro/to)*(1/to);

        k1   = delta*(-(c/m)*(*dun0)-Rm/m+d0*exp(-b*t/to)-
               d1*t*exp(-b*t/to));
        k2   = delta*(-(c/m)*(*dun0+k1/2)-Rm/m+d0*exp(-b*(t+delta/2)/to)-
               d1*(t+delta/2)*exp(-b*(t+delta/2)/to));
        k3   = delta*(-(c/m)*(*dun0+k2/2)-Rm/m+d0*exp(-b*(t+delta/2)/to)-
               d1*(t+delta/2)*exp(-b*(t+delta/2)/to));
        k4   = delta*(-(c/m)*(*dun0+k3)-Rm/m+d0*exp(-b*(t+delta)/to)-
               d1*(t+delta)*exp(-b*(t+delta)/to));

        *un0 = *un0+delta*(*dun0+(k1+k2+k3)/6);
        *dun0= *dun0+(k1+2*(k2+k3)+k4)/6;
}
```

7. TYPICAL OUTPUTS

DIRAC.OUT

```
  A     =   28800.000000 in.^2          delta =       0.050000 sec
  m     =    4500.000000 lbm            Pt    =     348.452000 lbf/in.^2
  to    =       0.005491 sec            ttotal=       1.000000 sec
  wn    =       2.683599 rad/sec        zilch =       0.100000
  W     =     100.000000 lbf            Z     =       4.140000 ft/lbr^(1/3)

                   sec         u          du
                --------    --------   --------
                0.000000    0.000000   6.122720
                0.050000    0.301159   5.906536
                0.100000    0.589002   5.591305
                0.150000    0.858775   5.185241
                0.200000    1.106172   4.697953
```

```
   0.250000   1.327396   4.140235
   0.300000   1.519217   3.523844
   0.350000   1.679010   2.861268
   0.400000   1.804790   2.165483
   0.450000   1.895225   1.449710
   0.500000   1.949648   0.727175
   0.550000   1.968047   0.010873
   0.600000   1.951049  -0.686655
   0.650000   1.899893  -1.353533
   0.700000   1.816391  -1.978743
   0.750000   1.702880  -2.552299
   0.800000   1.562169  -3.065395
   0.850000   1.397474  -3.510530
   0.900000   1.212351  -3.881607
   0.950000   1.010625  -4.174006
   1.000000   0.796315  -4.384624

      tmax =    0.550767 sec
```

TRIANGLE1.OUT
TRANSFORM METHOD

```
A     =   28800.000000 in.^2          delta =       0.050000 sec
m     =    4500.000000 lbm            Pt    =     114.564000 lbf
to    =       0.016733 sec            ttotal=       1.000000 sec
wn    =       2.683599 rad/sec        zilch =       0.100000
W     =    1000.000000 lbf            Z     =       6.090000 ft/lbf^(1/3)
```

sec	u	du
0.000000	-0.000001	-0.000000
0.050000	0.268604	5.946706
0.100000	0.558713	5.641588
0.150000	0.831229	5.244479
0.200000	1.081785	4.764857
0.250000	1.306517	4.213415
0.300000	1.502124	3.601836
0.350000	1.665906	2.942561
0.400000	1.795801	2.248548
0.450000	1.890402	1.533026
0.500000	1.948966	0.809257
0.550000	1.971408	0.090299
0.600000	1.958287	-0.611224
0.650000	1.910777	-1.283329
0.700000	1.830631	-1.914873
0.750000	1.720135	-2.495729
0.800000	1.582053	-3.016934
0.850000	1.419564	-3.470823
0.900000	1.236197	-3.851127
0.950000	1.035758	-4.153050
1.000000	0.822254	-4.373310

TRIANGLE2.OUT
ITERATIVE METHOD

```
A     =   28800.000000 in.^2            delta =       0.000010 sec
m     =    4500.000000 lbm              Pt    =     114.564000 lbf/in.^2
to    =       0.016733 sec              ttotal=       1.000000 sec
wn    =       2.683599 rad/sec          zilch =       0.100000
W     =    1000.000000 lbf              Z     =       6.09 ft/lbf^(1/3)
arg   =         5000
```

sec	u	du
0.000000	0.000000	0.000000
0.050000	0.268604	5.946705
0.100000	0.558713	5.641587

```
0.150000   0.831229   5.244478
0.200000   1.081785   4.764856
0.250000   1.306517   4.213414
0.300000   1.502123   3.601835
0.350000   1.665905   2.942560
0.400000   1.795800   2.248547
0.450000   1.890402   1.533026
0.500000   1.948966   0.809257
0.550000   1.971408   0.090299
0.600000   1.958286  -0.611224
0.650000   1.910776  -1.283329
0.700000   1.830630  -1.914873
0.750000   1.720134  -2.495728
0.800000   1.582052  -3.016933
0.850000   1.419564  -3.470822
0.900000   1.236197  -3.851127
0.950000   1.035758  -4.153049
1.000000   0.822254  -4.373309
```

crossover1.out

```
A     =   28800.000000 in.^2
b     =       2.000000
m     =    4500.000000 lbm
no_it =              10
Pt    =     201.000000 psi
strt  =       0.300000 sec
to    =       0.030162 sec
umax  =       1.971755 in.
wn    =       2.683599 rad/sec
zilch =       0.100000
```

t	f0	fp0
0.221383	0.439507	5.590506
0.229891	-0.060307	7.088084
0.229984	-0.000640	6.937082
0.229984	-0.000000	6.935428
0.229984	-0.000000	6.935427
0.229984	0.000000	6.935427
0.229984	-0.000000	6.935427
0.229984	0.000000	6.935427
0.229984	0.000000	6.935427
0.229984	-0.000000	6.935427

FRDLNDR1.OUT
ELASTIC-PLASTIC

```
A     =   28800.000000 in.^2          b     =       2.000000
c     =    2415.239000 lbf/ips        cro   =       0.229984 sec
delta =       0.050000 sec            k     =   32407.674000 lbf/in.
m     =    4500.000000 lbm            Pt    =     201.000000 lbf/in.^2
Rm    =   63900.000000 lbf            to    =       0.030162 sec
ttotal=       1.000000 sec            um    =       1.971755 in.
wn    =       2.683599 rad/sec        zilch =       0.100000
```

seconds	u	du
0.000	0.000000	-0.000000
0.050	0.460845	10.195211

```
 0.100   0.934053   8.944595
 0.150   1.363328   8.231620
 0.200   1.755933   7.454185
 0.250   2.106998   6.578586 <-- crossover
 0.300   2.413961   5.703824
 0.350   2.677767   4.852234
 0.400   2.899560   4.023193
 0.450   3.080452   3.216105
 0.500   3.221526   2.430387
 0.550   3.323837   1.665475
 0.600   3.388411   0.920817
 0.650   3.416248   0.195876
 0.700   3.408319  -0.509868
 0.750   3.365572  -1.196925
 0.800   3.288930  -1.865789
 0.850   3.179289  -2.516943
 0.900   3.037523  -3.150854
 0.950   2.864483  -3.767980
 1.000   2.660997  -4.368766
```

```
                FRDLNDR2.OUT
               ELASTIC-PLASTIC

A     =   28800.000000 in.^2          b     =       2.000000
c     =    2415.239000 lbf/ips        cro   =       0.229984 sec
delta =       0.000010 sec            k     =   32407.674000 lbf/in.
m     =    4500.000000 lbm            Pt    =     201.000000 lbf/in.^2
Rm    =   63900.000000 lbf            to    =       0.030162 sec
ttotal=       1.000000 sec            um    =       1.971755 in.
wn    =       2.683599 rad/sec        zilch =       0.100000
arg   =           5000

          seconds      u         du
          --------- ---------  ---------
            0.000   0.000000   0.000000
            0.050   0.460845  10.195211
            0.100   0.934053   8.944595
            0.150   1.363328   8.231620
            0.200   1.755933   7.454185
            0.250   2.106998   6.578586 <-- crossover
            0.300   2.413961   5.703824
            0.350   2.677767   4.852234
            0.400   2.899560   4.023193
            0.450   3.080452   3.216105
            0.500   3.221526   2.430387
            0.550   3.323837   1.665475
            0.600   3.388411   0.920817
            0.650   3.416248   0.195876
            0.700   3.408319  -0.509868
            0.750   3.365572  -1.196925
            0.800   3.288930  -1.865789
            0.850   3.179289  -2.516943
            0.900   3.037523  -3.150854
            0.950   2.864483  -3.767980
            1.000   2.660997  -4.368766
```

TABLE 7-1

BASIC PROPERTIES OF COMMON SINGLE-COMPOUNDHIGH EXPLOSIVES

Explosive	Equivalent Weight Relative to TNT*	Density (g/cm^3)	Detonation Velocity (km/s)	Consistency
Ammonium Nitrate, AN	—	1.73	7.00	Solid
Nitroglycerin, NG	—	1.60	7.58	Liquid
Trinitrotoluene, TNT	1.00	1 6 5	6 90	Solid
Pentaerythritol Tetranitrate, PETN	1.27	1.70	7.98	Solid
Cyclotrimethylene trinitramine, RDX	1.19	1.80	8.75	Solid
Cyclotetramethylene tetranitramine, HMX	−1.30	1.90	9.10	Solid
Nitrocellulose, NC	—	1.2–1.7	7.30	Solid

*Based on the peak pressure produced compared to TNT

TABLE 7-2

BASIC PROPERTIES OF COMMMON MIXED-COMPOUND EXPLOSIVES

Explosive*	Equivalent Weight Relative to TNT**	Density g/cm^3	Detonation Velocity km/s	Consistency
ANFO 94 AN/6 FG	0.82	−0.8	−4.7	Powder
Composition B 40 TNT/60 RDX	1.11	1.70	7.9	Solid
Octol 25 TNT/75 HMX	1.06	1.82	8.4	Solid
Pentolite 50 TNT/50 PETN	1.42	1.67	7.4	Solid
Dynamite 50 NG/0.2 NC/34 SN/15.8 C	0.90	1.40	−5.8	Solid
Bonded Mixtures				
Composition C4 91 RDX/ 2.1 rubber/ 1.6 oil/ 5.3 plasticizer	1.37	1.0–1.60	8.0	Plastic
Sheet Explosive 60–85 PETN/0–8 NC rubber and plasticizer	−1.27	−1.50	−7.0	Rubberlike sheets

*FO: Fuel oil; SN: Sodium Nitrate; C: Combustibles and chalk
**Equivalentto weights based on peak overpressure produced compared to TNT

Figure 2-15 Positive Phase Shock Wave Parameters for a Hemispherical TNT Explosion on the Surface at Sea Level