UNCLASSIFIED

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUNE 1990	Final, Jun 85 - May 90

4. TITLE AND SUBTITLE

Fortran Programs for Weapon Systems Analysis

5. FUNDING NUMBERS

1L162618AH80
DA30 6667

6. AUTHOR(S)

Bunn, Fred L.; Olah, Joseph M.; Ritondo, Michele

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Ballistic Research Laboratory
ATTN: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-5066

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

BRL-TR-3116

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report documents small Fortran 77 routines which are useful to those interested in ballistics and related work. The programs include skeletal combat models, a set of discrete-event timing routines, mathematical and statistical routines, hit probability routines, acquisition routines, ballistic routines, and others.

All of the programs, subroutines, and functions are in the public domain and may be copied freely. Government agencies and contractors may request copies from the Ballistic Research Laboratory.

14. SUBJECT TERMS

ballistics, Fortran, weapon, combat effectiveness

15. NUMBER OF PAGES

89

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

TABLE OF CONTENTS

TABLE OF CONTENTS (contd)

ACRONYMS

APFSDS	Armor-Piercing Fin-Stabilized Discarding-Sabot
DARCOM	Development and Readiness Command
HD	Hull Defilade (only tank turret is exposed)
KE	Kinetic Energy (projectile)
MKS	Meters/Kilograms/Seconds
TRAC	TRADOC Analysis Center
TRADOC	Training and Doctrine Command
WSMR	White Sands Missile Range, NM

INTENTIONALLY LEFT BLANK.

1. Preliminaries

1.1 Introduction

This report contains generally useful Fortran 77 programs, subroutines, and functions. They are short routines to aid those interested in ballistics and related work. The routines in the report are public domain. You may either copy them or obtain them from the authors. They are standard Fortran 77 with few exceptions.

Getting and Using the Software. How can you obtain copies? If you have access to the Smoke computer at BRL, simply copy them from the directory /secad/fred/Tools/Rpt. If not, and you work for a government agency, call the authors at AV 298-6676 (or 6653) or email your request to fred@brl.mil or joeo@brl.mil. The programs are available on IBM compatible 5-1/4 floppy discs. If you work for a government contractor, send a letter requesting the programs to:

Director
Ballistic Research Laboratory
SLCBR-SE-B (attn: Fred Bunn)
Aberdeen Proving Ground, MD 21005

Fortran Conventions. The algorithms are written in standard Fortran 77, with few exceptions. What's non-standard about the codes? They use mixed upper and lower case, the include statement, and tabs.

We insist on compilers that allow lower case. Lower case is far more readable and listings are more compact. If you want single case, use the tr command in Unix, or some similar command on your system, or we can translate to the appropriate case for you.

Some of the programs use the include statement. This is not standard Fortran 77, although many Fortran 77 compilers accept some form of the include statement and the Fortran 8X standard will have this feature. If this causes a problem, you have these options: use the Unix m4 macro processor, edit the include statements to conform to your compiler, manually replace the include statements with the file to be included, or let us apply the m4 macro processor for you.

We use the tab character to move to column 7. Many compilers accept this. If yours does not, run the program through the Unix expand program or request the programs without tabs.

To compile the programs on the Unix operating system, we suggest you use a command of the form: f77 name1.f name2.f ... -o name. Other operating systems will use some similar command.

The algorithms were written using the following ten commandments, which are generally accepted as good programming practice. They are so important we include them here.

1. Construct new code by successive refinements. (Top down, structured programming).
2. Never use assigned GOTO's or arithmetic IF's.
3. Never use alternate returns.
4. Avoid GOTO's as much as possible.
5. Make loops and branches with only 1 entry and 1 exit.
6. Keep routines to a page or less.
7. Indent the code neatly and logically.
8. Use upper case to highlight changes in the logical flow*.
9. Choose mnemonic variable names.

---

* These include FUNCTION, SUBROUTINE, END, IF-THEN, ELSE, ELSEIF, ENDIF, DO, CONTINUE, GOTO, and STOP. Calls to subroutines and functions are not included since these are a lower level of abstraction and should not be thought of as changes in the flow of control by the reader of the calling routine.

10. Make 10%-20% of the lines meaningful comments.

Other Needed Software. Some of the programs need subroutines from *Numerical Recipes*[1]. We cannot include them because of copyright. However, the book is excellent and should be on every weapon analyst's bookshelf. The fnd2d, polygon, and solid routines call the qsimp and trap routines found in *Numerical Recipes* and the drag routine calls the hunt routine found there.

Coordinates and Units. When the algorithms use geographical coordinates, they are the standard coordinates used at test ranges and in navigation systems. That is,

1. X is positive Eastward,
2. Y is positive Northward,
3. Z is positive upward,
4. θ (azimuth) is positive clockwise from North,
5. ϕ (elevation) is positive upward from the XY plane.

Figure 1.1 below shows the standard right-handed coordinate system this forms.

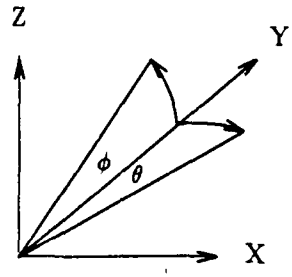


Figure 1.1 The Coordinate System

Inside the algorithms, units are meters, seconds, and radians unless otherwise stated. Frequently, the algorithms read input in other standard units. The most common is the military mil, which divides the circle in 6,400 parts.

Corrections. These routines have proved their usefulness over the years, however we make no claim to infallibility. If you find errors, have suggestions for improvements, or have difficulty using the routines, please contact the authors at AV 298-6676 or email to fred@brl.mil or joeo@brl.mil.

1. *Numerical Recipes*, Press, W., Flannery, B., Teukolsky, S., Vetterling, W., Cambridge University Press, NY.

1.2 On Not Re-inventing Software Wheels

We can only avoid the plague of software re-invention by banding together in sub-fields and developing catalogs of small, widely usable algorithms. These efforts will proceed only if managers realize the value of producing and disseminating such documentation and institutionalize rewards for this valuable work.

Software algorithms are constantly being re-invented and the waste caused by this is a significant factor in what has been termed "the software crisis." Richard Hamming observed in his 1966 Turing Lecture that

> *"While Newton could attribute his success to having stood on the shoulders of giants, in computer science we tend to stand mostly on each other's feet."*

This is as true in the development of scientific and engineering software for military applications as it is of computer science.

This crisis occurs in all fields, including the modeling of logistics, mobility, ballistics, and combat simulation. A partial solution to this 'crisis' is the construction and use of specialized Small Parts Catalogs for each field. These Small Parts Catalogs would reduce the constant re-invention of software 'wheels' much as a designer of machines uses a small parts catalog to select standard bolts, gears, pins, etc.

If a designer of machinery were to design each nut and bolt of a new machine he would not have a job for very long. Either he would be fired or his company would quickly go out of business because the product would be far too expensive to meet the competition. Yet that's exactly how many of us proceed when we develop software to aid us. The problem is that we have never created catalogs of standard parts.

We Constantly Re-Invent Software. Many is the time a model developer finds he needs an algorithm and says to himself, "This must have been coded before - and more than once." The chances are he's right. He then has three choices: search for the algorithm in books and among friends, cannibalize it from an existing large program, or write it anew.

So what does he do? He might go searching for the algorithm he needs. If he searches the literature, he'll probably come up empty handed. If he asks around he either draws a blank or somebody hands him a hundred pages of code and says, "It's in there."

If he actually wades through the code, or wonder of wonders, documentation of the code exists, and he wades through the documentation and eventually finds what he wants, he is cannibalizing. Cannibalizing isn't a bad idea in an emergency. In battle, tankers do it all the time when the parts aren't otherwise available, but only as a last resort.

One experience like this occurred about four years ago. We developed a quick and dirty algorithm to do a job, extended it, checked out the results with standard tables, and thought about writing it up. Belatedly, it occurred to us that we have a branch whose mission in life for fifty years or more has been to produce this kind of work - why not look over their code first. They were more than willing to give me a copy of their code. It was production code that was used constantly.

Now, admittedly, their code was far more powerful than our two page algorithm because they had to solve the problem for far more exotic conditions, but there was no way we could have ferreted out the portion we needed. The listing was 99 pages long, the main program was 37 pages long, comments were minimal, there was no indentation, 'goto' statements were rampant, variables were nowhere defined, non-standard Fortran was used, and some of the code appeared to use Fortran II constructs. The listing is now sitting on a shelf unused.

A Partial Solution. Those of us in the subfield of say, combat modeling, have developed small algorithms useful to many. We have explored new terrain - but have not blazed the trail well, either for ourselves or for those who follow after. We can only avoid this by combining efforts and blazing the trail with something like Small Parts Catalogs containing the algorithms most generally useful to our sub-field.

Such a catalog for combat modeling, for example, would contain many small stand-alone programs modeling facets of combat. For each program, the problem would be stated and the input and output described. The mathematical solution would be given, as would the source code, and test cases. Each program would contain a main routine to interface with the user and a subprogram to solve the algorithm. The subprogram could be removed and used as a part in a larger program or the program itself could be used for debugging purposes or to solve small problems.

This technique has been used in other subfields, but to my knowledge not in ours. Examples include:

The IBM Scientific Subroutine Package
BMDP Biomedical Computer Programs
Operations Research for Immediate Application, by Woolsey & Swanson
Software Tools, by Kernighan & Plauger
Applied Numerical Methods, by Carnahan, et al.
Numerical Recipes, by Press et al.

Benefits of Catalogs. This approach to the 'software crisis' has numerous benefits. First, the software builder can reduce his labor by quickly selecting ready made parts. Second, if he documents his large program, he need not document these standard parts. So the documentation task is both less formidable and more likely to get done. Third, the parts are available for wide scrutiny. In time this filters out bugs and bad algorithms, leaving algorithms that are widely available, documented, fast, precise, and as error free as possible. Fourth, the stand alone program may solve his problem directly, or he may want to use it for debugging purposes.

Developing Small Parts Catalogs. At the grass roots level, program developers must contact others in the same subfield and begin to form ad hoc groups. These groups must settle on the contents of their Small Parts Catalog (SPC), collect algorithms, cull, document, publish, advertise and distribute them.

On a higher level, management should be aware that millions of dollars are being wasted on the constant re-invention of what should be standard software algorithms. It is up to management to establish offices for this purpose, encourage the formation of ad hoc groups to develop SPC's, and develop methods to reward contributors and compilers of SPC's.

FORTRAN 77 PROGRAMS, SUBROUTINES, AND FUNCTIONS

2. Combat Simulation

2.1 Duel4: Simulate Combat Between Two Tanks.

Duel4 finds the probability that a Blue or Red tank (or neither) wins a duel. It is a deterministic, discrete event simulation, useful for examining important tank parameters and as a framework for building more detailed models of one on one combat.

Input. The input file contains 8 lines of data. All data are in the MKS metric system, unless otherwise stated. Lines 1-3 are data for the Blue system, lines 4-6 are for the Red system, and lines 7,8 are simulation control values.

Line 1 contains 5 firing cycle parameters for the Blue combatant.

1. Time between rounds in a burst (sec).
2. Time between bursts (sec). This is the time between the last round in a burst and the first round in the next burst. If you are simulating a weapon that doesn't fire bursts, just set this value to zero or one.
3. # rounds per burst.
4. # rounds on board
5. Time first round is fired (sec).

Line 2 contains 2 ballistic parameters for the rounds for the Blue combatant.

1. Muzzle velocity (m/s).
2. Drag (m/s per meter) (must be negative #).

Line 3 contains 3 lethality parameters for the Blue combatant.

1. Dispersion (mils). (assumes circular shot pattern w/ no bias).
2. System radius (m). The target is assumed to be circular; this is the radius of the Blue system. It used to find the hit probability of Red rounds.
3. Probability of kill given a hit.

Lines 4-6 contain similar data for the Red combatant.

Line 7 contains 4 range parameters.

1. Minimum range between combatants (m).
2. Maximum range between combatants (m). Section 2.6 discusses this.
3. Range increment (m).
4. Parameter for range between combatants (m).

Line 8 contains 2 game control parameters.

1. Print level (0=minimal print, 1 adds results of each shot, 2 adds time of fire).
2. Maximum time for a duel (sec).

Sample run at multiple ranges. The following shows the input and output for a duel at ranges from 200 meters to 2400 meters.

Table 2.1a: Sample Input File

DATA FILE	DESCRIPTION
1. 3. 0 8 1.5	time between rds, time between bursts, #rds/burst, #rds, tstart
1000. -.1	muzzle velocity, drag
0.5 1.0 0.5	dispersion (mils), system radius (m), pk|h
1. 5. 2 8 3.	time between rds, time between bursts, #rds/burst, #rds, tstart
0.5 1.0 0.5	dispersion (mils), system radius (m), pk|h
200 2400 200 700	min rg, max rg, inc rg, mean rg
0 20.	print level

Table 2.1b: Sample Output for Multiple Ranges

Range (meters)	P Eng at Range	State Probabilities — Both Alive	Blue Wins	Red Wins	Both Dead	Exchange Ratio
200	0.21	0.00	0.60	0.40	0.00	1.50
400	0.21	0.00	0.60	0.40	0.00	1.50
600	0.18	0.00	0.60	0.33	0.07	1.66
800	0.13	0.00	0.59	0.34	0.07	1.61
1000	0.09	0.00	0.57	0.36	0.07	1.50
1200	0.06	0.00	0.55	0.38	0.07	1.38
1400	0.04	0.01	0.40	0.31	0.28	1.15
1600	0.03	0.02	0.41	0.34	0.23	1.13
1800	0.02	0.04	0.42	0.35	0.19	1.11
2000	0.01	0.07	0.41	0.36	0.16	1.10
2200	0.01	0.10	0.41	0.36	0.13	1.09
2400	0.01	0.14	0.36	0.36	0.14	1.00

Mean exchange ratio over all ranges is 1.491

The second column shows the probability that an engagement will occur at the range shown in column 1. This is explained further in the description of the tgtrg routine.

Sample of detailed run at 1000 meters range. Now, we change the last 2 lines of the input data so that combat is simulated at only 1000 meters range and so that each event is printed. The last two input lines now read as follows:

```
1000 1000 200 700     min rg, max rg, inc rg, mean rg
2 20                  print level
```

Table 2.1c: Sample Output at One Range

	Range (meters)	P Eng at Range	State Probabilities — Both Alive	Blue Wins	Red Wins	Both Dead	Exchange Ratio
0.00	At start time		1.00	0.00	0.00	0.00	
1.50	Blue fires at Red						
2.55	Blue rd 1 hits.		0.56	0.44	0.00	0.00	
3.00	Red fires at Blue						
4.00	Red fires at Blue						
4.05	Red rd 1 hits.		0.32	0.44	0.25	0.00	1.78
4.50	Blue fires at Red						
5.05	Red rd 2 hits.		0.18	0.44	0.38	0.00	1.14
5.55	Blue rd 2 hits.		0.10	0.52	0.32	0.06	1.50
7.50	Blue fires at Red						
8.55	Blue rd 3 hits.		0.06	0.56	0.32	0.06	1.61
9.00	Red fires at Blue						
10.00	Red fires at Blue						
10.05	Red rd 3 hits.		0.03	0.56	0.35	0.06	1.51
10.50	Blue fires at Red						
11.05	Red rd 4 hits.		0.02	0.56	0.36	0.06	1.46
11.55	Blue rd 4 hits.		0.01	0.57	0.36	0.07	1.50
13.50	Blue fires at Red						
14.55	Blue rd 5 hits.		0.01	0.57	0.36	0.07	1.51
15.00	Red fires at Blue						
16.00	Red fires at Blue						
16.05	Red rd 5 hits.		0.00	0.57	0.36	0.07	1.50
16.50	Blue fires at Red						
17.05	Red rd 6 hits.		0.00	0.57	0.36	0.07	1.49
17.55	Blue rd 6 hits.		0.00	0.57	0.36	0.07	1.50
19.50	Blue fires at Red						
	1000	1.00	0.00	0.57	0.36	0.07	1.50

Mean exchange ratio over all ranges is 1.497

2.2 Duel4 Continued.

This section discusses the mathematics of Duel4 and lists the Fortran code.

Mathematics. The time of flight of a KE round may be approximated by:

$$t = \log((dr+v)/v)) \; / \; d$$

Where:

d = drag (m/s per km)(d<0).
r = range between combatants (km).
v = muzzle velocity (m/s).

Duel4 treats the combatants as circular targets for the purpose of finding hit and kill probabilities. The probability of kill given a shot is:

$$p_k = pe^{-(r/\sigma)^2}$$

Where:

p = the probability of kill given a hit,
r = the radius of the target, and
σ = the linear dispersion of the round.

Hierarchy. Here is the hierarchy tree showing which routines call which. It is followed by a list of the three event handling routines the program uses.

DUEL4 - simulate duels at various ranges.
- EVENTS - simulate all events in a single duel.
- FIRE - simulate firing a round.
- IMPACT - simulate impact of round.
- MARKOV - find probability of each outcome. (Section 2.7)
- TGTRG - finds probability combat occurs at range r. (Section 2.6)

Utility routines:
RESET - initialize the event queue. (Section 3.2)
SKEDUL - schedule an event. (Section 3.3)
EVENT - find the next event. (Section 3.4)

Data Structure. The following values are in the duel4.h file to be used by the main program as well as by the fire and impact subroutines. These values do not change during a run:

- name(2) - the color of the two sides.
- lev - the printing level
- nrds(2) - the number of rounds available at the beginning of combat.
- dt1(2) - the time between rounds (sec).
- dt2(2) - the time between bursts (sec).
- nrb(2) - the number of rounds in a burst.

These values do not change during an engagement at a single range:

- tof(2) - the time of flight (sec).
- pk(2) - the probability of kill given a shot.

These values change during the engagement:

- nrd(2) - the number of rounds remaining.
- palive(2,50) - the probability the ith tank is alive after firing j rounds.
- ex - the exchange ratio (probability Red is dead / probability Blue is dead).
- p(4) - the probability of each of the 4 final states.

Code.

```
c       duel4.h include file:
        character name*4
        common /charc / name(2)
        common /allc  / lev, nrds(2), nrd(2), palive(2,50)
        common /firec / dt1(2), dt2(2), nrb(2), tof(2)
        common /impctc/ ex, pk(2), p(4)

        PROGRAM DUEL4
c       Duel4: Simulate a duel at various ranges.
        include 'duel4.h'
        real tstart(2), vm(2), drag(2), radius(2), pkh(2), sigma(2)
        integer r
c       fr - frequency of duel occurring at range r.
c       exmean - mean exchange ratio over all engagement ranges.
c       ph - the hit probability
c       minr - minimum range between combatants (m).
c       maxr - maximum range between combatants (m).
c       incr - range increment (m).
c       meanr - the parameter defining the combat range distribution (m).
c       r - the range between combatants for the current engagement (m).
c       tstop - the time the engagement stops (sec).
        data name /'Blue','Red '/
2       format (//11x,
     *  'Range   P Eng     ----- State Probabilities -----    Exchange'/
     *  10x,
     *  '(meters)  at     Both    Blue     Red    Both     Ratio'/
     *  15x, '    Range    Alive   Wins     Wins    Dead')
3       format(10x, i6, 6f9.2)
4       format(10x, 'Mean exchange ratio over all ranges is',f8.3)
5       format(a4,' tof=',f6.2,' ph=',f6.2,' pk=',f6.2)
6       format(f11.2,'  At start time  ', f5.2,4f9.2)

c       Initialize.
        print *,' DUEL4: Written by Fred Bunn, July 1989'
        print *
        DO 20 i=1,2
          read *, dt1(i), dt2(i), nrb(i), nrds(i), tstart(i)
          read *, vm(i), drag(i)
          read *, sig, radius(i), pkh(i)
          sigma(i) = 0.9817e-3*sig
20      CONTINUE
        read *, minr, maxr, incr, meanr
        read *, lev, tstop
        exmean = 0.0
        DO 40 r=minr,maxr,incr
c       Simulate a duel at range r.
          p(1) = 1.0
          p(2) = 0.0
          p(3) = 0.0
          p(4) = 0.0
          call reset(.false.)
          DO 30 I=1,2
            nrd(I) = nrds(I)
            temp=(drag(I)*r+vm(I))/vm(I)
            if (temp.ne.0.0) tof(I) = alog(temp)/drag(I)
            palive(I,nrd(I)) = 1.0
            temp = -0.5*(radius(I)/(sigma(I)*r))**2
            ph = 1.0 - exp(temp)
            pk(I) = ph*pkh(I)
            if (lev.eq.2) print 5, name(I), tof(I), ph, pk(I)
            call skedul (tstart(I),I,'fire  ',nrds(I))
30        CONTINUE
          call skedul (tstop,0,'stop  ',0)
          if (r.eq.minr .or. lev.gt.0) print 2
          if (lev.ge.1) print 6, t,p
          call events
          fr = tgtrg(r,minr,maxr,incr,meanr)
          print 3, r, fr, p, ex
          exmean = exmean + fr*ex
40      CONTINUE
        print 4, exmean
        END

        SUBROUTINE EVENTS
c       Events: Simulate all events in a single duel.
        character what*6
        integer who, which

20      CONTINUE
          call event(who,what,which,when)
          if (what.eq.'fire  ') call fire(when,who)
          if (what.eq.'impact') call impact(when,who,which)
        IF (what.ne.'stop  ') GOTO 20
        END

        SUBROUTINE FIRE (t,I)
c       Fire: Simulate the firing of a round.
        include 'duel4.h'
c       t - the current time (sec).
c       I - the current firer.
c       n - the remaining rounds for the firer.
c       dt - the delay before firing the next round.
c       k - the number of rounds in the current burst.
1       format(f11.2,a6,' fires at',a5)

        if (lev.eq.2) print 1, t, name(I), name(3-I)
        call skedul(t+tof(I),I,'impact',nrd(I))
        n = nrd(I)-1
        palive(I,n) = palive(I,n+1)
        nrd(I) = n
        IF (n.gt.0) THEN
        Fire next round.
          dt = dt1(I)
          k = mod(nrds(I)-n,nrb(I))
          if (k.eq.0) dt = dt2(I)
          call skedul(t+dt,I,'fire  ',n)
        ENDIF
        END

        SUBROUTINE IMPACT (t,I,k)
c       Impact: Simulate the impact of a round.
c       t - current time (sec).
c       I - firer ID.
c       k - round ID.
c       prdead - the probability Red is dead.
c       pbdead - the probability Blue is dead.
        include 'duel4.h'
1       format(3x, f8.2, a6, ' rd',i3,' hits.',f5.2,4f9.2)

        call markov(I,k,pk(I),palive,p,nrd)
c       Find exchange ratio [p(red dead) / p(blue dead)].
          ex = 0.0
          prdead = p(2)+p(4)
          pbdead = p(3)+p(4)
          if (pbdead .ge. .001) ex = prdead/pbdead
        IF (lev.ge.1) THEN
          if (ex.gt.0.0) print 1, t, name(I), nrds(I)-k+1, p, ex
          if (ex.eq.0.0) print 1, t, name(I), nrds(I)-k+1, p
        ENDIF
        END
        include 'tgtrg.f'
        include 'markov.f'
        include 'reset.f'
        include 'skedul.f'
        include 'event.f'
```

2.3 Lanchester: Find the Survivors of Lanchester Square Law Combat.

Lanchester finds the number of combat survivors versus time. It uses the lanchester square law which is applicable to general combat (not hand-to-hand combat, area fire, or guerrilla warfare).

Input/Output. The first input is the number of Blue and Red combatants at the start of battle. The second are the Blue and Red kill rates. The Blue kill rate is the probability of kill (p_k) times the rate of fire (r) in rounds per time unit. If $p_k = 0.5$, and $r = 4$ rounds/minute, then the kill rate is 2 kills/minute. A similar calculation gives the Red kill rate.

The output is the number of survivors on each side at each time step. The time units will be the inverse of the kill rate units, i.e., if the kill rate is 2 kills/minute then the time units will be minutes.

The following is a sample dialog.

```
% lanchester.x
 How many Blue, Red troops? (float)
100., 50.
 What is Blue, Red kill rate? (float)
0.2,0.1
 Time    Blue     Red
    0  100.00   50.00
    1   90.97   40.47
    2   83.76   31.75
    3   78.23   23.66
    4   74.26   16.05
    5   71.79    8.76
    6   70.75    1.64
  end   70.71    0.00
 How many Blue, Red troops? (float)
^C
*** Interrupt!
*** Execution terminated
```

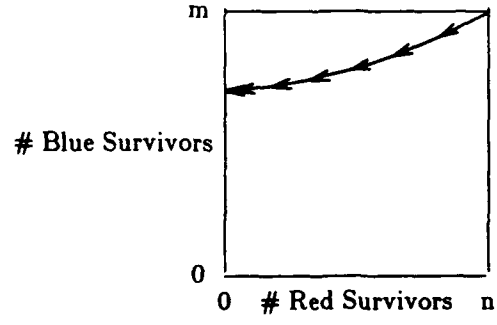


Figure 2.3 Survivors at Each Time Step

Mathematics. The lanchester square-law equations assume the following for battle:[2]

1. Everyone is visible and within range of fire.
2. Every member of one side randomly picks an opponent on the other side.
3. Each marksman fires until killed or makes a kill at which point he randomly picks a new target and resumes firing.
4. All marksmen fire independently.
5. Ammunition supply is unlimited.
6. Kill probabilities and firing rates are constant.
7. Battle continues until one side is eliminated.

Lanchester's equations are the following:

$$\dot{b} = -\beta r$$
$$\dot{r} = -\rho b$$

where

b is the number of 'blue' combatants, and
r is the number of 'red' combatants,
β is the number of 'blue' combatants killed by a single 'red' combatant in a unit of time, and
ρ is the number of 'red' combatants killed by a single 'blue' combatant in a unit of time.

The solutions are

$$b = b_o \cosh\sqrt{\beta\rho}t - \sqrt{\beta/\rho}\, b_o \sinh\sqrt{\beta\rho}t$$
$$r = r_o \cosh\sqrt{\beta\rho}t - \sqrt{\rho/\beta}\, r_o \sinh\sqrt{\beta\rho}t$$

When blue wins

$$b = \sqrt{b_0^2 - r_0^2\beta/\rho}, \quad r = 0.$$

When red wins

$$r = \sqrt{r_0^2 - b_0^2\rho/\beta}, \quad b = 0.$$

Code.

```
c       LANCHESTER: Simulates battle under lanchester square law.
1       format (' Time    Blue     Red')
2       format (i5,2f8.2)
3       format (' end',2f8.2)

20      Print *, 'How many Blue, Red troops? (float)'
        read *, b0, r0
        print *, 'What is Blue, Red kill rate? (float)'
        read *, beta, rho
        fac1 = sqrt(beta*rho)
        fac2 = sqrt(beta/rho)
        fac3 = sqrt(rho/beta)
        t = 0.0
        i = 0
        print 1
30      CONTINUE
         b = b0*cosh(fac1*t) - fac2*r0*sinh(fac1*t)
         r = r0*cosh(fac1*t) - fac3*b0*sinh(fac1*t)
         IF (b.gt.0 .and. r.gt.0) THEN
          print 2, i, b, r
          t=t+1.0
          i=i+1
         ELSE
          IF (b.gt.0) THEN
           b=sqrt(b0**2 - (beta/rho)*r0**2)
           r=0.0
          ELSEIF (r.gt.0) THEN
           r=sqrt(r0**2 - (rho/beta)*b0**2)
           b=0.0
          ELSE
           print *, 'Error: b, r= ',b,r
          ENDIF
          print 3, b, r
         ENDIF
        IF (b.gt.0.0 .and. r.gt.0.0) GOTO 30
        GOTO 20
        END
```

2. C. J. Ancker, Jr. and A. V. Gafarian, *The Validity of Assumptions Underlying Current Uses of Lanchester Attrition Rates*, TRAC-WSMR-TD-7-88, Department of the Army, US Army TRADOC Analysis Command, White Sands Missile Range, New Mexico 88002-5502, March 1988, pp. 1-2.

2.4 Tiny Wars: Simulate Combat Stochastically.

Tiny Wars is a stochastic, discrete event simulation of combat between homogeneous forces. It has limited capability but may be of some use as it stands. Primarily, it serves to show how one might build a Monte Carlo combat simulation and provides a base for developing more detailed, more realistic models.

In these battles, each side has guns or other long range weapons and no one moves or changes exposure. Detection, selection, and kills are random. After reading and echoing the input data, the main routine initializes summary statistics to zero, loops through the desired number of battles, and prints summary statistics.

When all the combatants on a side are killed, or time expires, the battle ends.

The following are likely extensions: extend detection to include firing signature detection (easy), motion (hard), mobility and firepower kills (hard), terrain intervisibility (hard).

Input. The data are free format, with numbers separated by blanks or commas. Below is a sample input file. The last line of the file contains 2 print flags. If the first flag is 0, minimal output prints; if it is 1, the number of survivors of each engagement is printed, and if it is 2, each event of each engagement is printed. The second flag is normally set to 0; each event scheduled or cancelled is printed if it is 1.

INPUT FILE	DESCRIPTION
3 3	Number of Blue, Red combatants
0.1 0.1	Probability Blue, Red detect 1 tgt in 1 sec.
0.5 0.5	Probability of kill given a shot.
5.0 5.0	Time of flight of round (sec).
13. 13.	Median time between shots (sec).
100 0 0 600. 1111111	# replications, print flag, print flag, max engagement time (sec), random number seed.

Output. The output contains the number of battles that end in each of 4 possible states: some alive on both sides, Blue wins, Red wins, and all dead. It also shows the number of Blue and Red survivors over all battles. A sample output file is:

Blue	Red	
3	3	Number of combatants
0.10	0.10	P(firer detects target in 1 second)
0.50	0.50	P(kill given a shot)
5.00	5.00	Time of flight
13.00	13.00	Time between rounds (sec)
100	1111111	Replications, seed
49	47	Blue, Red wins.
0	4	Draws w, w/o survivors.
77	82	Blue, Red survivors.

Code. The Tiny Wars code consists of the following code plus the code in section 2.5, the clock routines from sections 3.2 to 3.5, and the random number generators from sections 5.7 and 5.8.

The global file is:

```
      parameter (L=20)
      logical alive(L), busy(L), seen(L,L)
      common /inputc/ nblu, nred, pdet(2), pks(2), tof(2), tsub(2)
      common /inputg/ nreps, lev, lev2, tmax
      common /varblc/ alive, busy, ntgt(L), seen, t
```

The routines are:

```
      PROGRAM TINY WARS
c     Tiny Wars: Simulate battle nreps times.
      include 'global.h'
      common /crandm/ j
      integer nsys(2), sum(6)
1     format(a)
2     format(2i8,a)
3     format(2f8.2,a)

c     Read input data.
      read *, nblu, nred, pdet, pks, tof, tsub
      read *, nreps, lev, lev2, tmax, j
c     Echo input data.
      print 1, '   Blue    Red'
      print 2, nblu, nred,' Number of combatants'
      print 3, pdet,' P(firer detects target in 1 second)'
      print 3, pks,' P(kill given a shot)'
      print 3, tof,' Time of flight (sec)'
      print 3, tsub,' Time between rounds (sec)'
      print 2, nreps, j,' Replications, seed'
      print *
c     Initialize summary statistics.
      DO 20 k=1,6
        sum(k) = 0
20    CONTINUE
      DO 80 nrep=1,nreps
      call reset(lev2.ge.1)
      nsys(1) = nblu
      nsys(2) = nred
      call battle (nsys)
      if (lev.ge.1) print 2, nsys,' Survivors'
      if (nsys(1).gt.0 .and. nsys(2).eq.0) sum(1)=sum(1)+1
      if (nsys(1).eq.0 .and. nsys(2).gt.0) sum(2)=sum(2)+1
      if (nsys(1).gt.0 .and. nsys(2).gt.0) sum(3)=sum(3)+1
      if (nsys(1).eq.0 .and. nsys(2).eq.0) sum(4)=sum(4)+1
      sum(5)=sum(5)+nsys(1)
      sum(6)=sum(6)+nsys(2)
80    CONTINUE
c     Print summary statistics.
      print 2, sum(1), sum(2),' Blue, Red wins.'
      print 2, sum(3), sum(4),' Draws w, w/o survivors.'
      print 2, sum(5), sum(6),' Blue, Red survivors.'
      END
```

2.5 Tiny Wars Continued: a Single Battle.

The battle routine initializes the state of the combatants, starts the search process and then loops through the chains of subsequent events triggered by the search process. These events are: look, see, fire, and kill.

The look (search) event. The look event reschedules itself at 1 second intervals if alive systems have not yet detected alive targets. Look also schedules see events randomly based on detection probability.

The see (detect) event. The see event calls for target selection if the combatant is not busy firing at another target.

The fire event. The fire event schedules a subsequent fire event. It may also schedule a kill event. The probability of scheduling a kill event is equal to the probability of a kill given a shot.

The kill event. The kill event simulates destruction of a target and triggers a change in the firer's activity. It counts the destruction of the target, and cancels all detect events and fire events that have been scheduled for the target. It also calls the disengage routine which simulates the firer switching targets.

Target selection. The select routine simulates a firer selecting a target from among the alive foes he has seen.

Target disengagement. The disengage routine simulates systems disengaging a newly killed target. Any pending see events on the target are cancelled, then those engaging the target are disengaged and begin selection of a new target. *Disengagement means pending fire events are cancelled* and the systems are marked as not busy. Disengagement also has these systems select new targets and schedules fire on them.

When all the combatants on a side are killed, or time expires, the battle ends.

Code.

The routines are:

```
      SUBROUTINE BATTLE (nsys)
      include 'global.h'
      character what*6, color(2)*4
      logical o1, o2
      integer nsys(2)
      data color /' Blu',' Red'/
      data ALL, NULL /0,0/
3     format(f8.2,a4,i3,a7,a4,i3)

c     Set initial conditions for each combatant.
       DO 30 i=1,L
        alive(i) = .true.
        busy(i) = .false.
        ntgt(I) = 0
        DO 20 j=1,L
         seen(i,j) = .false.
20      CONTINUE
30     CONTINUE
       call skedul(tmax,NULL,'done..',NULL)
       call skedul(0.0, NULL,'looks ',NULL)
40    CONTINUE
c     Loop through all events or until tmax.
       call event(I,what,it,t)
       IF (what.eq.'done..') GOTO 70
c     Escape if max time is reached.
       m = 1
       if (I.gt.nblu) m=2
       n = 3-m
       if (lev.ge.2.and.what.ne.'looks ')
     1   print3,t,color(m),I,what,color(n),it
       IF (what.eq.'looks ') THEN
c      Simulate search.
        call search (1,2,1,nblu,nblu+1,nblu+nred,o1)
        call search (2,1,nblu+1,nblu+nred,1,nblu,o2)
        if ((o1.or.o2) .and. t.lt.tmax)
     1    call skedul (t+1.0,ALL,'looks ',ALL)
       ELSEIF (what.eq.'sees ') THEN
c      Simulate detection of a target.
        seen(I,it) = .true.
        if (alive(I) .and. .not.busy(I)) call select(I,m)
       ELSEIF (what.eq.'fires@') THEN
c      Simulate firing at a target.
        if (ranu().lt.pks(m)) call skedul(t+tof(m),I,'kills ',it)
        call rann(p,q)
        call skedul(t+tsub(m)*exp(0.5*p),I,'fires@',it)
       ELSEIF (what.eq.'kills ' .and. alive(it)) THEN
c      Simulate kill of a living target.
        nsys(n)=nsys(n)-1
        alive(it) = .false.
        call cancel(it,'fires@',NULL)
        call cancel(it,'sees ',NULL)
        call diseng(it,m)
       ENDIF
       GOTO 40
70     CONTINUE
       END

       SUBROUTINE DISENG (it,m)
c      Diseng: disengage any engaging it.
       include 'global.h'

       n1 = 1
       nn = nblu
       if (it.le.nblu) n1 = nblu+1
       if (it.le.nblu) nn = nblu+nred
       DO 20 I=n1,nn
        IF (ntgt(I).eq.it) THEN
         call cancel(I,'fires@ ',it)
```

```
            call cancel(I,'sees  ',it)
            busy(I) = .false.
            if (alive(I)) call select(I,m)
          ENDIF
20        CONTINUE
          END

          SUBROUTINE SEARCH (m,n,i1,in,it1,itn,repeat)
c         Search: Simulate searchers on side m searching for targets.
c         m, n - side of searcher, target.
c         i1, in - id of first, last searcher.
c         it1, itn - id of first, last target.
          include 'global.h'
          logical repeat

          repeat = .false.
          DO 30 I=i1,in
            IF (alive(I)) THEN
              DO 20 it=it1,itn
                IF (alive(it) .and. .not. seen(I,it)) THEN
                  p = ranu()
                  IF (p.lt.pdet(m)) THEN
                    call skedul(t+ranu(),I,'sees  ',it)
                  ELSE
                    repeat=.true.
                  ENDIF
                ENDIF
20            CONTINUE
            ENDIF
30        CONTINUE
          END

          SUBROUTINE SELECT (I,m)
c         Select: Combatant I on side m selects a target.
          include 'global.h'
          integer list(L)
1         format (f8.2,4x,i3,' picks  ',i5)

          it1 = 1
          itn = nblu
          if (I.le.nblu) it1 = nblu+1
          if (I.le.nblu) itn = nblu+nred
          k = 0
          DO 20 it=it1,itn
c           List seen, alive targets.
            IF (alive(it) .and. seen(I,it)) THEN
              k = k+1
              list(k) = it
            ENDIF
20        CONTINUE
          IF (k.gt.0) THEN
c           Randomly select from the list.
            j = 1+k*ranu()
            it = list(j)
            ntgt(i) = it
            call rann(p,q)
            p = exp(0.5*p)
            call skedul(t+tsub(m)*p,I,'fires@',it)
            busy(I) = .true.
            if (lev.ge.2) print 1, t, I, it
          ENDIF
          END
```

2.6 Tgtrg: Find Probability Target is in a Given Range Band.

Tgtrg is a subroutine for finding the probability that combat occurs within one or more range bands. It is based on data for tank vs tank combat in NW Europe in World War II. About 41% of the time visibility range was greater than 744 yards and about 41% of the time combat range was greater than 660 yards.

Tgtrg is used to weight simulated combat results at reasonable range bands. The average range between combatants will depend on the terrain and weather. More rugged terrain will generally have a smaller average range and flatter terrain will have a larger average range. However, we don't expect the average range between tank combatants to change much in NW Europe, where the data was taken. The terrain is the limiting factor and this doesn't change much. Factors that might affect this further are clearing of forests, the spread of urban areas, improved vision devices, and early warning.

Input. Tgtrg requires 5 inputs. They should be in a consistent set of units; all yards, all meters, etc.

r - the range between combatants
rmin - the minimum range for r
rmax - the maximum range for r
dr - the range increment
rmean - the constant defining the 'average' range

Normally, the calling routine will vary r between rmin and rmax. Tgtrg will then find the probability that combat occurs in a band from r-dr/2 to r+dr/2, however the first range band is from 0 to r+dr/2 and the last range band is from r-dr/2 to infinity. If rmin = rmax, the program assumes a single range band and returns a 1.0 as the probability that combat occurs in the range band.

Mathematics. Peterson[3] approximates the range between combatants as follows:

$$"G(r) = (1+2r/\mu_r)e^{-2r/\mu_r},$$

where

μ_r = 660 yards,

$G(r)$ = the fraction of casualties from ranges greater than R, and μ_r is the average* range of the distribution."

BRL MR 702[4], likewise approximates sighting ranges with the following distribution:

"... sighting ranges of NW Europe may be approximately represented by a cumulative distribution of the form

$$F(r) = (1+2r/\mu_r)e^{-2r/\mu_r},$$

with

μ_r = 744 yds."

It[5] further states: "Only 14% of the allied tanks destroyed by direct fire weapons were victims of guns at ranges greater than 1000 yds."

Figure 2.6 shows the probability that combat will occur in each range band. Note that the first band is from 0 to 300 meters, the last is from 2,500 meters to infinity, and the interior bands are only 200 meters wide.

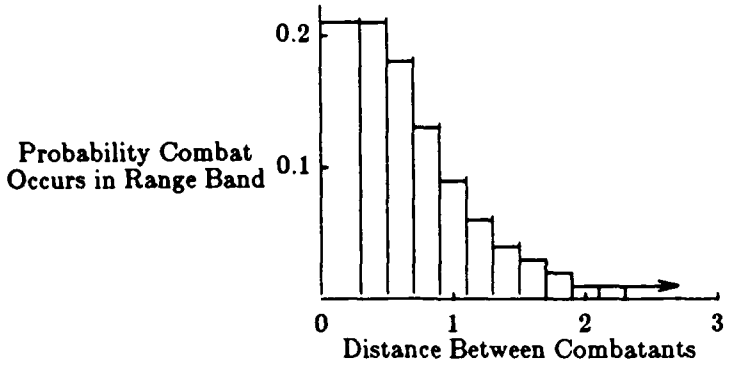


Figure 2.6 Probability of Combat in Each Range Band

Code.

```
      FUNCTION TGTRG (r,rmin,rmax,dr,rmean)
c     Tgtrg: Find probability tgt is in a given range band.
c     r - middle of range band (band is from r-dr/2 to r+dr/2,
c       except first band is from 0 to r+dr/2 and last band is
c       from r-dr/2 to infinity. If only 1 band, it is from 0 to infinity.
c     rmin - if r=rmin, we're at 1st band.
c     rmax - if r+dr/2 >= rmax, we're at last band.
c     rmean - mean rg to tgts. This defines the distribution.
      integer r, rmin, rmax, dr, rmean
      pr(x) = (1.0+2.0*x/rmean) * exp(-2.0*x/rmean)

      IF (r.le.rmin) THEN
c     Find for 1st range.
        IF (r+dr .gt. rmax) THEN
c       Find for only 1 range.
          p1 = 1.0
          p2 = 0.0
        ELSE
c       Find for 1st range of several.
          p1 = 1.0
          p2 = pr(r+0.5*dr)
        ENDIF
      ELSE
c     Find at later range.
        IF (r+dr .le.rmax) THEN
c       Find at an intermediate range.
          p1 = pr(r-0.5*dr)
          p2 = pr(r+0.5*dr)
        ELSE
c       Find at last range.
          p1 = pr(r-0.5*dr)
          p2 = 0.
        ENDIF
      ENDIF
      tgtrg = p1-p2
      END
```

---

3. Peterson, R. H., *The Range and Angular Distribution of A.P. Hit Tanks*, BRL MR 590A, APG, MD.

* The phrase 'average range' needs explanation. If $r = \mu_r$, then $G(r) = 3e^{-2} \approx 0.406$. Hence there's a 40.6% chance the combatants are more than 660 yards apart. Likewise, there's a 40.6% chance the 'visibility range' is more than 744 yards. Here, visibility range means the range to masking terrain.

4. Peterson, R. H., Hardison, D. C., Benvienuto, A. A., *Terrain and Ranges of Tank Engagements*, June 1953, Ballistic Research Laboratory, APG, MD, p3.

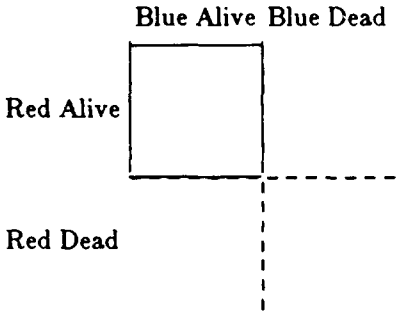
5. ibid p5.

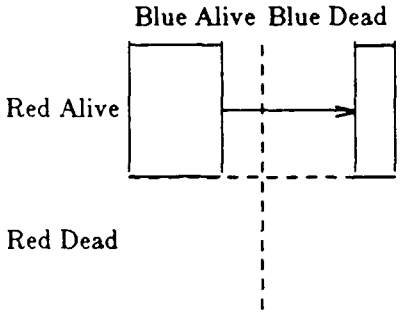
2.7 Markov: Update Kill Probabilities

In realistic duels, the time of flight of rounds is non-zero; so both combatants can have rounds in the air simultaneously and both can end up dead. Markov updates the 4 possible states of such a duel: both alive, Blue wins, Red wins, both dead.

As time passes during combat, the probability of either combatant remaining alive decreases. At any instant, a given combatant is in one of two states; dead or alive. If we consider both combatants, there are 4 possible states; both alive, Blue alive & Red dead, Blue dead & Red alive, and both dead.

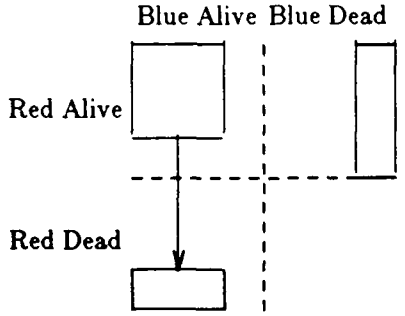
We can visualize the computations involved by imagining a sheet of paper being cut. The width of the paper is proportional to the probability Blue is alive and the height is proportional to the probability Red is alive. This is illustrated on the left below.





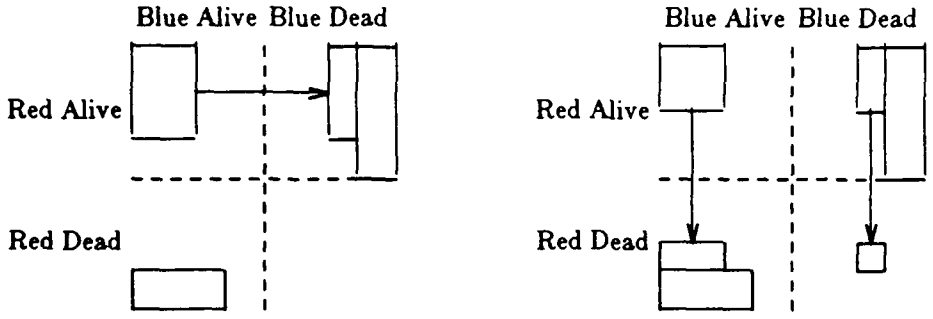
If the probability of kill given a shot is 0.3 for both combatants and a Red round hits before Blue has fired then the probability of Blue being alive is reduced to 0.7. Analogously, we may cut off the right 30% of the sheet and move it to the upper right quadrant as shown on the right above.

Now, imagine that a Blue round is fired and impacts before Red fires a second round. In our paper cutting analogy, we cut off the lower 30% of the sheet remaining in the upper left quadrant and move it to the lower left quadrant as shown in the figure below.



All this is quite simple. Suppose, however, that the second Blue round is in flight when the second Red round impacts. The paper slicing analogy is continued in the following two diagrams. First, Red round number 2 strikes as shown on the left.

Then, Blue round number 2 strikes and not only must we cut off the lower 30% of the sheet in the upper left quadrant, but we must cut off the lower 30% of the small sheet we just moved to the upper right quadrant and move it to the lower right quadrant. This is shown on the right below. This last operation is the tricky one; we must account for the times when several rounds are in the air resulting in the possibility that both combatants may end up dead.



Input/Output. Markov requires as input these integers:

i - integer identifying the firer (1=Blue, 2=Red)

l - number of this round by firer

n_j - number of rounds remaining on jth system

and the probabilities that:

K - firer kills the target given a shot

$P_{j,k}$ - jth system was alive when it had k remaining rounds

S_0 - both were alive

S_1 - Blue won

S_2 - Red won

S_3 - both were dead

Markov updates and returns:

$P_{j,n}$ - probability target is alive just after the round arrived

S - new state probabilities

Mathematics. On impact, the probability that neither is dead decreases and the probability that the firer wins increases by an equal amount. The first pair of equations below account for this. The probability that the target wins decreases and the probability that both die increases by an equal amount. The second pair of equations below account for this. The last equation updates the probability that the target is alive. In each of these equations, we simply find the area of the appropriate rectangle and increment or decrement another area.

$$S_0' = S_0 - KS_0$$
$$S_i' = S_i + KS_0$$
$$S_j' = S_j - KP_{j,n}(P_{i,l} - P_{i,m})$$
$$S_3' = S_3 + KP_{j,n}(P_{i,l} - P_{i,m})$$
$$P_{j,n}' = P_{j,n} - KP_{j,n}$$

Code.

```
      SUBROUTINE MARKOV (I,l,k,p,s,nrd)
c     Markov: update markov state probabilities.
c     i - firer ID
c     k - probability of kill given a shot (real).
c     l - ID of impacting round.
c     m - ID of last round I fired
c     n - ID of last round it (j) fired.
      real k, p(2,50), s(0:3)
      integer nrd(2)

      j = 3-i
c     Find prob firer alive now & prob tgt alive before this impact.
        m = nrd(i)
        n = nrd(j)
c     Update prob firer wins & prob firer draws.
        del = k*s(0)
        s(0) = s(0)-del
        s(i) = s(i)+del
c     Update prob tgt wins & prob tgt draws.
        del = k * p(j,n) * (p(i,l)-p(i,m))
        s(j) = s(j) - del
        s(3) = s(3) + del
        p(j,n) = p(j,n) - k*p(j,n)
      END
```

3. Event Routines

3.1 Event Handling Using Linked Lists.

The two major ways of handling events are stepping a fixed time interval and stepping to the next significant event. Stepping to the next significant event (the method discussed here) requires routines to reset (initialize) the data structure, schedule an event, fetch an event, and cancel events. In this section, we touch on various techniques for handling the event data and then discuss the Linked List technique used by the software in the next four sections.

As a minimum, we must store the time at which an event will occur, the identity of the entity that will perform the event, and the type of event. It may also be desirable to store the entity receiving the action of the event and other information about the event. If, at the current time t, the program finds that after a delay of 5 seconds, tank 4 may fire at tank 6, this would require a Fortran call as follows:

```
call skedul (t+5.0,4,'fire..',6)
```

Typically, the numbers in the above call would be variables.

Methods of handling event data. A great many methods have been used for storing and retrieving event data. The simplest is to add an event to the end of a list and when the next event is needed, simply search the list for the event with the smallest time. The next simplest is to insert the event just before the next following event. This requires moving the next and all subsequent events down in the list and is slow. The method used here uses linked lists, so that the events are always sorted chronologically, but records of subsequent events need not be moved. Other methods* include a partially sorted structure called a heap.

The search from the front linear linked-list technique was used in the algorithms in the following sections. The key elements are:

- A set of links
- A pointer to the first idle link
- A pointer to the link containing the next event
- Several auxiliary pointers for manipulating the links

Initially, the 'idle' pointer points to the first available link, which points to the subsequent link, and so on until the last available link, which points to the null link Ω. The 'next event' pointer points to Ω also. When an event is inserted in the list, the algorithm removes the first idle link from the chain of idle links, inserts it chronologically in the chain of active links, and inserts the event data into the link.

Retrieving the next event simply involves copying the data from the first link of the chain of active links, removing the link from that chain, and inserting it at the head of the chain of idle links.

Cancelling an event is similar, but involves links anywhere in the chain of active links. This implementation stores up to 100 events. Each type of information is stored in an array dimensioned to 100, however a given link consists of the ith element of each array. The arrays are:

---

* For a comparison of eight different event handling algorithms on a variety of differing simulation problems, see Current Issues in Computer Simulation, *Comparison of Future Event Set Algorithms for Simulations of Closed Queuing Systems*, McCormack, William M., and Sargent, Robert G., Academic Press, NY, 1979, pp 71-82. None of the eight was fastest at all 12 problems, however the method described here was best for 6 of them.

real when(100)	Time of the event
integer who(100)	The entity performing the event
character*6 what(100)	The type of event performed
integer whom(100)	The entity receiving the event
integer next(100)	The pointer to the next link

when(9)	who(9)	what(9)	whom(9)	next(9)
18.32	4	fire..	6	31

Figure 3.1 Contents of a Link

3.2 Reset: Re-initialise the Event List

The Reset subroutine 'resets the clock' to time zero. To do this it rebuilds the linked list of idle events and clears the linked list of active events. It is one of four routines, Reset, Skedul, Cancel, and Event, that cooperate to handle events in Monte Carlo Simulations. Although it was designed for use in combat simulations, it has much broader use.

If the single argument to reset is true, the event routines will print out each event as it is scheduled or cancelled; if false, this printing is not done. The subroutine then builds a linked list of idle links, as shown at the top of exhibit 3.2. It also makes a null linked list of active links as shown at the bottom of exhibit 3.2; no events are yet scheduled.

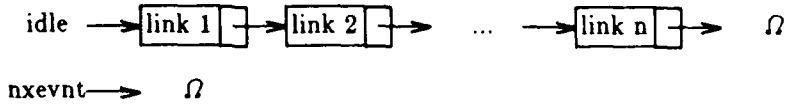


Figure 3.2 The Initial Linked Lists

Code.

```
c V7.1
c           clock.h file
            parameter (NE=200)
            character*6 what
            integer who, whom
            logical prflag
            common /event1/ what(NE)
            common /event2/ when(NE), who(NE),
     1       whom(NE), next(NE), nxevnt  nxidle, prflag
            save /event1/, /event2/
c V7.1
            SUBROUTINE RESET (prflg)
c 0         Reset: Initialize the clock to time zero.
            include 'clock.h'
            logical prflg
c
            prflag = prflg
            nxevnt = 0
            nxidle = 1
            DO 10 j=1,NE
              next(j) = j+1
10          CONTINUE
            next(NE) = 0
            END
```

INTENTIONALLY LEFT BLANK.

3.3 Skedul: Schedule an Event.

The Skedul subroutine schedules an event in a linked list of events. It is one of four routines, Reset, Skedul, Cancel, and Event, that cooperate to handle events in Monte Carlo simulations. Although it was designed for use in combat simulations, it has much broader use. The event information stored is; event type, entity that will perform the event, time the event will occur, and perhaps the receiver of the action.

The calling statement. A typical calling statement is:

call skedul (t+tf,I,'impact',it)

Where:

t is the current time,
tf is the time delay after which the event may occur,
I is the subject or actor causing the event,
'impact' is a 6 character string identifying the type of event, and
it is an integer identifying the object of the event.

Note that the event must always occur in the future, so in our example, tf $>$ 0.0.

Algorithm. On average, Skedul must traverse half the linked list to find the place to insert the event link. It must also check to see if an idle link is available. If so, it then inserts the new event using these 6 steps, as shown in exhibit 3.3.

1. Store the index of the idle link/event in n.
2. Store the index of the new head of the idle chain in idle.
3. Store the index of the immediately preceding link/event in l.
4. Store the index of the succeeding idle link/event in m.
5. Store the index of the now active link/event in next(l).
6. Store the index of the succeeding link/event in the now active link/event in next(n).

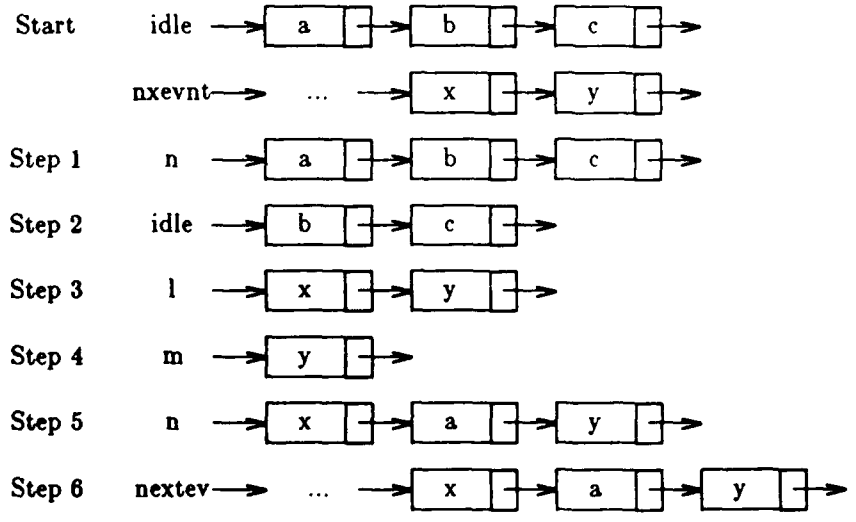


Figure 3.3 Scheduling an Event

Code.

```
c V7.3
      SUBROUTINE SKEDUL (t,I,act,it)
c 9   Schedule: Schedule an event for later execution.
      include 'clock.h'
      character*6 act
1     format(9x,'skedul ',i3,' ',a6,i3,' at time',f8.2)
c
      if (prflag) print 1, I, act, it, t
      IF (nxidle.eq.0) THEN
c     If storage all used stop
        print *,' Storage overloaded with too many events.'
        STOP
      ELSE
c     Store the event
c       Cut storage unit from empties
          n = nxidle
          nxidle = next(nxidle)
c       Then find where to insert this event in the event list.
        IF (nxevnt.le.0) THEN
c       New event is only event
          next(n) = 0
          nxevnt = n
        ELSE
c       Then find where to insert it.
c         Point to first 2 events
            l = nxevnt
            m = next(l)
c         Find where to insert them
          IF (t.ge.when(l)) THEN
c         See if between 2 scheduled events.
c           Loop till found.
20          IF (m.ne.0 .and. t.ge.when(m)) THEN
              l = m
              m = next(m)
              GOTO 20
            ELSE
c           Splice new event into list
              next(n) = m
              next(l) = n
            ENDIF
          ELSE
c         Place new event as most imminent
            next(n) = nxevnt
            nxevnt = n
          ENDIF
        ENDIF
c       Finally store event info
        when(n) = t
        what(n) = act
        who(n) = I
        whom(n) = it
      ENDIF
      END
```

3.4 Event: Find Next Event

The Event subroutine finds the next event to be simulated from a linked list of events. It is one of four routines, Reset, Skedul, Cancel, and Event, that cooperate to handle events in Monte Carlo simulations. Although it was designed for use in combat simulations, it has much broader use.

The event routine simply extracts the information for the next event from the first link on the linked list of events and then moves that link to the head of the linked list of idle links. The information extracted is:

I - the entity performing the event
act - the event or act
it - the object of the event (or other useful information)
t - the time the event occurs

Figure 3.4 shows the arrangement of the idle and active linked lists before and after the most imminent event is fetched.

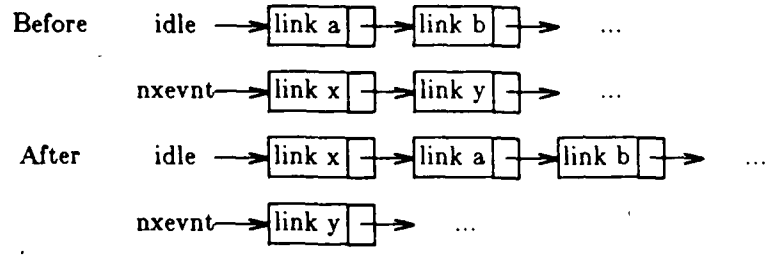


Figure 3.4 Selecting the Next Event

Code.

```
c V7.2
      SUBROUTINE EVENT (I,act,it,t)
c 0   Event: Find the next scheduled event.
      include 'clock.h'
      character*6 act
c
c     Fill arguments
        I = who(nxevnt)
        act = what(nxevnt)
        it = whom(nxevnt)
        t = when(nxevnt)
c     Drop storage unit from active storage chain
        n = nxevnt
        nxevnt = next(nxevnt)
c     Add storage unit to inactive storage.
        next(n) = nxidle
        nxidle = n
      END
```

INTENTIONALLY LEFT BLANK.

3.5 Cancel: Cancel an Event

The Cancel Subroutine cancels an event from a linked list of events. It is one of four routines, Reset, Skedul, Cancel, and Event, that cooperate to handle events in Monte Carlo Simulations. Although it was designed for use in combat simulations, it has much broader use.

Cancel removes zero or more links (events) from the list of scheduled events and places them in the linked list of idle links. This removes the record of these events, so they never occur. The cancel routine is called in the four ways illustrated below:

```
call cancel (I,'fire  ',it)
call cancel (I,'all   ',it)
call cancel (I,'all   ',NULL)
call cancel (I,'fire  ',NULL)
```

The first call to cancel cancels any fire events associated with entity I and object it. The second version cancels all events associated with entity I and object it. The third version cancels all events associated with entity I, no matter what is the object of the action. The third version cancels all fire events associated with entity I.

Figure 3.5 shows how the active and idle chains look before and after cancelling the second active event; event y.

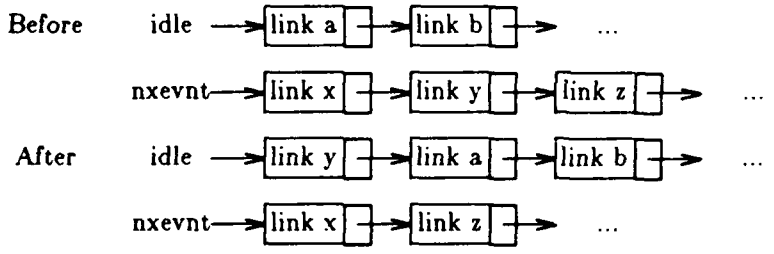


Figure 3.5 Cancelling an Event

Code.

```
c V7.1
      SUBROUTINE CANCEL (I, act, it)
c 0   Cancel: cancel 'act' events for 'I' entity.
c       (all events if act='')
c     Definitions of local variables:
c       m - pointer to previous event
c       n - pointer to current event being considered
      include 'clock.h'
      logical is what, is who, is whom
      character*6 act
1     format(9x,'cancel ',i3,' ',a6,i3,' at time',f8.2)
c
      m = 0
      n = nxevnt
10    IF (n.ne.0) THEN
c     Continue until n=0
        is who  = I .eq.who(n)
        is what = act.eq.what(n) .or. act.eq.'all   '
        is whom = it.eq.whom(n) .or. it.eq.0
        IF (is who .and. is what .and. is whom) THEN
c       Then remove event
          if (prflag )print 1, I, act, it, when(n)
          if (m.eq.0) nxevnt = next(n)
          if (m.ne.0) next(m) = next(n)
          next(n) = nxidle
          nxidle = n
          if (m.eq.0) n = nxevnt
          if (m.ne.0) n = next(m)
        ELSE
c       Don't remove event. Shift to next event.
          m = n
          n = next(n)
        ENDIF
        GOTO 10
      ENDIF
      END
```

INTENTIONALLY LEFT BLANK.

4 Mathematical

4.1 Center: Find the Centroid of a Polygon.

If you want to know the centroid (center of gravity) of an irregular shape, for instance to find the aim point on a target, use the center function. Input a set of x,y coordinates representing consecutive corner points of a polygon with 3 to 30 sides. Center will compute the centroid and output its coordinates.

Input/Output. This routine reads in from a data file one pair of x,y coordinates per line. The coordinates correspond to consecutive corners of a polygon. They may be ordered either clockwise or counterclockwise. You may enter a maximum of 30 points, which would describe a thirty-sided shape. Center responds with the x,y coordinates of the polygon's centroid. Figure 4.1 shows a sample input and output as well as a diagram of the polygon to which they refer.

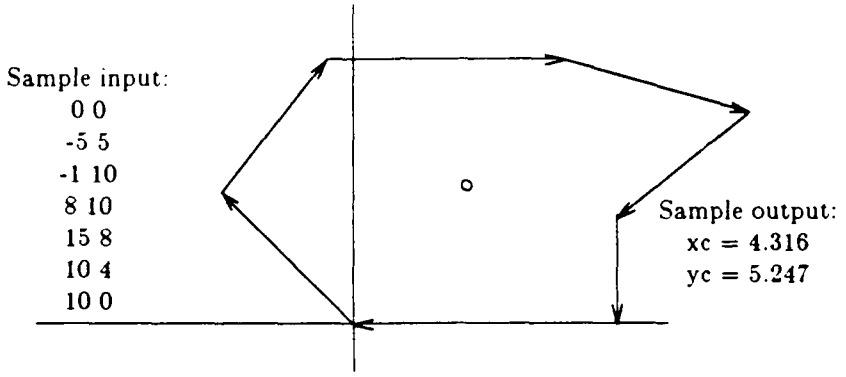


Figure 4.1 Centroid of a Polygon

Mathematics. Center 'integrates' under the perimeter vectors to find the area bounded by the polygon. It takes one line segment (i) at a time and determines the area between that segment and the x-axis. To do this, the program divides each area into a rectangle and a triangle and calculates the area and centroid for both. For the rectangles:

$$ar_i = (x_i - x_{i-1})y_{i-1}$$
$$xr_i = (x_i + x_{i-1})/2$$
$$yr_i = y_{i-1}/2$$

And for the triangles:

$$at_i = (x_i - x_{i-1})(y_i - y_{i-1})/2$$
$$xt_i = (2x_1 + x_{i-1})/3$$
$$yt_i = (y_i + 2y_{i-1})/3$$

Where,

ar_i, at_i are the areas of the rectangle and triangle,
xr_i, xt_i are the x coordinates of the centroids, and
yr_i, yt_i are the y coordinates of the centroids.

As the program moves around the perimeter above the x-axis, travel to the right adds area to the cumulative area, and travel to the left subtracts. The opposite occurs when the line segment is below the x-axis. After each segment, the program calculates the cumulative moment and area. Finally, the total moment divided by the total area gives us the coordinates of the centroid (xc,yc):

$$xc = \frac{\Sigma(ar_i xr_i + at_i xt_i)}{\Sigma(ar_i + at_i)} \; ; \; yc = \frac{\Sigma(ar_i yr_i + at_i yt_i)}{\Sigma(ar_i + at_i)}$$

Code.

```
      PROGRAM CENTER
c     Center: Find the centroid of a polygon.
      dimension x(30), y(30)
1     format(' xc=',f10.3,'  yc=',f10.3)

      DO 20 nseg=1,30
       read(5,*,end=25)x(nseg),y(nseg)
       nsegs = nseg
20    CONTINUE
c     Set initial values
25     sum A = 0
       sum Mx = 0
       sum My = 0
       x1 = x(nsegs)
       y1 = y(nsegs)
      DO 30 nseg=1,nsegs
       x2 = x(nseg)
       y2 = y(nseg)
c      Find area and cg of rectangle
         ar = (x2-x1)*y1
         xr = (x1+x2)/2
         yr = y1/2
c      Find area and cg of triangle
         at = (x2-x1)*(y2-y1)/2
         xt = (x2+x2+x1)/3
         yt = (y1+y1+y2)/3
c      Accumulate moments and areas
         sum Mx = sum Mx + ar*xr + at*xt
         sum My = sum My + ar*yr + at*yt
         sum A = sum A + ar + at
c      Set beginning of next line segment
         x1 = x2
         y1 = y2
30    CONTINUE
c     Find final result
       xc = sum Mx/sum A
       yc = sum My/sum A
       print 1,xc,yc
      END
```

4.2 Indexx: Find an Index in a Table for Interpolation Purposes.

To interpolate in tables, use the indexx function. Indexx assumes that the dependent variable is stored in a vector of reals, for example, x(1) .. x(n), in ascending or descending order. Given the arguments x, n, x_s, where x is an ascending vector, it finds the value i such that $x_i \leq x_s \leq x_{i+1}$ using binary search. If $x_s < x_i$ or $x_s > x_n$, it returns a zero value for i.

Suppose we wish to linearly interpolate in the table shown in exhibit 4.2. We may use the following lines of code, where the second line is a statement function:

```
real f(10), x(10)
fj(xj) = f(i) + (f(i+1)-f(i)) * (xj-x(i)) / (x(i+1)-x(i))
i = indexx(x,10,xj)
y = fj(xj)
```

Table 4.2 Find an index

x	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5
f(x)	0.0	0.4	0.81	1.23	1.68	2.10	2.52	2.95

Code.

```
c V7.2
      FUNCTION INDEXX(a, n, x)
c     Find the index j, where a(j) < x < a(j+1)
      integer n ,lo, hi, mid
      logical incres, above
      real a(n), x

      incres = a(n).gt.a(1)
      lo=0
      hi=n+1
10    IF (hi-lo.gt.1) THEN
        mid=(hi+lo)/2
        above=x.gt.a(mid)
        IF (incres.eqv.above) THEN
          lo=mid
        ELSE
          hi=mid
        ENDIF
      GOTO 10
      ENDIF
      indexx=lo
      END
```

INTENTIONALLY LEFT BLANK.

4.3 LinEqs: Solve a Set of Linear Equation

LinEqs uses the simple text book method for solving a set of up to 19 linear equations. If you have a single problem to solve, want to check out another algorithm for linear equations, or are just learning how to solve linear equations, this program works fine. It is not robust, however, because it fails if there is a zero element on the diagonal (which you may be able to get around by re-arranging the input lines). You should use another algorithm* if you wish to solve many sets of linear equations.

Input/Output. The program reads in each of the simultaneous linear equations as a row of coefficients separated by commas or blanks. Each equation must be on a separate input line. In the first line put the number of equations (19 is the maximum) and a 't' or 'f', depending on whether you want the program to trace its steps or just print the results. You must arrange the equations in an order that does not place zeroes on the diagonal. If the equations are dependent, LinEqs will tell you. Otherwise, it will return the solution to the system of equations. As an example, consider the following four simultaneous linear equations:

$$\begin{aligned} 5x_1 + 1x_2 + 3x_3 + 0x_4 &= 16 \\ 1x_1 + 4x_2 + 1x_3 + 1x_4 &= 11 \\ -1x_1 + 2x_2 + 6x_3 - 2x_4 &= 23 \\ 1x_1 - 1x_2 + 1x_3 + 4x_4 &= -2 \end{aligned}$$

The program gives its solution as $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, and $x_4 = -1$.

Input File	Output file
4 f	Input is:
5. 1. 3. 0. 16.	5.00000 1.00000 3.00000 0.00000 16.00000
1. 4. 1. 1. 11.	1.00000 4.00000 1.00000 1.00000 11.00000
-1. 2. 6. -2. 23.	-1.00000 2.00000 6.00000 -2.00000 23.00000
1. -1. 1. 4. -2.	1.00000 -1.00000 1.00000 4.00000 -2.00000
	SOLUTION IS:
	1.00000 2.00000 3.00000 -1.00000

If you change the 'f' to a 't' on the first input line, the output will contain a step by step description of the solution. Below is a fragment showing how the off diagonal elements in the first column are zeroed.

```
:
 Row  1 Col  1 is on the diagonal.
 Row  2 Col  1 is being zeroed.
-1.00000  -0.20000  -0.60000   0.00000  -3.20000
 Row  3 Col  1 is being zeroed.
 1.00000   0.20000   0.60000   0.00000   3.20000
 Row  4 Col  1 is being zeroed.
-1.00000  -0.20000  -0.60000   0.00000  -3.20000
 Current matrix is:
```

* We highly recommend the algorithms in Numerical Recipes by Press et al. *Numerical Recipes*, Press, W., Flannery, B., Teukolsky, S., Vetterling, W., Cambridge University Press, NY.

```
 5.00000   1.00000   3.00000   0.00000  16.00000
 0.00000   3.80000   0.40000   1.00000   7.80000
 0.00000   2.20000   6.60000  -2.00000  26.20000
 0.00000  -1.20000   0.40000   4.00000  -5.20000
:
```

Mathematics. LinEqs works by zeroing the off-diagonal elements of the matrix that has one linear equation as each row. The program proceeds down each column, except the last column of constants, setting each element to zero by adding to its row the appropriate multiple of another row. This method eventually isolates each variable on the diagonal, so that its value becomes apparent.

Code.

```
      PROGRAM LINEQS
c     LinEqs: Solve simultaneous linear equations.
c     Trace each step if desired.
      real c(20,20), x(20)
      integer n
      logical trace
1     format(10f10.5)
2     format('  Row ',i2,' Col ',i2,' is on the diagonal.')
3     format('  Row ',i2,' Col ',i2,' is already zero.')
4     format('  Row ',i2,' Col ',i2,' is being zeroed.')

c     Read equations
      read *, n, trace
      m = n+1
      DO 20 i=1,n
        read *, (c(i,j),j=1,m)
        IF (c(i,i).eq.0) THEN
          print *,'Re-arrange eqns so zeros aren''t on diagonal.'
          STOP
        ENDIF
20      CONTINUE
c     Zero off-diagonal elements of jth column
      DO 70 j=1,n
        if (j.eq.1) print *, 'Input is:'
        if (j.ne.1 .and. trace) print *, 'Current matrix is:'
        DO 30 i=1,n
          if (j.eq.1 .or. trace) print 1,(c(i,k),k=1,m)
30      CONTINUE
        IF (abs(c(n,n)).lt.0.00001) THEN
          print *,'The equations are probably not independent.'
          STOP
        ENDIF
        DO 60 i=1,n
          IF (i.eq.j) THEN
            if (trace) print 2, i, j
          ELSEIF (c(i,j).eq.0.0) THEN
            if (trace) print 3, i, j
          ELSE
            if (trace) print 4, i, j
            DO 40 k=1,m
              x(k) = - c(j,k)*c(i,j)/c(j,j)
40          CONTINUE
            if (trace) print 1, (x(k),k=1,m)
            DO 50 k=1,m
              c(i,k) = c(i,k) + x(k)
50          CONTINUE
            y = c(n,n)
          ENDIF
60        CONTINUE
70      CONTINUE
        print *
        DO 80 i=1,n
          if (trace) print 1,(c(i,k),k=1,m)
          x(i) = c(i,m)/c(i,i)
80      CONTINUE
        print *, 'SOLUTION IS:'
        print 1, (x(i),i=1,n)
      END
```

4.4 Xform: Transform Cartesian Coordinates

A target is being approached by a projectile. The target may be approximated by a concave or convex polyhedron with convex polygons as faces. We know the coordinates of the corners of the polygons, i.e. of the target. We also know the position and velocity of the incoming projectile near the time of impact. For convenience of further calculation, Xform rotates and translates all these coordinates so that the projectile is traveling through the origin of the new coordinate system along the y axis in the negative direction. Normally, other software then finds whether a projectile penetrates the target or integrates the bivariate normal to find the hit probability.

Input/Output. The main program passes to the xform subroutine the following values:

The number of corners (in ncorn)
The coordinates of the corners (in matrix corner(NN,3)))
The projectile position (in vector p(3))
The projectile velocity (in vector v(3))
output level

The subroutine passes the newly calculated corner positions back to the calling program in matrix cornr2(nn,3). If the output level option is 3 it also prints out the projectile position after rotation as well as the final (x,y,z) coordinates of each corner.

Mathematics. First, xform rotates the projectile coordinates through the angles θ and ϕ, determined from the projectile velocity as shown in Figure 4.4. This reorients the velocity vector such that it becomes aligned with the positive y axis.

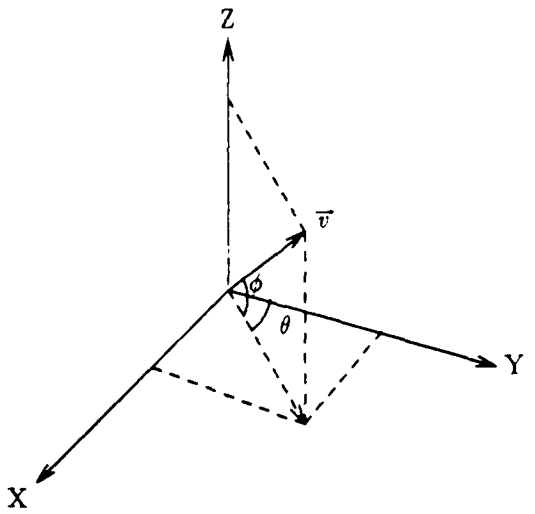


Figure 4.4 The Rotation Angles θ and ϕ

We can represent each rotation by a matrix, such that multiplying that matrix by any set of coordinates (x,y,z) will yield "new" rotated coordinates (x',y',z'). We can rotate first through the angle θ and then through the angle ϕ with the following matrix operation:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Xform applies the corresponding algebraic relations to rotate the projectile coordinates:

$$p_x' = p_x \cos\theta - p_y \sin\theta$$
$$p_y' = p_x \sin\theta \cos\phi + p_y \cos\theta \cos\phi + p_z \sin\phi$$
$$p_z' = -p_x \sin\theta \sin\phi - p_y \cos\theta \sin\phi + p_z \cos\phi$$

Next, xform rotates the corner coordinates using the same equations. It also has to translate the target corners to correspond with a projectile positioned at the origin. To do this translation, the routine simply subtracts (p_x', p_y', p_z') from each set of rotated corner coordinates.

Code

```
      SUBROUTINE XFORM(cornr2)
c     Xform: Rotate and translate target & proj to proj base coords.
      parameter (NN=25)
      common /comtgt/ corner (NN,3), iface(NN,6), ncorns, nfaces
      common /projec/ v(3), p(3), lev
      real cornr2(NN,3)
1     format(' New projectile pos:',3f10.3)
2     format(' New corner positions are:')
3     format(20x,3f10.3)

c     Find sin, cos of 1st rotation angle
       fac  = sqrt(v(1)**2 + v(2)**2)
       fac1 = 1.0/fac
       sin1 = v(1)*fac1
       cos1 = v(2)*fac1

c     Find sin, cos of 2nd rotation angle.
       fac2 = 1.0/sqrt(v(1)**2 + v(2)**2 + v(3)**2)
       sin2 = v(3)*fac2
       cos2 = fac*fac2

c     Rotate projectile position
       xp = p(1)*cos1 - p(2)*sin1
       yp = p(1)*sin1*cos2 + p(2)*cos1*cos2 + p(3)*sin2
       zp = -p(1)*sin1*sin2 - p(2)*cos1*sin2 + p(3)*cos2
       if (lev.eq.3) print 1, xp, yp, p(3)

c     Rotate & translate coordinates of target corners
       DO 20 nc=1,ncorns
         cornr2(nc,1) = corner(nc,1)*cos1 - corner(nc,2)*sin1 - xp
         cornr2(nc,2) = corner(nc,1)*sin1*cos2 +
     1     corner(nc,2)*cos1*cos2 + corner(nc,3)*sin2 - yp
         cornr2(nc,3) = -corner(nc,1)*sin1*sin2 -
     1     corner(nc,2)*cos1*sin2 + corner(nc,3)*cos2 - zp
20     CONTINUE
       if (lev.eq.3) print 2
       if (lev.eq.3) print 3,((cornr2(i,j),j=1,3),i=1,ncorns)
      END
```

5. Probability and Statistics

5.1 Binomial: Calculates Values of the Binomial Probability Function.

Given the two parameters of the binomial probability distribution, $p \equiv$ the probability of a success, and $n \equiv$ the number of trials, this program calculates the probability of i successes, $i=0,1,...,n$. The binomial probability distribution is discussed and a sample run is shown.

Input/Output. Binomial prompts for the probability of success in a single trial and the number of trials. It then finds the probabilities of zero to n successes. If the number of trials is greater than 20, it abbreviates the output by finding only the probabilities for zero to five trials and for multiples of five trials thereafter. To produce all values, simply delete the IF...THEN and ENDIF statements in the main routine.

Here is a sample dialog, showing how to find the probability of zero to 10 hits on a target, given ten shots and a probability of 0.2 of hitting with a single shot. Figure 5.1 illustrates the results of this calculation.

```
% a.out
 Binomial distribution.
 What is prob, number of trials?
0.2, 10
 Probability of  0 successes =  0.11
 Probability of  1 successes =  0.27
 Probability of  2 successes =  0.30
 Probability of  3 successes =  0.20
 Probability of  4 successes =  0.09
 Probability of  5 successes =  0.03
 Probability of  6 successes =  0.01
 Probability of  7 successes =  0.00
 Probability of  8 successes =  0.00
 Probability of  9 successes =  0.00
 Probability of 10 successes =  0.00
 What is prob, number of trials?
*** Interrupt!
*** Execution terminated
%
```

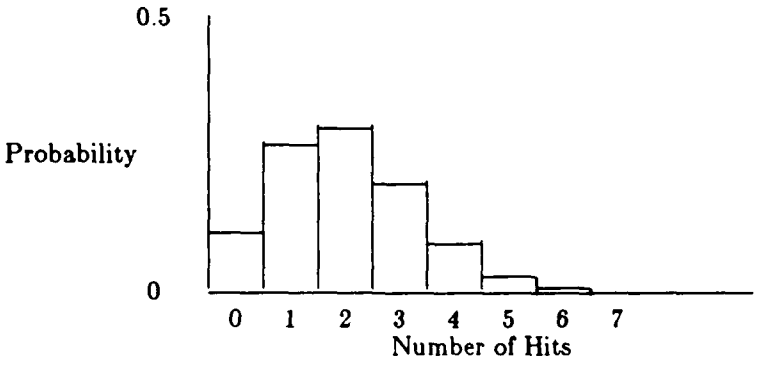


Figure 5.1 Probability of Exactly K Hits

Mathematics. The binomial probability function is

$$P_k = \binom{n}{k} p^k (1-p)^{n-k}, \quad k=0,1,\ldots,n$$

Where,

P_k is the probability of exactly k successes in n trials, and
p is the probability of 1 success in 1 trial,

Code.

```
      PROGRAM BINOMIAL
1     format (' Probability of', i3, ' successes =', f6.2)

      print *, 'Binomial distribution.'
20    print *, 'What is prob, number of trials?'
      read *, p, n
      DO 30 i=0,n
       IF (n.le.20 .or. i.lt.5 .or. 0.eq.mod(i,5)) THEN
         pi = bico(n,i) * p**i * (1.-p)**(n-i)
         print 1, i, pi
       ENDIF
30    CONTINUE
      GOTO 20
      END

      FUNCTION BICO (i,j)
c     Bico: find binomial coefficient.
      IF (j.eq.0 .or. j.eq.i) THEN
       bico = 1
      ELSE
       k = min0(j,i-j)
       p = i-k+1
       DO 20 m=2,k
         p = p*(i-k+m)/float(m)
20     CONTINUE
       bico = p
      ENDIF
      if (j.gt.i) bico = 0.0
c     Will fail if j<0 or j>i
      END
```

5.2 Hyper: Find the Outcomes of Draws Without Replacement

Hyper uses the hypergeometric distribution to find the number of successes in trials without replacement. Traditionally, this is illustrated by the problem of randomly drawing balls from an urn. The program uses input parameters analogous to this situation: number of red balls, number of draws, and total number of balls. Hyper computes the probability for each number of successes, or number of red balls drawn.

Two versions of the code are included, the first with hyper as a program, and the second with hyper as a subroutine.

Sample Problems. Figure 5.2 shows the probability of each outcome for the following two problems.

1. In the 7-card poker game, baseball, 3's and 9's are wild. What's the chance of drawing 0-7 wild cards?

2. Intelligence tells you the enemy is moving forces to 2 of 4 key military targets, but you don't know which of the 4 will be attacked. Further, you only have forces to defend 2 of the 4 targets. What is the chance you'll defend 0,1, or 2 of the targets that will actually be attacked?

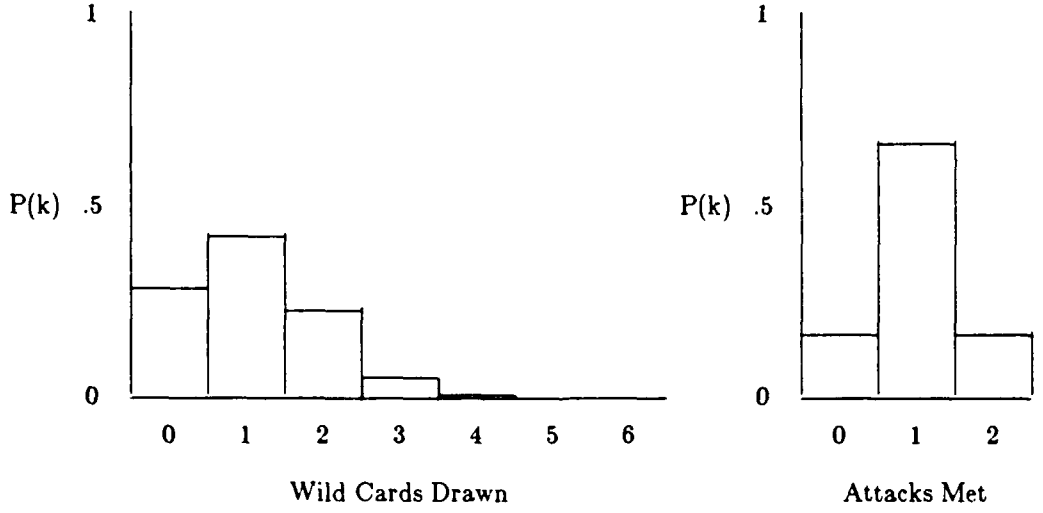


Figure 5.2 Probability of k Occurrences Without Replacement.

Input/Output. Execute the Hyper program and it will ask you for the number of red balls, the number od draws, and the number of balls. For the first sample problem, enter 8, 7, 52, since there are 8 wild cards, 7 cards drawn, and 52 cards in the deck. Results are shown in figure 5.2. For the second sample problem, enter 2, 2, 4, since two will be attacked, you can protect 2 targets, and there are 4 targets. Again, results are shown in figure 5.2. To escape the program, type ^C or ^D.

Mathematics.

Given:

i = number of red balls,
j = number of draws,
k = number of successes,
n = number of balls, and

Find P_k, the probability of exactly k successes.

$$P_k = \frac{\binom{i}{k}\binom{n-i}{j-k}}{\binom{n}{j}}$$

Code

```
      PROGRAM HYPER
c     Hyper: Find the probability of outcomes without replacement.
c     i = #of desired objects in the set from which you draw.
c     j = #of draws w/o replacement
c     m = maximum number that can be drawn
c     n = number of objects in the set from which you draw.
1     format (' Probability of',i3,' successes is',f6.3)

      print *,'What is #of red balls,  #of draws, #of balls?'
10    CONTINUE
       read *,i,j,n
       m=min0(i,j)
       DO 20 k=0,m
        a1 = bico(i,k)
        a2 = bico(n-i,j-k)
        a3 = bico(n,j)
        p=a1*a2/a3
        print 1, k, p
20     CONTINUE
      GOTO 10
      END

      FUNCTION BICO (i,j)
c 1   Bico: find binomial coefficient.
      IF (j.eq.0 .or. j.eq.i) THEN
       bico = 1
      ELSE
       k = min0(j,i-j)
       p = i-k+1
       DO 20 m=2,k
        p = p*(i-k+m)/float(m)
20     CONTINUE
       bico = p
      ENDIF
      if (j.gt.i) bico = 0.0
c     Will fail if j<0 or j>i
      END

      SUBROUTINE HYPER(i,j,n)
c     Hyper: find the outcomes of draws without replacement.
c     i = #of desired objects in the set from which you draw.
c     j = #of draws w/o replacement
c     m = maximum number that can be drawn
c     n = number of objects in the set from which you draw.
1     format (' Probability of',i3,' successes is',f6.3)
c
      m=min0(i,j)
      PRINT*,'HYPER: i,j,m,n=',i,j,m,n
      DO 20 k=0,m
       a1 = bico(i,k)
       a2 = bico(n-i,j-k)
       a3 = bico(n,j)
       p=a1*a2/a3
       print 1, k, p
20    CONTINUE
      END
c
      FUNCTION BICO (i,j)
c 1   Bico: find binomial coefficient.
      IF (j.eq.0 .or. j.eq.i) THEN
       bico = 1
      ELSE
       k = min0(j,i-j)
       p = i-k+1
       DO 20 m=2,k
        p = p*(i-k+m)/float(m)
20     CONTINUE
       bico = p
      ENDIF
      if (j.gt.i) bico = 0.0
      Will fail if j<0 or j>i
      END
```

5.3 Ndtr, Fnd: Integrate the Normal Distribution.

Ndtr and Fnd are two common routines that find the integral under the standard normal density function from $-\infty$ to x using an approximation technique adapted from the *ndtr* routine in the IBM Scientific Subroutine Package[6]. A simple transformation of the argument makes it usable for normal density functions with $\mu \neq 0$ and $\sigma \neq 1$. The approximation technique is discussed and a sample run is shown.

The standard normal density function, illustrated in Figure 5.3, is defined by the following equation:

$$p = \frac{1}{\sqrt{2\pi}} \int e^{-x^2/2} dx$$

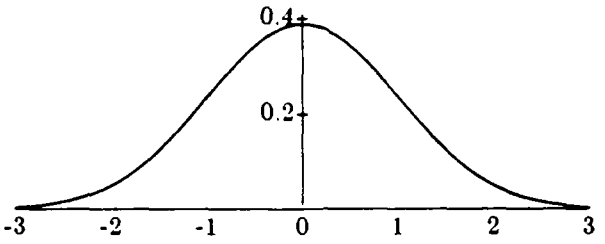


Figure 5.3 The Area Under the Standard Normal Density Function

Input/Output. These functions find the probability that a normally distributed number is less than x. Thus, the user passes a real argument x and either routine produces an output between 0 and 1. For the standard normal density function, $\mu = 0$ and $\sigma = 1$. If you wish to find the area under a normal density function with any other mean and standard deviation, just use $(x-\mu)/\sigma$ instead of x for the argument.

Mathematics. Ndtr uses this approximation by Hastings[7]. (Because of a change of variable, all Hastings constants must be multiplied by $\sqrt{2}$). Various authors differ on the maximum error: Hastings says it is 1.5E-7, the Handbook of Mathematical Functions[8] says it is 7.5E-8, the IBM Scientific Subroutine Package says it is 7E-7.

$$P(x) = 1 - Z(x)(b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5)$$

Where:

$$t = 1/(1+px)$$

$$Z(x) = e^{-x^2/2}/\sqrt{2\pi}$$

We have added a line of code truncating the argument at ± 6 to avoid overflow on 32 bit single precision computers. Values of x outside this range yield probabilities sufficiently close to 1 or 0 to be ignored in most practical problems.

Fnd uses this polynomial approximation by Hastings[9]:

$$P(x) = 1 - \frac{1}{\left[1 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6\right]^{16}},$$

6. System/360 Scientific Subroutine Package (360A-CM-03X) Version III Programmer's Manual, H20-0205-3, IBM Data Processing Division, 112 East Post Road, White Plains, NY 10601, undated, p78.
7. Approximations for Digital Computers, C. Hastings, Jr., Princeton Univ. Press, Princeton, N.J., 1955, p 169.
8. Handbook of Mathematical Functions, Abramowitz, M., and Stegun, Irene A., Dover Publications, Inc., NY, p 932, eq 26.2.17.
9. ibid, p. 187.

where each a_n is a coefficient given by Hastings, multiplied by $(\sqrt{2})^n$. This factor of $\sqrt{2}$ for each x is due to a change of variable which was used to convert the given form to the standard normal density function.

Hastings reports the maximum error in his approximation as 3.0E-7. We added a line in the code that truncates the argument at ± 5 to avoid overflow. Values of x beyond this range yield probabilities sufficiently close to 0 or 1 to be assigned these values in most practical problems.

Code

```
      REAL FUNCTION NDTR (x)
c     Ndtr: Integrate the Normal Distribution from -infinity to x.
c     Source: Adapted from ndtr in the IBM SSP pg78.

      ax = abs(x)
      t = 1.0/(1.0+.2316419*ax)
      d = 0.0
      if(ax.lt.6.0)d = 0.3989423*exp(-x*x/2.0)
      p = 1.0 - d*t*((((1.330274*t - 1.821256)*t + 1.781478)*t -
     * 0.3565638)*t + 0.3193815)
      if (x.lt.0.0) p = 1.-p
      ndtr = p
      END

      REAL FUNCTION FND(x)
c     Fnd: Integrate the Normal Distribution from -infinity to x.
c     from hastings approximations for digital computers

      f = 0.
      ax = abs(x)
      if(ax .ge. 5.) go to 10
      f = (((((.5383e-5*ax+.488906e-4)*ax+.380036e-4)*ax+.0032776263)*ax
     1 +.0211410061)*ax + 0498673469)*ax+1.0
      f = .5/((f**8)**2)
10    if(x .ge. 0.) f = 1.-f
      fnd = f
      END
```

5.4 Fnd2d: Integrate the Normal Distribution Under a Directed Line Segment.

Fnd2d finds the area under the bivariate normal between the limits x_a and x_b in the x direction, and between $-\infty$ and $y_b = c+dx$ in the y direction. Figure 5.4 illustrates the limits. The solution is useful in for finding the hit probability on one or more triangles, hence on polygons.

This subroutine, originally named fsubij, was developed by Art Groves[10]. We have made cosmetic changes and substituted a call to qsimp, a cleaner, more efficient simpson integration routine. See Numerical Recipes for qsimp.

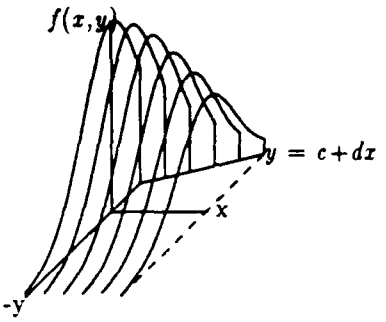


Figure 5.4 The Limits of Integration

Input/Output. When you call fnd2d, pass eight values in the argument list. The first and second arguments are the x and y coordinates of the starting point of the directed line segment under which you want to integrate. The next two arguments are the coordinates of its endpoint. The last four arguments give the means and standard deviations of the bivariate normal density function, in this order: $\bar{x}$, y, σ_x, σ_y.

Mathematics.

$$p = \frac{1}{2\pi\sqrt{\sigma_x\sigma_y}} \int_{-\infty}^{y=c+dx} \int_{x_a}^{x_b} e^{((x-\mu_x)^2/2\sigma_x^2)+((y-\mu_y)^2/2\sigma_y^2)} dx\ dy$$

10. The Probability of Hitting a Polygonal Target, Arthur Groves, U.S. Army Materiel Systems Analysis Activity, Aberdeen Proving Ground, MD, Technical Report No. 330, April 1981.

Code.

```
      FUNCTION FND2D (xi,yi,xj,yj,xbar,ybar,sigmax,sigmay)
c     Fnd2d: Integrate the bivariate normal density function 'under' the
c     directed line segment from (xi,yi) to  (xy,yj).
      common /fij/ a,b,c
      external fun

      fnd2d=0.
      IF (xi.ne.xj) THEN
       xmin=amin1(xi,xj)
       xmax=amax1(xi,xj)
       smin=amax1((xmin-xbar)/sigmax,-3.5)
       smax=amin1((xmax-xbar)/sigmax,3.5)
       IF (smin.lt.smax) THEN
        a=(yi-ybar)/sigmay
        c=(xi-xbar)/sigmax
        IF (yi.eq.yj) THEN
         fnd2d=fnd(a)*(fnd((xj-xbar)/sigmax)-fnd(c))
        ELSE
5         b=sigmax*(yj-yi)/(sigmay*(xj-xi))
          call qsimp(fun,smin,smax,fnd2d)
          if(xj.lt.xi)fnd2d=-fnd2d
        ENDIF
       ENDIF
      ENDIF
      END

      FUNCTION FUN (s)
      common /fij/ a,b,c
      fun=.3989422804*exp(-.5*s*s)*fnd(a+b*(s-c))
      END

      include 'qsimp.f'
      include 'fnd.f'
```

5.5 Confb: Find Confidence Intervals on a Binomial Outcome.

Construct confidence intervals on an outcome with a binomial distribution. Enter the sample size and either the probability of occurrence or the number of occurrences. The confb function gives 90%, 95%, and 99% confidence intervals.

Input/Output. An input line consists of an integer and a floating point number separated by a comma or blank. The integer is the sample size. If the floating point number is less than one, it is taken to be the probability of occurrence; otherwise, it is taken to be the number of occurrences. If you do not designate an input file on the command line, confb will prompt for each input line. Hit ^D to quit. The output appears as a line of a table with the headings, N, PROB, 90%, 95%, and 99%.

Sample Problem. Say you run Tank Wars for 50 repetitions. You find that the Blue side wins 10% or 20% or 50% or 80% or 90% of the confrontations. What are the 95% confidence intervals on these binomial outcomes? The confb output for this case follows. See Figure 5.5 for a graphical representation of the results.

N	PROB	--- 90% ---	--- 95% ---	--- 99% ---
50	0.100	0.037-0.198	0.031-0.220	0.022-0.265
50	0.200	0.112-0.315	0.101-0.338	0.083-0.383
50	0.500	0.377-0.623	0.357-0.643	0.320-0.680
50	0.800	0.685-0.888	0.662-0.899	0.617-0.917
50	0.900	0.802-0.963	0.780-0.969	0.735-0.978

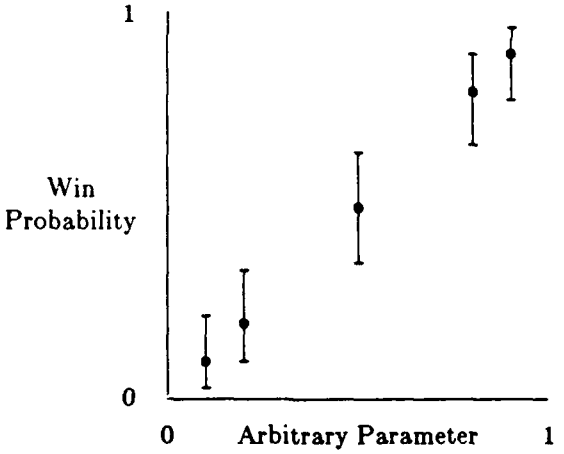


Figure 5.5 Tank Wars Probabilities With 95% Confidence Intervals

Mathematics. The confb routine uses the method of Dixon and Massey[11], to calculate the confidence intervals as follows:

11. *Introduction to Statistical Analysis*, 3rd edition, Dixon, Wilfred J., and Massey, Frank J., McGraw Hill, New York, 1969, p.246.

$$a = 0.5/n$$
$$b = (p+a)(1-p-a)/n$$
$$c = (p-a)(1-p+a)/n$$
$$d = n/(n+z_i^2)$$
$$e = az_i^2$$
$$f = a^2 z_i^2$$
$$L_i = d(p-a+e-z_i\sqrt{b+f})$$
$$H_i = d(p+a+e+z_i\sqrt{c+f})$$

where

n is the sample size

p is the sample mean (the experimentally determined probability)

$z_i = 1.645, 1.960, 2.576$ for 90, 95, and 99% intervals, respectively

L_i is the lower bound of the 90, 95, or 99% confidence interval

H_i is the upper bound of the 90, 95, or 99% confidence interval

Code.

```
c       CONFB: Find confidence intervals on a binomial probability.
c       First value read is sample size. Second value read is taken
c         to be number of successes if > 1 and probability of success
c         if < 1.
        real hi(3), lo(3)
        logical fail
1       format(i5,f10.3)
2       format(' Sample too small. N*prob < 5 or N*(1-prob) < 5.')
3       format(8x,i6,f8.3,3(2x,f5.3,'-'f5.3))
5       format(12x,'N    PROB  --- 90% --- '
     *  '--- 95% ---  --- 99% ---')

        write(6,5)
10      CONTINUE
          read (5,1,END=99) n,p
          if (p.gt.1.0) p = p/n
          call confb (p,n,hi,lo,fail)
          if (fail)write(6,2)
          if (.not.fail) write(6,3) n,p,(lo(i),hi(i),i=1,3)
        GOTO 10
99      CONTINUE
        END

        SUBROUTINE CONFB (p, nr, hi, lo, fail)
c       Confb: Find the 90, 95, and 99% confidence intervals
c         on a probability p, where p is the estimated probability
c         and nr is the sample size.
c       Reference: Introduction to Statistical Analysis, 3rd edition,
c         Dixon and Massey, p248.
        real hi(3), lo(3), z(3), n
        logical fail
        data z /1.645, 1.960, 2.576/

        n = float(nr)
        fail = (n*p.lt.5) .or. ((n-n*p).lt.5)
        IF (.not.fail) THEN
c       Method is applicable, so apply it
          DO 20 i=1,3
            s1 = n/(n+z(i)**2)
            s2 = 0.5*z(i)**2/n
            s3 = (p+0.5/n) * (1.0-p-0.5/n)
            s4 = (p-0.5/n) * (1.0-p+0.5/n)
            s5 = z(i)**2/(4.0*n*n)
            lo(i) = s1*(p-0.5/n+s2-z(i)*sqrt(s3/n+s5))
            hi(i) = s1*(p+0.5/n+s2+z(i)*sqrt(s4/n+s5))
20        CONTINUE
        ENDIF
        END
```

5.6 Confn: Find Confidence Intervals on a Normal Outcome.

Confn constructs confidence intervals for the mean of a normal population. Enter the mean, an estimate of the standard deviation, and the sample size. The program gives 90%, 95%, and 99% confidence intervals.

Input. The program reads from standard input using free format. A line of input consists of the following in order: the mean of the sample, the estimate of the standard deviation, and the sample size. There is no limit to the number of input lines. Use the -i option if you want the program to prompt for each input line. Hit ^D to quit.

Sample Problem. Say that we fire a gun 20 times and find that the mean and standard deviation in the horizontal direction are 0.10 and 1.17 meters, and in the vertical direction they are 0.44 and 0.74 meters. What are the horizontal and vertical 95% confidence intervals on these measurements?

Following is the conf dialog used to solve this problem, and Figure 5.6 gives a graphical representation of the results. Only the horizontal bar crosses the axis, indicating that the gun has a vertical, but not a horizontal, bias. Since we simulated these shots, we know that the true mean is (0.2 m, 0.3 m). As you would expect, this point falls within the horizontal and vertical confidence intervals.

```
confn -i
What is sample: mean, S.D., size?
Sample   Sample   Sample        ---90%---        ---95%---        ---99%---
 Mean     S.D.     Size          Interval         Interval         Interval
0.10,1.17,20
  0.10     1.17     20        ( -0.35,  0.55)  ( -0.45,  0.65)  ( -0.65,  0.85)
0.44,0.74,20
  0.44     0.74     20        (  0.15,  0.73)  (  0.09,  0.79)  ( -0.03,  0.91)
^D
```

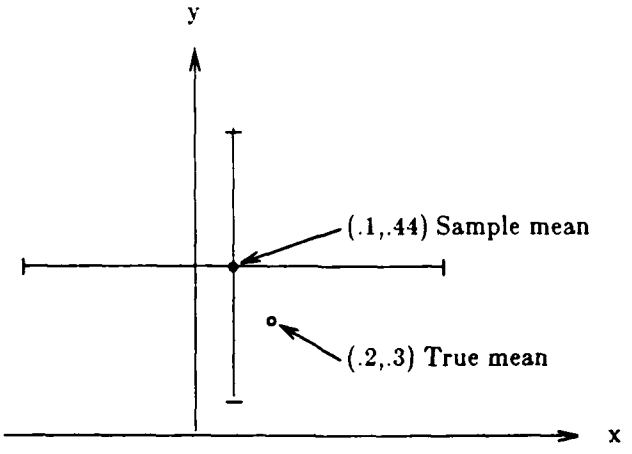


Figure 5.6 95% Confidence Intervals on Sample Data

Mathematics. The method used by this program comes from Mood and Graybill.[12] They give the upper and lower bounds of the 90% confidence interval as follows:

$$\bar{x} \pm t_{.05}\sqrt{[\sum(x_i - \bar{x})^2/(n^2 - n)]},$$

where: $\bar{x}$ is the sample mean
n is the sample size
$t_{.05}$ is the 5% level of t

The 5% level of t refers to the student's t distribution, the subscript denoting the percentage of the area under the curve that the desired point will cut off from each end. Values for other subscripts, or levels of t, are used to determine other confidence intervals. Besides 90%, this program finds 95% and 99% confidence intervals, using $t_{.025}$ and $t_{.005}$, respectively. These values vary with sample size and are retrieved from a table that is stored in DATA statements, either directly or by interpolating between sample sizes. In the above equation, the square root term is almost identical to the expression for the standard deviation, the difference being a factor of $1/\sqrt{n}$. Thus, the user inputs σ and n are sufficient for confn tc calculate the three confidence intervals.

Code.

```
c       CONFN: Find the 90, 95, 99% confidence
c       intervals on a normal distribution, given
c       sample mean, sample S.D., sample size
        real d(6)
1       format(f9.2,f8.2,i6,3x,3(' (',f6.2,',',f6.2,')'))
2       format('   Sample  Sample  Sample ',
     1  '  ---90%---      ---95%---      ---99%---')
3       format('    Mean     S.D.    Size  ',
     1  '    Interval       Interval       Interval')

        print *,'What is sample: mean, S.D., size?'
        print 2
        print 3
10      CONTINUE
         read(5,*,end=99) xbar, s, n
         factor = s/sqrt(float(n))
         call tstats(n-1,t1,t2,t3)
         d(1) = xbar - t1*factor
         d(2) = xbar + t1*factor
         d(3) = xbar - t2*factor
         d(4) = xbar + t2*factor
         d(5) = xbar - t3*factor
         d(6) = xbar + t3*factor
         print 1, xbar, s, n, d
        GOTO 10
99      CONTINUE
        END

        SUBROUTINE TSTATS (df,t1,t2,t3)
c       Tstats: Find Student's T-statistic
c       for 95%, 97.5%, 99.5%
        integer df, dfs(34)
        real t950(34), t975(34), t995(34)
        fun(a,b,c) = a + (b-a)*c
        data dfs  / 1, 2, 3, 4, 5,
     1              6, 7, 8, 9,10,
     2             11,12,13,14,15,
     3             16,17,18,19,20,
     4             21,22,23,24,25,
     5             26,27,28,29,30,
     6             40,60,120,9999999/
        data t950 / 6.314, 2.920, 2.353, 2.132, 2.015,
     1              1.943, 1.995, 1.860, 1.833, 1.812,
     2              1.796, 1.782, 1.771, 1.761, 1.753,
     3              1.746, 1.740, 1.734, 1.729, 1.725,
     4              1.721, 1.717, 1.714, 1.711, 1.708,
     5              1.706, 1.703, 1.701, 1.699, 1.697,
     6              1.684, 1.671, 1.658, 1.645/
        data t975 /12.706, 4.303, 3.182, 2.776, 2.571,
     1              2.447, 2.365, 2.306, 2.262, 2.228,
     2              2.201, 2.179, 2.160, 2.145, 2.131,
     3              2.120, 2.110, 2.101, 2.093, 2.086,
     4              2.080, 2.074, 2.069, 2.064, 2.060,
     5              2.056, 2.052, 2.048, 2.045, 2.042,
     6              2.021, 2.000, 1.980, 1.960/
        data t995 /63.657, 9.925, 5.841, 4.604, 4.032,
     1              3.707, 3.499, 3.355, 3.250, 3.169,
     2              3.106, 3.055, 3.012, 2.977, 2.947,
     3              2.921, 2.898, 2.878, 2.861, 2.845,
     4              2.831, 2.819, 2.807, 2.797, 2.787,
     5              2.779, 2.771, 2.763, 2.756, 2.750,
     6              2.704, 2.660, 2.617, 2.576/

        IF (df.le.0) THEN
         print *,'TSTATS: Degrees of freedom must be > 0.'
         STOP
        ELSEIF (df.le.30) THEN
         t1 = t950(df)
         t2 = t975(df)
         t3 = t995(df)
        ELSE
         j = 30
         if (df.gt.40) j = 31
         if (df.gt.60) j = 32
         if (df.gt.120) j = 33
         factor = float (df-dfs(j+1))*dfs(j) /
     1            float((dfs(j)-dfs(j+1))*df)
         t1 = fun(t950(j+1),t950(j),factor)
         t2 = fun(t975(j+1),t975(j),factor)
         t3 = fun(t995(j+1),t995(j),factor)
        ENDIF
        END
```

12. *Introduction to the Theory of Statistics*, Alexander M. Mood and Franklin A. Graybill, McGraw-Hill, 1963, New York, p.252.

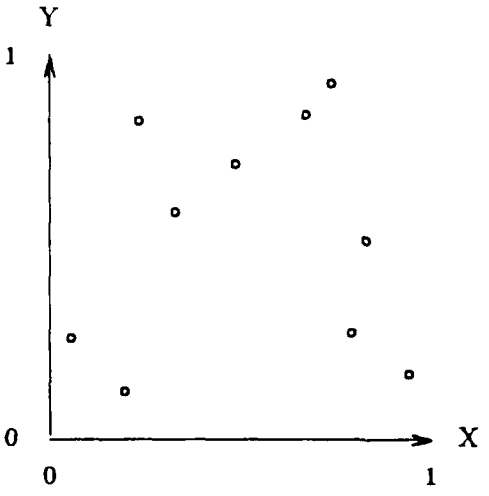
5.7 Ranu: Draw a Random Number

This subroutine uses a version of the uran31 uniform random number generator to pseudo-randomly draw a number from the uniform distribution extending from 0 to 1. The following explains how to "seed" the generator and shows some sample draws.

Why use a random number generator coded in Fortran? For the following reasons:

1. First, we believe this is one of the better random number generators*.
2. If you are transporting a program from one computer to another, and it draws random numbers, you'll be more confident if test cases generate exactly the same results on each machine.
3. If a long run dies in mid-stream, and you've printed the random number seed periodically, you may be able to restart the run at the point it last printed the seed.
4. And finally, if you are debugging a run by turning on more and more print statements, you can suppress enormous volumes of printout by judiciously setting the random number seed and restarting the run in mid-stream.

Input/Output. Ranu requires the calling program to initialize the variable j in the common statement /crandm/ j. We have used the value j=1111111, however other odd integers are legal. The common statement may be replaced with a data statement such as: data j /1111111/ if you do not wish to reset the seed. The calling statement: call ranu(), of course, requires no argument. Figure 5.7, below illustrates ten draws using ranu, and shows a plot of 20 draws using pairs of variates as the coordinates of the 10 points.



Seed= 1111111
First ten draws from the function:
0.7401378
0.9308131
0.7908101
0.2813551
0.2351466
0.8330226
0.1955611
0.1284252
0.3288300
0.5939014

Figure 5.7 Ten Pairs of Numbers Drawn from a Uniform Distribution

Mathematics. Ranu* is a variant of the uran31 subroutine long used at BRL. We have tested a number of random number generators and found Ranu to be the only one to pass all 5 tests. Ranu & uran31 will work on any computer with 31 or more bits per integer. Any odd seed between 1 and 67108863 was acceptable for the earliest version. Revisions to accommodate 31 bit machines will have reduced this upper limit and the cycle length. Cycle length of the current version is 16,777,215.

Code.

```
      FUNCTION RANU (dm)
c     Ranu: A version of uran31 uniform random nr generator.
      common /crandm/ j
      real a1

      j=j*25
      j=j-(j/67108864)*67108864
      j=j*25
      j=j-(j/67108864)*67108864
      j=j*5
      j=j-(j/67108864)*67108864
      a1=j
      ranu= a1/67108864
      END
```

12. We recommend the discussion of random number generators in *Numerical Recipes*, by Press, Flannery, Teukolsky, & Vetterling, pages 191-199. See also, *Seminumerical Algorithms, The Art of Computer Programming*, by Donald Knuth, pages 1-190.

* For further information on this pseudo-random number generator, see: *Collected Algorithms from CACM*, volume II, Algorithm 266, Psuedo-Random Numbers, page 266-P 1- 0, by M.C. Pike and I.D. Hill. Also see: Remark on Algorithm 266, Psuedo-Random Numbers, page 266-P 2- R1.

5.8 Rann: Draw From a Normal Distribution.

Rann draws two random variates from the standard normal distribution using the Box-Muller method.

Output. This subroutine generates two real numbers randomly chosen from the normal distribution. Interpreting the output as x and y coordinates, we produced the twenty points drawn in Figure 5.8, which represent random shots.

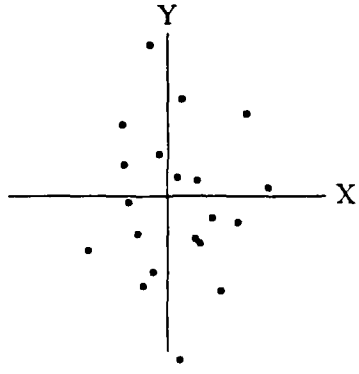


Figure 5.8 Draw of 20 Random Shots

Mathematics. Rann is based on CACM algorithm 267 by James R. Bell, called "Random Normal Deviate" published by the Association for Computing Machinery, Inc. (ACM),[13] that in turn is based on the method developed by Box and Muller in 1958. It produces two independent random variables, each from the normal distribution with mean 0 and standard deviation 1. The subroutine calls the real function ran twice. Ranu is a pseudo-random number generator that produces a number lying strictly between 0 and 1. See section 5.7 for details. Algorithm 334 is a slightly faster, but more complex version of algorithm 267. See also algorithm 442 for a slower, but higher precision algorithm. Finally, see algorithm 448, which may be faster if one of the 2 random deviates must be discarded.

Code.

```
      SUBROUTINE RANN(p,q)
c 7   Rann: draw two random numbers from the std normal distribution.
c     Box-Muller method

      x = sqrt(-2.*alog(ranu(dm)))
      y = 2.*3.1415926535*ranu(dm)
      p = x*cos(y)
      q = x*sin(y)
      END
```

13. *Collected Algorithms From ACM*, Volume II, 1980, Algorithm #267, Association for Computing Machinery, New York.

INTENTIONALLY LEFT BLANK.

6. Hit Probability

6.1 Box: Find the Probability of Hitting a Rectangle.

Box assumes the shots fall in an uncorrelated, bivariate normal distribution on a rectangular box.

Input. For input use any consistent set of units; feet, meters, whatever. You can quit at any time by typing ^D.

Box asks for the left and right edges of the target and then the lower and upper edges of the target. Then it asks for the horizontal and vertical standard deviations of the shot pattern. With this information, it finds the hit probability on the rectangle.

After completing this, it loops back and asks for further shot pattern sizes. If you are not interested in further shot patterns at this target, just enter a pair of zeros. It then gives you a chance to change the target size and repeat the process.

Sample Dialog. Below is a sample dialog. In the first case, the target is a square extending 1 unit left and right, below and above the center of the shot pattern and the shot pattern is circular with a standard deviation of one unit. The probability of hitting the target is 46.6%. In the second case, the size of the shot pattern has doubled in each direction; now the hit probability is 14.7%. We are no longer interested in this case, so we enter 0.,0. for the dimensions of the next shot pattern and Box asks for new target dimensions. This time, the rectangle extends 1 unit left and 2 units right, and 2 units above and below the aim point. Again, the standard deviation of the shot pattern is 2 units in each direction. In this third case, the hit probability is 36.4%.

```
 If you make xsig=0, I will ask for new tgt.
 What is xlow, xhigh?
-1. 1.
 What is ylow, yhigh?
-1. 1.
 What is xsig, ysig?
 1. 1.
 p =  0.4660650
 What is xsig, ysig?
 2. 2.
 p =  0.1466315
 What is xsig, ysig?
^C
```

Mathematics. The probability (P_h) of hitting a rectangular target is:

$$P_h = \frac{1}{2\pi} \int_{x_l/x_\sigma}^{x_h/x_\sigma} e^{-x^2/2} dx \int_{y_l/y_\sigma}^{y_h/y_\sigma} e^{-y^2/2} dy$$

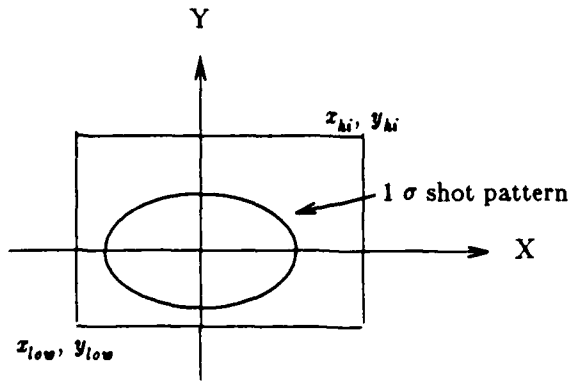


Figure 6.1 Box

Code.

```
      PROGRAM BOX
c     Box: Find the probability of hitting a rectangular box.
      real ndtr

20    CONTINUE
       print *, ' If you make xsig=0, I will ask for new tgt.'
       print *, ' What is xlow, xhigh?'
       read (5,*,end=99,err=99)  xl, xh
       print *, ' What is ylow, yhigh?'
       read (5,*,end=99,err=99)  yl, yh
30     CONTINUE
        print *, ' What is xsig, ysig?'
        read (5,*,end=99,err=99)  xs, ys
        IF (xs.lt..005) GOTO 20
c       Find phit
         p1 = ndtr(xh/xs)
         p2 = ndtr(xl/xs)
         px = p1-p2
         p3 = ndtr(yh/ys)
         p4 = ndtr(yl/ys)
         py = p3-p4
         p  = px*py
         print *,' p =', p
       GOTO 30
99    CONTINUE
      END
      include 'ndtr.f'
```

6.2 Circle: Find the Probability of Hitting a Circle.

Circle finds the probability of a circular shot pattern with no aim bias hitting a circular target. The program prompts for the radius of the circle and the total linear dispersion of the round. Then it calculates the hit probability using the following equation:

$$p = 1 - e^{-r^2/(2\sigma^2)}$$

Input/Output. This program is interactive. It simply prompts for the radius of the circular target and the linear dispersion of the round. These may be entered in any consistent set of units, as floating point values separated by a comma or blank. Circle returns the value for probability of hit. Enter zeros to quit.

Sample Problem. Suppose the target is a circle with a 1 meter radius. The dispersion of your rounds is 1 meter at 100 meters range, 2 meters at 200 meters range, ..., 5 meters at 500 meters range. To analyze these data with the circle routine, respond to the prompt with the following pairs of numbers, and you will get the following answers:

radius, dispersion	phit
1.,1.	.393
1.,2.	.118
1.,3.	.054
1.,4.	.031
1.,5.	.020

From this information, you can plot hit probability as a function of range. Figure 6.2 contains such a plot for this sample data.

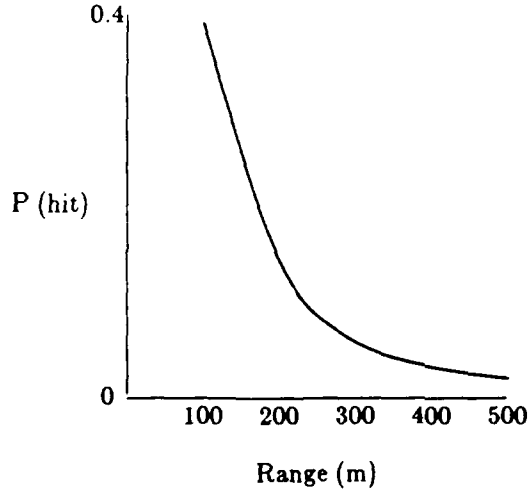


Figure 6.2 Plot of Circle Output Data

Mathematics. The mathematical basis of the circle program is straightforward*. We assume a normally distributed shot pattern. The probability density function in one dimension is:

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma_x} e^{-((x-\mu_x)^2/2\sigma_x^2)} \quad (\textit{and similarly for } f(y)),$$

where:

μ_x is the mean, and

* The source for the following derivation is DARCOM Pamphlet No. 706-101, *Army Weapon Systems Analysis, Part One*, November 1977, pp. 13-6 - 13-11.

σ_x is the linear dispersion.

Now, the dispersion we observe in firing weapons is bivariate. If x and y are statistically independent, the product $f(x)f(y)$ gives the bivariate probability density function $f(x,y)$. Assuming no aim bias ($\mu_x = \mu_y = 0$) and a circular dispersion ($\sigma_x = \sigma_y$), we get this equation:

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-((x^2+y^2)/2\sigma^2)}.$$

To obtain the probability of hitting a circular target of radius r with a weapon having the shot pattern described above, integrate $f(x,y)$ over x and y for $x^2 + y^2 \leq r^2$. Converting to polar coordinates simplifies the integral to:

$$p = \frac{1}{2\pi\sigma^2} \int_0^r \int_0^{2\pi} e^{-R^2/(2\sigma^2)} R dR d\theta.$$

This can be evaluated to yield the hit probability for a circle, given the radius r and dispersion σ, as:

$$p = 1 - e^{-r^2/(2\sigma^2)}.$$

Code.

```
c       CIRCLE: Find the probability of hitting a circular tgt.
1       format (' Phit=',f6.3)

20      print *, ' What is radius, dispersion (m)?'
        read *, r, s
        IF (r.eq.0) STOP
         p = 1.-exp(-0.5*r**2/s**2)
         print 1, p
        GOTO 20
        END
```

6.3 Polygon: Find the Probability of Hitting a Polygon.

Polygon computes the probability of hitting a target of arbitrary shape. First, approximate the boundaries of the target by straight line segments. Then, input the number of vertices, the coordinates of each, and the aim point and linear dispersion. The program assumes the shot pattern is normally distributed.

Input. An input file consists of the following lines of data:

Number of straight line segments ($n \leq 100$)
Coordinates of the first corner

.
.
.

Coordinates of the nth corner
Coordinates of aim point, linear x and y dispersions

The corners must traverse the figure in clockwise order, and commas must separate all elements on the same line.

Sample Problem. Suppose you have used the center routine to determine the centroid, or aim point, of an irregularly shaped target. Now you wish to find the probability of hitting that target. For our sample application let's use the same polygon as Figure 4.1 and take as the aim point the centroid found by center. With a linear dispersion of 5 units in each direction, the program calculates a hit probability of 0.5664. Figure 6.3 shows the output for this case as well as a diagram of the figure with a one-sigma ellipse (a circle here because $\sigma_x = \sigma_y$) around the aim point.

```
Output:
 number of vertices =     7
 coordinates of vertices (horiz, vert)
    0.000     0.000
   -5.000     5.000
   -1.000    10.000
    8.000    10.000
   15.000     8.000
   10.000     4.000
   10.000     0.000

 mean (horiz,vert) and sigma (horiz,vert)
    4.316     5.247     5.000     5.000

   hit probability =   0.5664
```

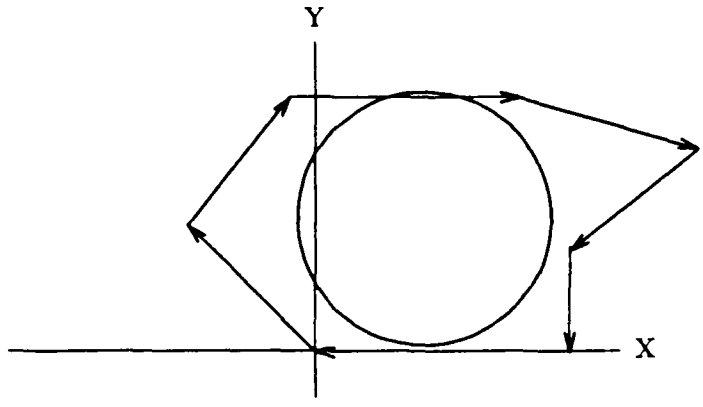


Figure 6.3 Hit Probability on an Irregular Target

Mathematics. Polygon assumes the shot pattern is normally distributed. It uses the bivariate normal density function with the aim point coordinates as its means and the linear dispersions as its standard deviations. The program integrates this function under each directed line segment, using Simpson's rule and the trapezoidal rule.

Code.

```
c       POLYGON: Find prob of hitting a polygon.
        dimension x(100),y(100)
1       format(i10)
2       format(2f10.0)
3       format(' ',2f10.3)
4       format(' ',4f10.3)
5       format(4f10.0)
6       format(' number of vertices = ',i5,
     1  /' coordinates of vertices (horiz, vert) ')
7       format(/' mean (horiz,vert) and sigma (horiz,vert) ')
8       format(/'    hit probability = ',f8.4)

        read(5,1)n
        write(6,6)n
        read(5,2)(x(i),y(i),i=1,n)
        write(6,3)(x(i),y(i),i=1,n)
        read(5,5)xbar,ybar,sigmax,sigmay
        write(6,7)
        write(6,4)xbar,ybar,sigmax,sigmay
        p=phit(n,x,y,xbar,ybar,sigmax,sigmay)
        write(6,8)p
        END

        FUNCTION PHIT(n,x,y,xbar,ybar,sigmax,sigmay)
c       Phit: Compute the probability of hitting an
c       n-sided polygonal target. The coordinates of the vertices
c       of the polygon are stored in the x and y arrays in
c       consecutive clockwise order around the polygon. The
c       delivery error distribution is assumed to be bivariate
c       normal with mean (xbar,ybar) and with standard deviations
c       sigmax and sigmay respectively.
        dimension x(1),y(1)
1       format(' both sigmax and sigmay must be greater than zero'/
     * ' sigmax = ',f10.3,'    sigmay = ',f10.3)

        IF (sigmax.gt.0. .and. sigmay.gt.0.) THEN
          phit=0.
          DO 10 i=1,n
            j=mod(i,n)+1
            f=fsubij(x(i),y(i),x(j),y(j),xbar,ybar,sigmax,sigmay)
            phit=phit+f
10        CONTINUE
        ELSE
          write(6,1) sigmax,sigmay
          STOP
        ENDIF
        END
```

```
FUNCTION FUN (s)
common /fij/ a,b,c
fun=.3989422804*exp(-.5*s*s)*fnd(a+b*(s-c))
END
include 'qsimp.f'
include 'fnd2d.f'
include 'fnd.f'
```

6.4 Solid: Find the Probability of Hitting an Irregular Solid

Use solid to determine the probability of hitting a three-dimensional target of arbitrary shape. This program assumes a shot pattern that is normally distributed horizontally and vertically. It finds the probabilities of hitting polygonal faces and sums them.

Input/Output. An input file for solid contains the following lines:

σ_x, σ_y	linear dispersions
lev	output options
v	velocity of projectile
p	position of projectile
n	number of corners
x_1, y_1, z_1	coordinates of first corner
	.
	.
x_n, y_n, z_n	coordinates of nth corner
m	number of faces
$k_{1,1}, k_{1,2}, \ldots, k_{1,6}$	corners clockwise around face 1
	.
	.
$k_{m,1}, k_{m,2}, \ldots, k_{m,6}$	corners clockwise around face m

The value for *lev* determines the output. Input a "1" if you want the hit probability only. Input a "2" if you want solid to echo the input as well. Input a "3" and the program will also print the new projectile and corner positions after rotation.

Mathematics. This program finds the probabilities of hitting polygonal faces and sums them. The faces can be one or more planar targets, or they can be the sides of one or more solid targets. The program first rotates and translates the target and the projectile to the projectile base coordinates.

Figure 6.4 shows a box shaped target before and after transforming the coordinates of the target's corners. The left box is the target before transformation. It is then rotated 20 degrees counterclockwise around the Z axis, rotated 5 degrees counterclockwise around the X axis, and translated. The right box is the target after transformation. The projectile (not shown) undergoes a similar transformation so that it moves to the origin and is travelling up the Y axis.

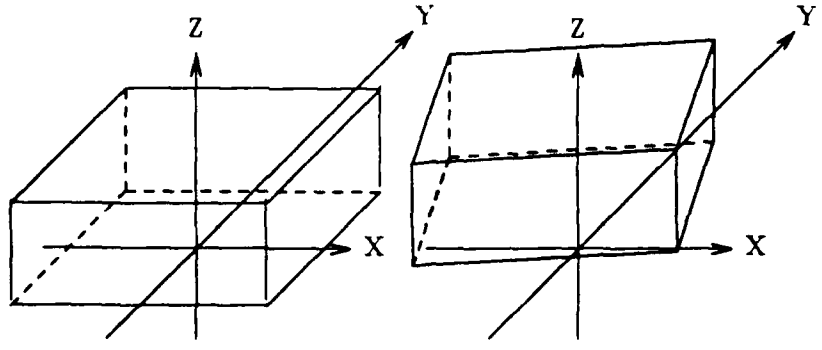


Figure 6.4 Rotation and Translation of a Target

Solid then applies the same routine as the polygon program to determine the hit probability for each face; that is, it integrates the bivariate normal density function under each directed line segment. The program assigns a zero probability of hitting polygons that 'face' away from the incoming shot. If faces overlap after rotation, the calculated hit probabilities will be too high, so be careful with convex solids.

Code.

```
      PROGRAM SOLID
c     Solid: Find hit probability on a polyhedral target.
      dimension x(100),y(100)
      parameter (NN=25)
      common /comtgt/ corner(NN,3), iface(NN,6), ncorns, nfaces
      common /projec/ v(3), p(3), lev
      real cornr2(NN,3)
1     format(25x,'x        y        z')
2     format(' Projectile velocity',3f10.3)
3     format(' Projectile position',3f10.3)
4     format(' Positions of the',i3,' corners:')
5     format(' Corners of the',i3,' faces:')
6     format(20x,3f10.3)
7     format(6i10)
8     format(' ',4f10.3)

      read *, sigmax, sigmay
      read *, lev
      read *, v
      read *, p
      read *, ncorns
      read *, ((corner(i,j),j=1,3),i=1,ncorns)
      read *, nfaces
      read *, ((iface(i,j),j=1,6),i=1,nfaces)
      IF (lev.gt.1) THEN
        print 8, sigmax, sigmay
        print 1
        print 2,v
        print 3,p
        print 4, ncorns
        print 6, ((corner(i,j),j=1,3),i=1,ncorns)
        print 5, nfaces
        print 7, ((iface(i,j),j=1,6),i=1,nfaces)
      ENDIF
      call xform(cornr2)
      pcum=0.
      DO 40 n=1,nfaces
c       Find j, the number of corners of face n
        j=0
        DO 20 k=1,6
          if (iface(n,k).gt.0) j=j+1
20      CONTINUE
c       Move corner coords to x,y vectors.
        DO 30 l=1,j
          m=iface(n,l)
          x(l)=cornr2(m,1)
          y(l)=cornr2(m,3)
30      CONTINUE
        ph=phit(j,x,y,0.,0.,sigmax,sigmay)
        if (ph.gt.0) pcum = pcum+ph
40    CONTINUE
      print *,' Phit =', pcum
      END

      FUNCTION PHIT(n,x,y,xbar,ybar,sigmax,sigmay)
c ***
c   this function computes the probability of hitting an
c   n-sided polygonal target.  the coordinates of the vertices
c   of the polygon are stored in the x and y arrays in
c   consecutive clockwise order around the polygon.  the
c   delivery error distribution is assumed to be bivariate
c   normal with mean (xbar,ybar) and with standard deviations
c   sigmax and sigmay respectively.
c ***
      dimension x(1),y(1)
1     format(' both sigmax and sigmay must be greater than zero'/
     * ' sigmax = ',f10.3,'    sigmay = ',f10.3)
c
      IF (sigmax.gt.0. .and. sigmay.gt.0.) THEN
        phit=0.
        DO 10 i=1,n
          j=mod(i,n)+1
          f=fsubij(x(i),y(i),x(j),y(j),xbar,ybar,sigmax,sigmay)
          phit=phit+f
10      CONTINUE
      ELSE
        write(6,1) sigmax,sigmay
        STOP
      ENDIF
      END
```

```
FUNCTION FUN(s)
common /fij/ a,b,c
fun=.3989422804*exp(-.5*s*s)*fnd(a+b*(s-c))
END
include 'qsimp.f'
include 'fnd.f'
include 'fnd2d.f'
include 'xform.f'
```

6.5 Tank3: Find Hit Probabilities on a target.

Tank3 finds the probability of hitting a target, where the target is represented by 2 rectangles. It was designed for tank-like targets but may be usable for other targets. The target is represented by 2 boxes, one above the other and meeting at the turret ring. Tank3 reads the width and height of the boxes and dispersions and bias at each desired range. At each range it finds the probability of hitting for a shot aimed at the fully exposed tank 0.3 meters below the center of the turret ring, for a shot at a hull defilade tank aimed at the center of the turret, and also for a shot at a standard 2.3 by 2.3 meter NATO target. Tank3 finds the hit probability for each of these postures at any desired range.

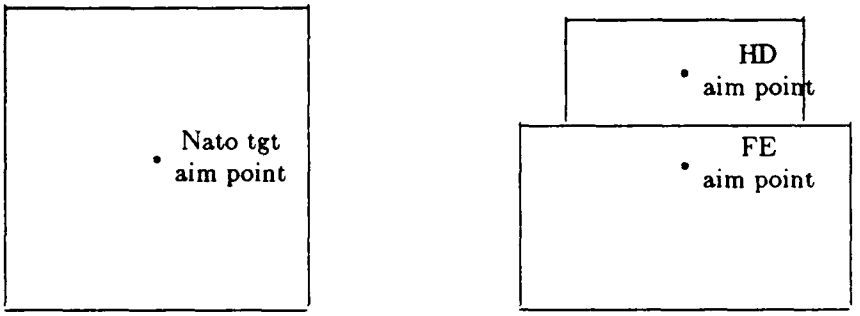


Figure 6.5 Targets and Aim Points.

Input. The command line can be 'tank3 -ph', where the p option prompts for input and the h option prints headers. Experiment using the p option to become familiar with the input. The input consists of two lines giving the linear dimensions of the turret and hull, followed by an arbitrary number of lines , each giving a range to target and accuracy parameters. The program stops at the end of the input file (^D) or (^C).

Sample Input

Data File	Description
2.35 0.75	Width, height of turret (m)
3.55 1.5	Width, height of hull (m)
0.25 1.3544 1.3544 0.0 0.0	Dist to target (km), σ_h, σ_v, μ_h, μ_v
0.50 0.8032 0.8020 0.0 0.0	
1.00 0.5747 0.5775 0.0 0.0	
1.50 0.5164 0.5218 0.0 0.0	
2.00 0.4957 0.5008 0.0 0.0	
2.50 0.4902 0.4916 0.0 0.0	
3.00 0.4919 0.4878 0.0 0.0	

Sample Output

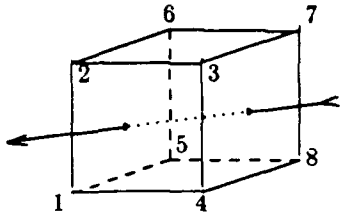
PHTANK3 19 Mar 83							
rg	dispersn (mils)		bias (mils)		hit prob		
(km)	horiz	vert	horiz	vert	FE	HD	NATO
0.25	1.35	1.35	0.00	0.00	1.00	0.74	1.00
0.50	0.80	0.80	0.00	0.00	0.99	0.66	0.99
1.00	0.57	0.58	0.00	0.00	0.94	0.47	0.92
1.50	0.52	0.52	0.00	0.00	0.81	0.33	0.75
2.00	0.50	0.50	0.00	0.00	0.66	0.23	0.58
2.50	0.49	0.49	0.00	0.00	0.52	0.16	0.44
3.00	0.49	0.49	0.00	0.00	0.40	0.12	0.33

Code.

```
c       TANK3: Find the probability of hitting a FE or HD target.
c        The target is represented as two rectangles.

        logical HEADER, PROMPT, is arg
1       format (6f8.2, 2f5.2)

        PROMPT= is arg('p')
        HEADER= is arg('h')
        IF (HEADER) THEN
         print*,' PHTANK3  19 Mar 83'
         print*,
     1  '  rg    dispersn (mils)  bias (mils)      hit prob',
     2  ' (km)    horiz   vert  horiz   vert    FE   HD NATO'
        ENDIF
        if (PROMPT) print*,' What is turret width & height (m)?'
        read *, tw, th
        if (PROMPT) print*,' What is hull width & height (m)?'
        read *, hw, hh

20      CONTINUE
c        Read and convert inputs
          if (PROMPT) print*,
     1    ' What are rg (km), disp-h, disp-v, bias-h, bias-v (mils)?'
          if (PROMPT) call flush()
          read(5,*,END=40) r, sh, sv, bh, bv
c        Convert to linear error
          xs = sh*r*0.9817
          ys = sv*r*0.9817
          xb = bh*r*0.9817
          yb = bv*r*0.9817
c        Find probability of hitting FE target
          ptur = prob(-.5*tw-xb,.5*tw-xb,0.3-yb,th+0.3-yb,xs,ys)
          phul = prob(-.5*hw-xb,.5*hw-xb,0.3-hh-yb,0.3-yb,xs,ys)
          ptank= ptur+phul
c        Find probability of hitting HD target
          pturHD = prob(-.5*tw-xb,.5*tw-xb,
     1       -.5*th-yb,.5*th-yb,xs,ys)
c        Find probability of hitting NATO target
          pNATO = prob(-1.15-xb,1.15-xb,
     1       -1.15-yb,1.15-yb,xs,ys)
          print 1, r, sh, sv, bh, bv, ptank, pturHD, pNATO
        GOTO 20
40      CONTINUE
        END

        FUNCTION PROB(xl,xh,yl,yh,xs,ys)
c       Prob: find probability of hitting box.
c       xl - distance from aim point to left side of box.
c       yl - distance from aim point to lower side of box.
c       xh - distance from aim point to right side of box.
c       yh - distance from aim point to top side of box.
c       xs - linear standard deviation of shot pattern horizontally.
c       ys - linear standard deviation of shot pattern vertically.
        real ndtr

        p1 = ndtr(xh/xs)
        p2 = ndtr(xl/xs)
        px = p1-p2
        p3 = ndtr(yh/ys)
        p4 = ndtr(yl/ys)
        py = p3-p4
        prob = px*py
        END
        include 'ndtr.f'
        include 'isarg.f'
```

6.6 Target1: Determines Which Faces of a Target are Pierced.

Given a target whose faces are convex polygons, the velocity vector of the projectile which travels in a straight line, and a point along the trajectory of the projectile, this program determines which target faces are entered and exited by the projectile. An explanation of the input and a sample run follow.



Figu 6.6 Diagram of Sample Input/Output Below.

Input/Output.

```
% target1
1         <-- output level
1.,-1.,0. <-- velocity vector
0.,1.414,0.5<-- position vector
8         <-- number of corners
1.,0.,0. \
1.,0.,1. |
1.,1.,1. |
1.,1.,0. |-- coordinates of the corners
0.,0.,0. |
0.,0.,1. |
0.,1.,1. |
0.,1.,0. /
6         <-- number of faces
1 2 3 4 0 0\
3 7 8 4 0 0|
6 2 1 5 0 0|-- corners of faces (in clockwise order)
2 6 7 3 0 0|
1 4 8 5 0 0|
7 6 5 8 0 0/

 Projectile exited  thru face  1
 Projectile entered thru face  2
%
```

Mathematics.

This program determines which target faces are entered or exited by a projectile.
The target is modelled as a solid whose faces are convex polygons.
The path of projectile is straight line through the target.
The flight path is determined by a velocity vector and a position vector.
The following is a sample run wherein the target is a rectangular box.

Code.

```
      PROGRAM TARGET1
c     Target1: Find whether a projectile strikes a polyhedral target
      parameter (NN=20)
      logical hit tgt, hit
      common /comtgt/ corner(NN,3), iface(NN,6), ncorns, nfaces
      common /projec/ v(3), p(3), lev
      real cornr2(NN,3)
1     format(26x,'x       y       z')                          50
2     format(' Projectile velocity',3f10.3)                    60
3     format(' Projectile position',3f10.3)
4     format(' Positions of the',i3,' corners:')
5     format(' Corners of the',i3,' faces:')
6     format(20x,3f10.3)
7     format(6i10)

c     Read inputs
      read *,lev
      read *,v
      read *,p
      read *,ncorns
      read *,((corner(i,j),j=1,3),i=1,ncorns)
      read *,nfaces
      read *,((iface(i,j),j=1,6),i=1,nfaces)
      IF (lev.gt.1) THEN
       print 1
       print 2,v
       print 3,p
       print 4, ncorns
       print 6,((corner(i,j),j=1,3),i=1,ncorns)
       print 5, nfaces
       print 7,((iface(i,j),j=1,6),i=1,nfaces)
      ENDIF
      call xform(cornr2)
      hit = hit tgt(cornr2)
      END

      LOGICAL FUNCTION HIT TGT (cornr2)
c     Purpose: Find whether target is hit
      logical hit fac
      parameter (NN=20)
      common /comtgt/ corner(NN,3), iface(NN,6), ncorns, nfaces
      common /projec/ v(3), p(3), lev
      dimension c(3), e(3), cornr2(NN,3)
1     format(' Corner vector',i3,' is:',3f10.3)
2     format(' Edge vector  ',i3,' is:',3f10.3,' cross product is:',f10.3)
3     format(' Face',i3,' has',i2,' corners.')

      hit tgt=.false.
c     Check each face to see if it was hit
      DO 50 n=1,nfaces
       hit fac=.false.
c      Find j, the number of corners of face n
       j=0
       DO 20 k=1,6
        IF (iface (n,k) .gt. 0) j=j+1
20     CONTINUE
       if (lev.eq.3) print 3,n,j
       k=iface(n,j)
       nneg = 0
       npos = 0
c      Check each edge to see if the polygon and origin are in the
c      same half plane.
       DO 30 l=1,j
c       Load corner vector
        c(1)=cornr2(k,1)
        c(2)=cornr2(k,2)
        c(3)=cornr2(k,3)
        if (lev.eq.3) print 1,k,c
c       Load edge vector
        k=iface(n,l)
        e(1)=cornr2(k,1) - c(1)
        e(2)=cornr2(k,2) - c(2)
        e(3)=cornr2(k,3) - c(3)
       y=e(1)*c(3)-c(1)*e(3)
       if (lev.eq.3) print 2,k,e,y
       if (y .lt. 0) nneg = nneg + 1
       if (y .gt. 0) npos = npos + 1
       if ((nneg .ge. 1) .and. (npos .ge. 1)) GOTO 40
30     CONTINUE
40     CONTINUE
       if (npos.eq.j .or. nneg.eq.j) hit fac = .true.
       IF (lev.gt.1 .and. hit fac) THEN
        if (nneg.eq.j) print*,'Projectile exited  thru face',n
        if (npos.eq.j) print*,'Projectile entered thru face',n
       ENDIF
       hit tgt=hit tgt .or. hit fac
       if (lev .eq. 1 .and. hit tgt) GO TO 60
50    CONTINUE
60    CONTINUE
      if (lev.eq.1 .and. hit tgt) print*,'Yes it was hit'
      if (.NOT. hit tgt) print*,'No it was not hit'
      END
      include 'xform.f'
```

7. Sensor

7.1 Eye: Find Detection Rate for the Human Eye.

Eye can be used to find detection probabilities for various types of targets; however, we generally use it for tank targets. Eye reads the sensor kind, the light level, the height of a tank turret, and the total tank height. It then finds the probability of ever detecting the tank and the median time to detect given that it is detectable. Eye generates these values for targets at 0.5, 1.0, ..., 4.0 km ranges.

Eye is code that was stripped from a much larger program developed by the Electro-Optical and Night Vision Laboratory. Unfortunately, the larger program is undocumented, so we cannot provide details of its inner workings; however, it is widely used.

Assumptions. Eye has the following built-in assumptions:

1. Acquisition is divided into the following categories:
   Detection - there's something there.
   Classification - it's tracked.
   Recognition - it's a tank.
   Identification - it's a T80.
   The program assumes acquisition at the recognition level.
2. The size of the search field is 225 degrees squared, e.g. 5 degrees high and 45 degrees wide.
3. The visibility range is 7 kilometers.
4. The contrast ratio (target to background?) at the target is 0.4.

Input. As an example, suppose the ambient light level is 300 ft-candles, the tank turret is 0.8 meters high, and the total tank is 2.2 meters high. The single input line would then be: '1 300. 0.8 2.2'. The 1 means that the sensor is the human eye. If you are interested in other targets, just input the appropriate heights. Typical light levels are:

Ft-candles	Typical Day
1000	Clear day
100	Overcast day
10	Heavy overcast day
1	Sunset overcast day

Output. Figure 7.1 illustrates the output. It shows the probability of ever detecting as a function of range, and the median time to detect given detection is possible.

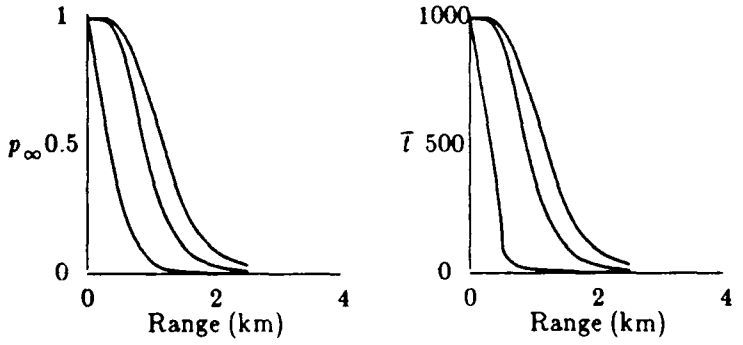


Figure 7.1 The Probability of Detecting a Target.

The output consists of 12 lines of input echo, then 7 lines of output proper. Line 13 is ranges in kilometers. Lines 14-16 are median ($\bar{t}$) times to detect given that detection is possible. The 14th line is for a stationary, hull-defilade target; the 15th for a stationary, fully-exposed target; and the 16th for a moving, fully-exposed target. Lines 17-19 are the corresponding probabilities that the target will ever be

detected. The probability of detecting in a time t is then: $p_d = p_\infty e^{-t/\tau}$

The output will look like this:

```
Input echo:
Kind sensor=      1
Light level=  300.00 (ft-candles)
Case:  1
Acquisition=      3.0 (scan lines)
Tgt height =    0.800 (meters)
Case:  2
Acquisition=      3.0 (scan lines)
Tgt height =    2.200 (meters)
Case:  3
Acquisition=      2.0 (scan lines)
Tgt height =    2.200 (meters)
  0.500   1.000   1.500   2.000   2.500   3.000   3.500   4.000
  0.240   0.028   0.006   0.002   0.001   0.000   0.000   0.000
  0.933   0.335   0.086   0.028   0.012   0.005   0.001   0.000
  0.990   0.667   0.230   0.080   0.034   0.013   0.001   0.001
 25.978 223.504 988.640 999.000 999.000 999.000 999.000
999.000
  6.516  18.632  72.974 220.279 527.750 999.000 999.000
999.000
  4.346   9.366  27.117  77.958 185.402 473.096 999.000
999.000
```

Code.

```
      real high(2), job(5,2)
      common /tables/ r(8), p(8,3), t(8,3)
1     format (/,' Input echo:',/,
     1  ' Kind sensor=',i10,/,
     2  ' Light level=',f8.2,' (ft-candles)')
2     format (' Case: ',i2,/,
     2  ' Acquisition=',f10.1,' (scan lines)',/,
     3  ' Tgt height =',f10.3,' (meters)')
3     format (8f8.3)
      data r /0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0/
      data job /1.0, 2.0, 2.5, 3.0, 6.4,  1.0, 2.0, 2.5, 4.0, 6.4/

      read*,kind,alumin,high(1),high(2)
      print 1, kind, alumin
c     Find values for HD stationary target.
       print 2, 1, job(4,kind), high(1)
       call nvl (1, alumin, high(1), job(4,kind), kind)
c     Find values for FE stationary target.
       print 2, 2, job(4,kind), high(2)
       call nvl (2, alumin, high(2), job(4,kind), kind)
c     Find values for FE moving target.
       print 2, 3, 0.667*job(4,kind), high(2)
       call nvl (3, alumin, high(2), 0.667*job(4,kind), kind)
      print 3, r, p, t
      END

      SUBROUTINE NVL (j,alumnc, dim, ajob, kind)
      common /tables/ r(8), p(8,3), t(8,3)
      data zone, rinfd /225., .1/
      data visrg /7./

      DO 20 i=1,8
c     Find probabilities and median times for 8 ranges.
       rc   = 0.
       pinf = 0.
       tbar = 999.
       range = r(i)
       attn = 3.912/visrg
c      IF (kind.eq.1) THEN
        rc =  eye (alumnc, attn, range, visrg)
        fov = 24.5
c      ELSEIF (kind.eq.2) THEN
c       rc = devic2 (attn, range)
c       fov = 11.98
c      ENDIF
       rc = rc*dim/range
       IF (rc.ge.rinfd) THEN
        x = rc/ajob
        y = 2.7+0.7*x
        z = x**y
        pinf = z/(z+1.0)
        pinf = amin1(pinf,.99)
        tau = zone / (fov*amin1(5.,fov))
        tbar = 3.4*tau/pinf
        if (pinf.gt.0.9) tbar = tau*ajob*6.8/rc
       ENDIF
       if (tbar.gt.999.0) tbar=999.0
       p(i,j) = pinf
       t(i,j) = tbar
20    CONTINUE
      END

      FUNCTION EYE (alumnc, attn, range, visrg)
c     Eye: find resolvable cycles for the human eye. (Device 1)
      real a(4,7)
c     sunset o'cast  heavy o'cast  overcast day  clear day
      data a/
     1  1.2378091942, 1.7176916034, 1.9909928015, 2.0892716525,
     2  0.4694720809,  .4739084812,  .4484981232,  .2813866389,
     3   .0493317078, -.2102695514, -.4084256747,-1.0084578626,
     4 -.0601756751, -.4161055149, -.6856409935,-1.4323484287,
     5 -.0558327470, -.2696921300, -.4318233767, -.8450225947,
     6 -.0174190671, -.0756229822, -.1197712507, -.2235482536,
     7 -.0018530403, -.0077222394, -.0121729428, -.0218136690/
      data acon /.4/

c     Find sky-to-ground ratio
       sog = (visrg+1.0)/3.
       sog = amin1(3.,amax1(1.,sog))
      eye = 0.0
      cntrst = acon/(1.0+sog*(exp(attn*range)-1.0))
      IF (cntrst.ge.0.02) THEN
c     Target/Background contrast is sufficient to detect
       i = min0(4,1+int(alog10(alumnc)))
       ack = 10.**i
       j = min0(4,i+1)
       clog = alog(cntrst)
       rlo = a(i,7)
       rhi = a(j,7)
       DO 20 k=6,1,-1
        rlo = rlo * clog + a(i,k)
        rhi = rhi * clog + a(j,k)
20     CONTINUE
c      Interpolate & compute cycles across target
        eye = rlo+(rhi-rlo)*(alumnc-ack/10.)/(ack*.9)
      ENDIF
      END
```

7.2 Los: Approximate Line-of-Sight Distribution

The length of line-of-sight segments can be modelled using the Weibull distribution. This tool finds the 1-(j*.05) quantiles (j=1,2,...,19) of 1-F(x), where F(x) is the Weibull distribution with shape parameter, β and scale parameter, α. The Weibull distribution and how to use it to model line-of-sight segments is discussed, and an example of a run is shown.

The broken line in Figure 7.2 shows the probability that a target travels a given distance (or greater) before going out of view. The statistics are based on field measurements. The smooth curve shows the Weibull fit to the data, using the parameters $\alpha = 300$ and $\beta = 1.1$ and illustrated in the dialog below. As expected, the probability that a target will travel more than a given distance before going out of sight decreases as the given distance increases.

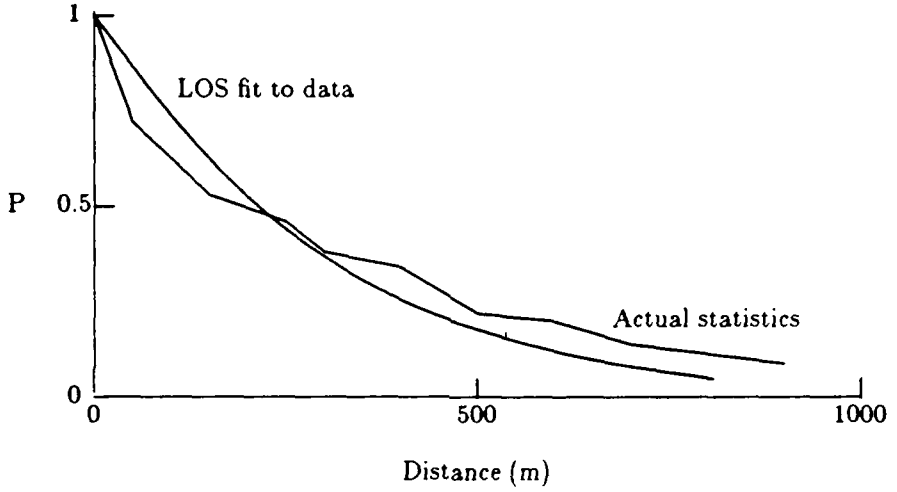


Figure 7.2 Probability vs Line-of-Sight Segment Length

Input/Output. Here is a sample dialog illustrating the use of Los:

```
% a.out
  What are shape, scale constants? (zeros to quit)
1.1,300.
  beta=   1.100000   alpha=   300.0000
%=  95  90  85  80  75  70  65  60  55  50  45  40  35  30  25  20  15  10   5
L>  20  38  57  76  96 117 139 162 187 214 244 277 313 355 403 462 536 640 813

  What are shape, scale constants? (zeros to quit)
1.0,300.
  beta=   1.000000   alpha=   300.0000
%=  95  90  85  80  75  70  65  60  55  50  45  40  35  30  25  20  15  10   5
L>  15  31  48  66  86 107 129 153 179 207 239 274 314 361 415 482 569 690 898

  What are shape, scale constants? (zeros to quit)
2.0,750.
  beta=   2.000000   alpha=   750.0000
%=  95  90  85  80  75  70  65  60  55  50  45  40  35  30  25  20  15  10   5
L> 169 243 302 354 402 447 492 536 579 624 670 717 768 822 883 951103311381298
```

```
  What are shape, scale constants? (zeros to quit)
0.,0.
  Bye!  I take shape <=0 as a signal to quit.
%
```

Mathematics. This software tool finds the quantiles, .05, .10, ..., .95, of F(x) = 1-W(x), where W(x) is the Weibull distribution, given its shape and scale parameters. The Weibull distribution is given below.

$$W(x) = \int_0^x (\alpha/\beta) t^{\alpha-1} e^{-t^{\alpha}/\beta} dt = 1 - e^{-(X/\alpha)^{\beta}}, X > 0$$

$$F(x) = 1 - W(x) = e^{-(x/\alpha)^{\beta}}$$

To find the quantile of order p, ς_p, one simply sets $F(\varsigma_p)$ to the desired p and solves for ς_p.

$$p = F(\varsigma_p) = e^{-(\varsigma_p/\alpha)^{\beta}}$$

$$\varsigma_p = \alpha(-\ln p)^{1/\beta}$$

To use the program, specify an α and a β. The program will calculate the ς_ps, p = 1 - j/20, j=1,2,...,19.

Code.

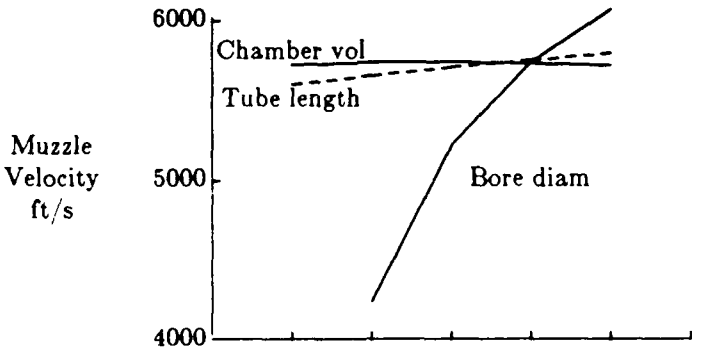
```
c       Los: given Weibull parameters, find LOS segment lengths.
c       a - alpha, the scale parameter.
c       b - beta, the shape parameter.
c       l(i) - length of segment in meters, corresponding to p2(i).
c       p2(i) - the ith probability in percent.
c       The shape constant should be between 0.5, 1.5.
c       Typical scale constants are 300m, 750m.
        integer l(19), p2(19)
1       format ('%=',19i4)
2       format ('L>',19i4)

20      CONTINUE
         print *,'What are shape, scale constants? (zeros to quit)'
         read *, b, a
         IF (b.le.0.) GOTO 99
          p = 0.95
          DO 30 n=1,19
           p2(n) = p*100.0+0.5
           f = - alog(p)
           l(n) = a*f**b
           p = p-0.5
30        CONTINUE
          print 1, p2
          print 2, l
         GOTO 20
99      CONTINUE
        print *,'Bye!  I take shape <=0 as a signal to quit.'
        END
```

8. Ballistics

8.1 Mayer: Find the Muzzle Velocity Using the Mayer-Hart Equation

The muzzle velocity of a kinetic energy projectile is a function of projectile, gun, and charge parameters. Mayer reads these parameters and finds the muzzle velocity using the Mayer-Hart[14] equation.

Designing a gun system involves choosing the parameters of the projectile, gun, and charge. Figure 8.1 shows the muzzle velocity when 3 parameters are varied around the values given in the sample input/output below.



Bore Diam (in)	1	2	3	4	5	6
Tube Length (in)	150	160	170	180	190	200
Chamber Vol (in^3)	150	160	170	180	190	200

Figure 8.1 Effects of Varying 3 Parameters in the Mayer-Hart Equation

aput/Output. The following illustrates the input and output of a sample run.

INPUT	OUTPUT		
4.000,	5.250, 1.300	Bore diameter	4.000
4368000.0,	4.630, 57500.0	Charge mass	5.250
0.060,	177.200, 170.000	Specific heat ratio	1.300
	Specific force	4368000.0	lbs/lb
	Package mass	4.630	lbs
	Maximum pressure	57500.0	lbs/sq in
	Charge density	0.060	lbs/cu in
	Tube length	177.200	in
	Chamber volume	170.000	cu in
	Muzzle velocity =	5729.56	ft/sec

Although the equation was derived in 1943 for cannon of that era, this program is still in use, and inputs are available from the Interior Ballistics Division of the United States Army Ballistics Research Laboratory. The numbers are simple to get except for specific heat ratio and specific force, though the former is probably constant.

14. Joseph Mayer, *Simplified Equations of Interior Ballistics*, BRL Report #388, Ballistic Research Laboratory, APG, MD, 1943.

Mathematics. The Mayer-Hart equation finds the kinetic energy of a projectile at the muzzle. This equation can be rearranged to find the velocity of the projectile at the muzzle. The Mayer-Hart equation is

$$\frac{1}{5.37} W u_m^2 = \frac{C\lambda}{\gamma-1}\left\{1-\left(v_o/v_m\right)^{\gamma-1}\left[1-\frac{1}{2}(\gamma-1)\left(P_c/P_q\right)\right]^{-1}\right\}$$

Solving for the muzzle velocity, u_m, yields

$$u_m = \left\{ 5.37\frac{C\lambda}{W(\gamma-1)}\left\{1-\left(v_o/v_m\right)^{\gamma-1}\left[1-\frac{1}{2}(\gamma-1)\left(P_c/P_q\right)\right]^{-1}\right\}\right\}^{1/2}$$

where

λ= specific force of propellant (dimensionless)
γ= ratio of the specific heats (dimensionless)
C= total charge (lbs)
$W = M + C/\nabla$
M= package mass (lbs)
$\nabla = 3 + .175C/M$
v_o= chamber volume minus original charge volume (in^3)
$= v_c - C/\rho$
v_c= chamber volume (cu in)
ρ= charge density (lbs/lb)
v_m= volume behind the projectile when the projectile is at the muzzle (in^3)
$= v_o + \pi r^2 t$
r= radius of the bore (in) = $b_d/2$
b_d= bore diameter (in)
t= tube length (in)
P_c= a constant with dimension of pressure (lbs/in^2)
$= C\lambda/v_o$
P_q= another constant with dimension of pressure (lbs/in^2)

$$= F_b P_{max}\left[\left(\frac{\gamma-1}{2}\right)^{\gamma-1}\gamma^{-2\gamma}\right]^{\frac{-1}{\gamma-1}}$$

F_b= correction factor for high velocity rounds

$$= \left(1+\frac{C}{\nabla+M}\right)\Big/\left(1+\frac{3/2\ C}{\nabla+M}\right)$$

P_{max}= maximum pressure (lbs/in^2)

The following assumptions are imbedded in this equation:
zero starting pressures
covolume equal to charge volume
burning rate proportional to pressure
constant burning surface
no heat loss through gas or projectile friction

Code.

```
c       MAYER-HART: Find muzzle velocity using Mayer-Hart eq.
c       bd - bore diameter (inches)
c       c - charge mass (pounds)
c       del - ?
c       G - specific heat ratio (gamma)
c       L - specific force (lbs/lb)? (lambda)
c       m - package mass (lbs)
c       pc - ?
c       pq - ?
c       r - radius of the bore (in)
c       rho - charge density (pounds/cu inch)
c       pmax - max pressure (lbs/sq in)
c       tl - tube length (inches)
c       vc - chamber volume (cu inches)
c       vm - volume of chamber + tube (cu inches)
c       w - projectile weight plus '1/3'

        real L, m
        data PI /3.14159265/
1       format(' Muzzle velocity   =',f10.2,' ft/sec')
2       format(a20,f10.3,a)
3       format(a20,f10.1,a)

c       Read and print
         read *, bd, c, G
         read *, L, m, pmax
         read *, rho, tl, vc
         print 2, ' Bore diameter      ',bd,' in'
         print 2, ' Charge mass        ',c, ' lbs'
         print 2, ' Specific heat ratio',G
         print 3, ' Specific force     ',L, ' lbs/lb'
         print 2, ' Package mass       ',m, ' lbs'
         print 3, ' Maximum pressure   ',pmax,' lbs/cu in'
         print 2, ' Charge density     ',rho,' lbs/cu in'
         print 2, ' Tube length        ',tl,' in'
         print 2, ' Chamber volume     ',vc,' cu in'
        r = 0.5*bd
        v0 = vc - c/rho
        del = 3.0+0.175*(c/m)
        w=m+c/del
        vm = v0 + tl*PI*r**2
        pc = c*L/v0
c       Find constant pq
         g1 = c/(del+m)
         fb = (1.0+g1) / (1.0+1.5*g1)
         g2 = (G+1.0) ** (G+1.0)
         g3 = G ** (-2.0*G)
         g4 = 2.0 ** (-(G+1.))
         g5 = (g2*g3*g4) ** (1.0/(G-1.0))
         pq = fb * pmax / g5
        f1 = 5.37*c*L / (w*(G-1.0))
        f2 = (v0/vm) ** (G-1.0)
        f3 = 1.0 - 0.5*(G-1.0)*pc/pq
        vel = sqrt(f1*(1.-f2/f3))
        print 1, vel
        END
```

INTENTIONALLY LEFT BLANK.

8.2 Shoot: Find the Trajectory of a Bullet.

Shoot solves the differential equations of motion to determine the trajectory of a bullet fired at a target. It asks for the launch angle, muzzle velocity, target range, and drag.

Input/Output. Execute Shoot using this syntax: shoot -phH
where, the p option prints prompts, the h option prints headings, and the H option implies a HEAT round rather than the default KE round.

If the HEAT option is chosen, the program reads drag related input from file 'heat.dat'. The last two lines of the 'heat.dat' file contain drag data. The Firing Tables Branch of BRL generates drag data that is easily converted to the form used here. See section 8.4 for more details on the drag on a HEAT round. Here is a sample heat.dat file:

Data File	Description
27. 1.2249992	Ambient temperature (Centigrade), air density
.105 10.2	Round diameter (m), mass (kg)
6	# drag values on next two lines.
1.2 1.4 1.8 2.5 3.0 3.8	mach number
.86 .8 .71 .575 .5 .39	drag coefficient

Sample Dialog. The following is a sample dialog in which Shoot generates the trajectory of a HEAT round:

```
% shoot -phH
  SHOOT  8 Dec 83
  What are launch ang, vm, tgt rg?
0.885,1000.,2000.
 Launch ang (deg) =   0.88 Muz vel (m/s) = 2000.000
 Drag (m/s/km)    =    0.0 Tgt rg (m) =    2000.0
   time     x      y     dx     dy     ddx     ddy
   0.00     4.0    0.1   999.    15.  -269.    -14.
   0.20   198.5    2.9   947.    13.  -252.    -13.
   0.40   382.9    5.1   898.    10.  -236.    -12.
   0.60   557.8    6.9   852.     8.  -221.    -12.
   0.80   724.0    8.2   809.     5.  -207.    -11.
   1.00   881.8    9.1   769.     3.  -194.    -11.
   1.20  1031.9    9.5   732.     1.  -181.    -10.
   1.40  1174.7    9.5   697.    -1.  -169.    -10.
   1.60  1310.8    9.2   664.    -3.  -158.     -9.
   1.80  1440.5    8.5   634.    -5.  -147.     -9.
   2.00  1564.3    7.4   605.    -6.  -138.     -8.
   2.20  1682.7    6.0   578.    -8.  -129.     -8.
   2.40  1795.8    4.2   553.    -9.  -121.     -8.
   2.60  1904.2    2.    530.   -11.  -113.     -7.
   2.79  2000.0    0.    510.   -12.  -106.     -7.
%
```

Mathematics. Finding the velocity of direct fire rounds is easy, since there is little change in air density for a flat trajectory and since the time of flight is so short that factors like the Earth's rotation can be ignored. For KE rounds, we assume the velocity simply decreases as a linear function of range. For HEAT rounds, the velocity is more complex. Section 8.4 discusses the mathematics of drag on a HEAT round.

Given the position x_0, and velocity $\dot{x}_0$ at time zero, we can use the relationship $\ddot{x} = -k v \dot{x}$ to find the acceleration, and then use

$$x_1 = x_0 + \dot{x}_0 t + \ddot{x}_0 t^2/2$$

to find the position at time t. But this is only exact if the acceleration is constant! It is not; it is changing monotonically in the region of interest, so let's use $\ddot{x}_i = (\ddot{x}_0 + \ddot{x}_1)/2$ as a better approximation to the average acceleration during the interval. There are 4 steps; step 3 is the key one.

STEP	CODE	EQUATION
1	ddx	$\ddot{x}_0 = -kv\dot{x}_0$
2	dx1	$\dot{x}_1 = \dot{x}_0 + \ddot{x}_0 t$
3	ddx	$\ddot{x}_i = (\ddot{x}_0 + \ddot{x}_1)/2 = (\ddot{x}_0 - kv\dot{x}_1)/2$
4	x	$x_1 = x_0 + \dot{x}_1 t + \ddot{x}_i t^2/2$

Code.

```
c       SHOOT: Numerically generate a ballistic trajectory
        logical PROMPT, HEADER, HEAT, is arg
3       format(
     1  ' Launch ang (deg) = ',f6.2,' Muz vel (m/s) = ',f8.3,/
     2  ' Drag (m/s/km)    = ',f6.1,' Tgt rg (m) =   ',f8.1)

        PROMPT= is arg('p')
        HEADER= is arg('h')
        HEAT= is arg('H')
c       Read inputs.
         IF (HEADER) THEN
          print*,' SHOOT  8 Dec 83'
          print*,' '
         ENDIF
         IF (PROMPT) THEN
          if (HEAT) print*,' What are launch ang, vm, tgt rg?'
          if (.not.HEAT) print*,
     1      ' What are launch ang, vm, tgt rg, drag?'
         ENDIF
         if (HEAT) read *, ang, v, tgt rg
         if (.not.HEAT) read *, ang, v, tgt rg, drag
        IF (HEADER) THEN
         print*
         write(6,3) ang, v, drag, tgt rg
         print*,
     1    ' time    x     y     dx     dy     ddx     ddy'
        ENDIF
        call fire(ang,v,drag,tgtrg,HEAT)
        END

        SUBROUTINE FIRE(ang,v0,drag,tgtrg,HEAT)
c       ang - launch angle w/ respect to horizontal (deg)
c       ddx - horizontal acc (m/s)
c       ddy - vertical acc (m/s)
c       drag - velocity decrease (m/s/km)
c       dt - time increment (sec)
c       dx - horizontal vel (m/s)
c       dy - vertical vel (m/s)
c       G - gravitational acceleration (m/s**2)
c       k - a drag constant (?)
c       rg - distance traveled by projectile thus far (m)
c       t - time of flight (sec)
c       tgt rg - range from firer to target (m)
c       theta - launch angle (rad)
c       t hit - time remaining to hit tgt (sec)
c       v - current velocity (m/s)
c       v0 - muzzle velocity (m/s)
c       x - horizontal coord (m)
c       y - vertical coord (m)
        logical HEAT
        real k
        data G, PI /9.80665, 3.14159265/
2       format(f8.2,2f8.1,4f8.0)

        theta = PI*ang/180.
c       Find initial conditions
        i = 0
        v = v0
        k = drag/(1000.*v)
        if (HEAT) call drag0
        x = 0
        y = 0
        dt = .002
        t = 0.
        dx = v*cos(theta)
        dy = v*sin(theta)
c       Find position at current time
20      CONTINUE
        if (HEAT) call dragf(k,v)
        dt2 = 2.*dt
        ddx = -v*k*dx
        dx3 = dx +ddx*dt2
        ddx = .5*(ddx-v*k*dx3)
        dx2 = dx + ddx*dt
        x  = x + dx2*dt2
        dx = dx3

        ddy = -v*k*dy - G
        dy3 = dy + ddy*dt2
        ddy = .5*(ddy-v*k*dy3-G)
        dy2 = dy + ddy*dt
        y  = y + dy2*dt2
        dy  = dy3
        v  = sqrt(dx*dx + dy*dy)
        t  = t+dt2
c       Print trajectory
        if (mod(i,50).eq.0)  print 2, t, x, y, dx, dy, ddx, ddy
        i = i+1
        rg = sqrt(x*x+y*y)
        t hit = 0.5*(tgt rg - rg)/v
        if (dt .gt. t hit) dt = t hit
       IF (rg.lt.tgtrg-.001) GOTO 20
       write(6,2) t,x,y,dx,dy,ddx,ddy
       END
       include 'isarg.f'
       include 'drag.f'
       include 'hunt.f'
```

8.3 Super: Find the Super-Elevation of a Gun.

Super finds the angle of super-elevation for an APFSDS round. Tell the program the muzzle velocity (assumed to be supersonic), drag, and target range. It solves for the angular elevation of the gun barrel needed to compensate for gravitational effects when engaging a target at the same altitude.

Input/Output. Super is an interactive program that prompts for three floating point values. Enter the muzzle velocity (m/s), drag (m/s/km), and target range (m). The program calculates the super elevation in milliradians, degrees, and mils.

Mathematics. The program solves for the super elevation angle θ that is diagrammed in Figure 8.3. This represents the elevation of a gun barrel necessary to compensate for the effects of gravity on a bullet's trajectory. We assume that the firer and target are at the same altitude, the muzzle velocity is supersonic, and the round is APFSDS and not HEAT. To determine the desired quantity, super uses this equation:

$$\sin\theta = \frac{g}{4r(kv)^2}\left[\, e^{2kr} - 2kr - 1 \,\right]$$

Where,

θ = super elevation (rad),
g= gravitational acceleration = 9.8 m/s^2,
v= muzzle velocity (m/s),
k= drag coefficient = drag/(1000v),
r= target range (m).

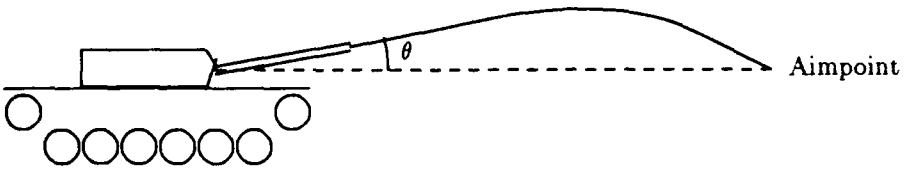


Figure 8.3 The Super-elevation Angle θ

Code.

```
c       SUPER: find super-elevation given muzzle vel, drag factor, tgt rg
        real k, MILS
        data MILS, DEG /1.0185916, 0.05729578/
1       format(6f10.3)
2       format(
     *  ' Muzzle vel (m/s) = ',f8.2,'  drag (m/s/km) =',f6.2,/,
     * ' Tgt range (m)    = ',f8.2,/,
     *  )
3       format(' S.E.=',f6.2,'mrad,',f6.3,'deg, or',f6.2,'mils.')

        print*,' What is v, drag, tgt rg?'
        read 1, v,drag,tgt rg
        print 2, v,drag,tgt rg
        k = drag/(1000.*v)
        rg = tgt rg
        a = 9.8/(4.0*rg*(k*v)**2)
        b = 2.0*k*rg
        stheta = a*(exp(b) - b - 1.0)
        r = asin(stheta)*1000.
        rdeg = r*DEG
        rmils = r*MILS
        print 3,r,rdeg,rmils
        END
```

8.4 Drag: Find the Deceleration of a Bullet.

Drag finds the coefficient k for these equations giving the deceleration of a bullet:

$$\ddot{x} = -kv\dot{x}$$
$$\ddot{y} = -kv\dot{y} - g$$

Step 1: Find the speed of sound. We use the standard atmosphere at sea level to find the speed of sound v_t at the desired temperature t in degrees Centigrade. A temperature of 15 degrees centigrade (59 degrees Fahrenheit) is customarily used for the temperature at which combat occurs.

Standard Conditions at Sea Level

Symbol	Value	Definition
T_0	273.15 deg Kelvin	Absolute air temperature (freezing)
ρ	1.2249992 kg/m^3	Air density
g	9.80665 m/sec^2	Gravitational acceleration
v_0	331.3 m/s	Speed of sound

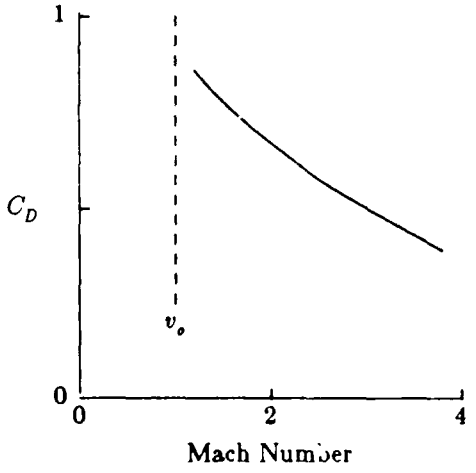
The formula for the speed of sound at T degrees Centigrade is:

$$v_t = v_0\sqrt{(T_0+T)/T_0}$$

Step 2: Find the Mach number. The mach number is:

$$n = v/v_t$$

Step 3: Find the drag coefficient (C_D). C_D is tabled for each round. Figure 8.4 shows a typical curve.



Mach #	C_D
1.2	.86
1.4	.80
1.8	.71
2.5	.575
3.0	.50
3.8	.39

Figure 8.4 Drag Coefficient for a Typical 105mm HEAT Round.

Linear interpolation using the following formula is satisfactory:

$$C_D = C_i + (C_{i+1} - C_i)(n - n_i)/(n_{i+1} - n_i)$$

Step 4: Find the constant k.

$$k = C_D \pi d^2 \rho / 8m,$$

Where,

$v = \sqrt{\dot{x}^2 + \dot{y}^2}$,
d = diameter of the round (m).
C_D = the drag coefficient (kg · m/sec^2).
m = mass of the round (kg).

Code. To imbed this in a program, use a call to drag0 once each time the atmospheric conditions or round parameters change. Call drag once for each step along the trajectory.

```
      SUBROUTINE DRAG0
c     Drag0: read atmospheric, round, and round drag data.
      real x(6), c(6), k, m, n
      data G, PI /9.80665, 3.14159265/
      data T0, v0 /273.13, 331.3/
      save

      open(4,file='heat.dat',status='old')
      rewind 4
      read(4,*) T, rho
      read(4,*) d, m
      read(4,*) n
      read(4,*) (x(i),i=1,n)
      read(4,*) (c(i),i=1,n)
      vs = v0*sqrt((T0+T)/T0)
      c1 = PI*d**2*rho/(8.0*m)
      RETURN

      ENTRY DRAGF (k,v)
c     Drag: Find drag factor k.

      n = v/vs
      call hunt (x,6,v/vs,i)
      CD = c(i) + (c(i+1)-c(i)) * (n-x(i)) / (x(i+1)-x(i))
      k = CD*c1
      END
      include 'hunt.f'
```

9. General Utility

9.1 Calc: Calculate Little Miscellaneous Items.

Calc performs numerous small calculations that must be made from time to time. It aggregates what would otherwise be a large number of Fortran object files into one. It demonstrates how this can be done and is easily modified to suit your own purposes.

Since we type slowly and read quickly, Calc requires few keystrokes and generally produces more than we need. It is then simple to spot the needed item in a list of several outputs. Calc calculates simple mathematical functions, some statistical measures, and hit probabilities on simple targets. The program uses the first 4 characters of your input to determine the type of calculation you wish to make. It may then read up to 6 numbers with decimal points, separated by commas as the data.

The following table lists the commands and results.

Table 9.1 Commands

COMMAND	RESULT
q	Quits the Calc program
?	Anything unrecognizable prints help info
echo	Toggle input echo
	SIMPLE ARITHMETIC FUNCTIONS
math a b	a+b, a-b, a*b, a/b, a**b, e**a, log a, ln a
root a b	$\sqrt{a}$, $\sqrt{b}$, $\sqrt{(a^2+b^2)}$, $\sqrt{((a^2+b^2)/2)}$
intp x x_1 y_1 x_2 y_2	Linear interpolation
vec **a** **b**	a, b, **a**+**b**, **a**−**b**, **a**·**b**, **a**×**b**
	TRIGONOMETRY
aerr e r	Convert from linear error (m) & range (km) to angular error (mils).
lerr α r	Convert angular error (mils) & range (km) to linear error (meters).
rad	Shift to radians
deg	Shift to degrees
mil	Shift to mils (6,400 per circle)
trig a	Sin (a), cos (a), tan (a)
	PROBABILITY
bino n p	Draw w/ replacement
hypr i j n	Draw w/o replacement
ndtr a b	Integrate the standard normal from a to b
	HIT PROBABILITIES
circ r σ	Find hit probability on a circular target.
box x_a x_b y_a y_b	Define edges of a rectangular target
sig σ_x σ_y	Find hit probability on a rectangular target
	STATISTICS
conb	Find binomial confidence interval
conn	Find normal confidence interval

Sample dialog. Here is a sample dialog. The user typed the lines beginning at the left margin and Calc responded with the indented lines.

```
math 3. 4.
 a  =  3.000000  b  =  4.000000
 a+b =  7.000000  a-b = -1.000000
 a*b =  12.00000  a/b = 0.7500000
 a**b=  80.99998  e**a=  20.08554
 log a 0.4771210  ln a=  1.098612
root 3. 4.
 sqrt=  1.732051  2.000000
 hypot  5.000000
 rms =  3.535534
intp 3. 1. 10. 5. 20.
 x1  =  1.000000  y1  =  10.00000
 x   =  3.000000  y   =  15.00000
 x2  =  5.000000  y2  =  20.00000
vec  1. 2. 3. 2. 3. 1.
 a  =  1.000000  2.000000  3.000000
 b  =  2.000000  3.000000  1.000000
 |a|=  3.741657  |b|=  3.741657
 a+b =  3.000000  5.000000  4.000000
 a-b = -1.000000 -1.000000  2.000000
 dot =  11.00000
 cros= -7.000000  5.000000 -1.000000
lerr .5 1.
 err =  0.5000000  mils
 range  1.000000  km
 err =  0.4908500  meters
aerr 2. 2.
 err =  2.000000  meters
 range  2.000000  km
 err =  1.018590  mils
deg
 DEGREES
trig 30.
 ang =  30.00000    deg
 sin =  0.4999998
 cos =  0.8660254
 tan =  0.5773500
rad
 RADIANS
trig 1.
 ang =  1.000000    rad
 sin =  0.8414710
 cos =  0.5403023
 tan =  1.557408
mil
 MILS
trig 10.
 ang =  10.00000    mil
 sin =  9.817317E-03
 cos =  0.9999519
 tan =  9.817790E-03
bino 5 .5
 n  =  5  |# of trials|
 p  =  0.5000000  |probability of success on 1 trial|
 Probability of  0 successes =  0.03
 Probability of  1 successes =  0.16
 Probability of  2 successes =  0.31
 Probability of  3 successes =  0.31
 Probability of  4 successes =  0.16
 Probability of  5 successes =  0.03
hypr 3 4 52
 3  # items drawn
 4  # desired items in urn
 52  # items in urn
 Probability of  0 successes is 0.783
 Probability of  1 successes is 0.204
 Probability of  2 successes is 0.013
 Probability of  3 successes is 0.000
ndtr -1. 1.
 a,b =  -1.000000  1.000000
 prob=  0.6826895  |integral of std normal from a to b.|
 densa  0.2419708  |exp(-a*a/2) /sqrt(2*pi)|
 densb  0.2419708  |exp(-b*b/2) /sqrt(2*pi)|
box -1. 1. -1. 1.
sig 1. 1.
 phit=  0.4660650  |hit probability on box|
conb 1000 .5
 n  =  1000  |sample size|
 p  =  0.5000000  |fraction of successes|
 90% =  0.4735267  0.5264733
 95% =  0.4685709  0.5314291
 99% =  0.4589077  0.5410923
conn 1000 50. 5.
 n  =  1000
 mean=  50.00000  s.d.=  5.000000
 90% =  49.73814  50.26186
 95% =  49.68739  50.31261
 99% =  49.58714  50.41286
q
```

Code.

```
c       Calc - makes a few simple calculations.
        character line*50, unit*3
        real ndtr, hi(3), lo(3), cn(6), v1(3), v2(3), v3(3)
        logical fail, echo
        data echo /.false./
3       format (a50)
4       format (8f8.4)
5       format('  a=',f8.2,a4,' sin(a)=',f9.6,'  cos(a)=',f9.6,
     1  '  tan(a)=',f9.6)
6       format (4x,i5, f10.3)
7       format ('90%  ',f6.3,'-'f6.3,'  95%  ',f6.3,'-'f6.3,'  99%  ',
     1  f6.3,'-'f6.3)

        pi = acos(-1.0)
        angfac = 1.0
        unit='rad'
20      CONTINUE
         read 3, line
         if (echo) print '(a)', line
         IF (line(1:1) .eq. 'q') THEN
          STOP
c       ARITHMETIC SECTION
         ELSEIF (line(1:4) .eq. 'math') THEN
          read (line(5:50),*) a,b
          print *, 'a  =',a ,' b  =',b
          print *, 'a+b =',a+b,' a-b =',a-b
          print *, 'a*b =',a*b,' a/b =',a/b
          print *, 'a**b=',a**b,' e**a=',exp(a)
          print *, 'log a',alog10(a),' ln a=',alog(a)
         ELSEIF (line(1:4) .eq. 'root') THEN
          read (line(5:50),*) a, b
          print *, 'sqrt=',sqrt(a),sqrt(b)
          print *, 'hypot',sqrt(a**2+b**2)
          print *, 'rms =',sqrt(0.5* (a**2+b**2))
         ELSEIF (line(1:4) .eq. 'intp') THEN
          read (line(5:50),*) x, x1, y1, x2, y2
          y = y1 + (y2-y1)*(x-x1)/(x2-x1)
          print *,'x1  =',x1,' y1  =',y1
          print *,'x   =',x ,' y   =',y
          print *,'x2  =',x2,' y2  =',y2
         ELSEIF (line(1:3) .eq. 'vec') THEN
          read (line(4:50),*) v1, v2
          vabs1 = sqrt(v1(1)**2 + v1(2)**2 + v1(3)**2)
          vabs2 = sqrt(v2(1)**2 + v2(2)**2 + v2(3)**2)
          print *,'a  =',v1
          print *,'b  =',v2
          print *,'|a|=',vabs1, '|b|=', vabs2
          print *,'a+b =',v1(1)+v2(1), v1(2)+v2(2), v1(3)+v2(3)
          print *,'a-b =',v1(1)-v2(1), v1(2)-v2(2), v1(3)-v2(3)
          print *,'dot =',v1(1)*v2(1)+v1(2)*v2(2)+v1(3)*v2(3)
          v3(1) = v1(2)*v2(3) - v1(3)*v2(2)
          v3(2) = v1(3)*v2(1) - v1(1)*v2(3)
          v3(3) = v1(1)*v2(2) - v1(2)*v2(1)
          print *,'cros=',v3
c       TRIGONOMETRY SECTION
         ELSEIF (line(1:4) .eq. 'lerr') THEN
          read (line(5:50),*) a, b
          print *, 'err =', a, ' mils'
          print *, 'range', b, ' km'
          print *, 'err =', 0.9817*a*b, ' meters'
         ELSEIF (line(1:4) .eq. 'aerr') THEN
          read (line(5:50),*) sx, r
          xmil = atan2(sx, 1000*r) * 1018.59
          print *, 'err =', sx,  ' meters'
          print *, 'range', r,   ' km'
          print *, 'err =', xmil, ' mils'
```

```
      ELSEIF (line(1:3) .eq. 'deg') THEN
        unit='deg'
        angfac = pi/180.0
        print *, 'DEGREES'
      ELSEIF (line(1:3) .eq. 'rad') THEN
        unit='rad'
        angfac = 1.0
        print *, 'RADIANS'
      ELSEIF (line(1:3) .eq. 'mil') THEN
        unit='mil'
        angfac = pi/3200.
        print *, 'MILS'
      ELSEIF (line(1:4) .eq. 'trig') THEN
        read (line(5:50),*) a
        s = sin(a*angfac)
        c = cos(a*angfac)
        t = tan(a*angfac)
        print *, 'ang =', a, ' ', unit
        print *, 'sin =', s
        print *, 'cos =', c
        print *, 'tan =', t
c     PROBABILITY SECTION
      ELSEIF (line(1:4) .eq. 'bino') THEN
        read (line(5:50),*) n, p
        print *, 'n   =', n, ' |# of trials|'
        print *, 'p   =', p, ' |probability of success on 1 trial|'
        call binom(n,p)
      ELSEIF (line(1:4) .eq. 'hypr') THEN
        read (line(5:50),*) i,j,n
        print *, i, ' # items drawn'
        print *, j, ' # desired items in urn'
        print *, n, ' # items in urn'
        call hyper(j,i,n)
      ELSEIF (line(1:4) .eq. 'ndtr') THEN
        read (line(5:50),*) a, b
        c = ndtr(b) - ndtr(a)
        e = exp(-.5*a**2)/sqrt(2*pi)
        f = exp(-.5*b**2)/sqrt(2*pi)
        print *, 'a,b =', a, b
        print *, 'prob=', c, ' |integral of std normal from a to b.|'
        print *, 'densa', e, ' |exp(-a*a/2) /sqrt(2*pi)|'
        print *, 'densb', f, ' |exp(-b*b/2) /sqrt(2*pi)|'
c     HIT PROBABILITY SECTION
      ELSEIF (line(1:4).eq.'circ') THEN
        read (line(5:50),*) r,d
        p = 1.0-exp(-0.5*r**2/d**2)
        print *, 'rad =', r, ' |target radius|'
        print *, 'S.D.=', d, ' |dispersion of rounds|'
        print *, 'prob=', p, ' |hit probability on circular tgt|'
      ELSEIF (line(1:3).eq.'box') THEN
        read (line(4:50),*) xl, xh, yl, yh
      ELSEIF (line(1:3).eq.'sig') THEN
        read (line(5:50),*) xs, ys
        px = ndtr(xh/xs) - ndtr(xl/xs)
        py = ndtr(yh/ys) - ndtr(yl/ys)
        print *,'phit=', px*py, ' |hit probability on box|'
c     STATISTICS SECTION
      ELSEIF (line(1:4).eq.'conb') THEN
        read (line(5:50),*) n, p
        print *, 'n   =', n, ' |sample size|'
        print *, 'p   =', p, ' |fraction of successes|'
        call confb(p,n,hi,lo,fail)
        if (fail) print *, 'Sample too small'
        if (.not.fail) print *, '90% =', lo(1), hi(1)
        if (.not.fail) print *, '95% =', lo(2), hi(2)
        if (.not.fail) print *, '99% =', lo(3), hi(3)
      ELSEIF (line(1:4).eq.'conn') THEN
        read (line(5:50),*) n, xbar, s
        call confn(xbar, s, n, cn)
        print *, 'n   =', n
        print *, 'mean=', xbar, ' s.d.=', s
        print *, '90% =',cn(1),cn(2)
        print *, '95% =',cn(3),cn(4)
        print *, '99% =',cn(5),cn(6)
c     MISCELLANEOUS SECTION
      ELSEIF (line(1:4).eq.'echo') THEN
        echo = .not. echo
      ELSE
        call help
      ENDIF
      GOTO 20
      END

      FUNCTION BICO (i,j)
c 1   Bico: find binomial coefficient.
      IF (j.eq.0 .or. j.eq.i) THEN
        bico = 1
      ELSE
        k = min0(j,i-j)
        p = i-k+1
        DO 20 m=2,k
          p = p*(i-k+m)/float(m)
20      CONTINUE
        bico = p
      ENDIF
      if (j.gt.i) bico = 0.0
c     Will fail if j<0 or j>i
      END

      SUBROUTINE BINOM(n,p)
1     format (' Probability of', i3, ' successes =', f6.2)

      DO 30 i=0,n
        IF (n.le.20 .or. i.lt.5 .or. 0.eq.mod(i,5)) THEN
          pi = bico(n,i) * p**i * (1.-p)**(n-i)
          print 1, i, pi
        ENDIF
30    CONTINUE
      END


      SUBROUTINE CONFB (p, nr, hi, lo, fail)
c     Conf: Find the 90, 95, and 99% confidence intervals
c      on a probability p, where p is the estimated probability
c      and nr is the sample size.
c     Reference: Introduction to Statistical Analysis, 3rd edition,
c      Dixon and Massey, p246.
      real hi(3), lo(3), z(3), n
      logical fail
      data z /1.645, 1.960, 2.576/

      n = float(nr)
      fail = (n*p.lt.5) .or. ((n-n*p).lt.5)
      IF (.not.fail) THEN
c     Method is applicable, so apply it
        DO 20 i=1,3
          s1 = n/(n+z(i)**2)
          s2 = 0.5*z(i)**2/n
          s3 = (p+0.5/n) * (1.0-p-0.5/n)
          s4 = (p-0.5/n) * (1.0-p+0.5/n)
          s5 = z(i)**2/(4.0*n*n)
          lo(i) = s1*(p-0.5/n+s2-z(i)*sqrt(s3/n+s5))
          hi(i) = s1*(p+0.5/n+s2+z(i)*sqrt(s4/n+s5))
20      CONTINUE
      ENDIF
      END

      SUBROUTINE CONFN(xbar, s, n, d)
c     CONFN: Find the 90, 95, 99% confidence intervals on a normal
c     distribution, given sample mean, sample S.D., sample size
      real d(6)
      factor = s/sqrt(float(n))
      call tstats(n-1,t1,t2,t3)
      d(1) = xbar - t1*factor
      d(2) = xbar + t1*factor
      d(3) = xbar - t2*factor
      d(4) = xbar + t2*factor
      d(5) = xbar - t3*factor
      d(6) = xbar + t3*factor
      END

      SUBROUTINE HELP
c     Help: print instructions.
1     format(a)

      print 1,
     1 'q        Quit',
     1 '?        Anything unrecognizable prints help info',
     1 'echo     Toggle input echo',
     1 '         SIMPLE ARITHMETIC',
     1 'math a b Simple arithmetic',
     1 'root a b Square roots of functions of a,b',
     1 'intp x x1 y1 x2 y2Linear interpolation',
     1 'vec a1 a2 a3 b1 b2 b3Vector arithmetic',
     1 '         TRIGONOMETRY',
     1 'aerr e r Convert from linear err (m) & range (km)',
     1 '          to angular error (mils)',
     1 'lerr a r Convert angular error (mils) & range (km) to',
     1 '          linear error (m)',
     1 'rad      Shift to radians',
     1 'deg      Shift to degrees',
     1 'mil      Shift to mils',
```

```
     1    'trig a     Find sin(a), cos(a), tan(a)'
          print 1,
     1    '           PROBABILITY',
     1    'bino n p  Draw w/ replacement',
     1    '            n is # of draws, p is prob of success',
     1    'hypr i j nDraw w/o replacement',
     1    '            i is # draws, j is # red balls, n is # balls',
     1    'ndtr a b  Integrate the standard normal from a to b',
     1    '           HIT PROBABILITY',
     1    'circ r sig Find hit probability on a circular target',
     1    'box xa ya xb ybDefine edges of a rectangular target',
     1    'sig sx sy  Find hit probability on a rectangular target',
     1    '            sx, sy are horiz, vert dispersions',
     1    '           STATISTICS',
     1    'conb n p  Find binomial confidence interval',
     1    '            n is sample size, p is estimated probability',
     1    'conn n xbar sdFind normal confidence interval',
     1    '            n, xbar, sd is sample: size, mean, std dev'
          END

          SUBROUTINE HYPER(i,j,n)
c         Hyper: find the outcomes of draws without replacement.
c         i = #of desired objects in the set from which you draw.
c         j = #of draws w/o replacement
c         m = maximum number that can be drawn
c         n = number of objects in the set from which you draw.
1         format (' Probability of',i3,' successes is',f6.3)
c
          m=min0(i,j)
          DO 20 k=0,m
            a1 = bico(i,k)
            a2 = bico(n-i,j-k)
            a3 = bico(n,j)
            p=a1*a2/a3
            print 1, k, p
20        CONTINUE
          END

          REAL FUNCTION NDTR (x)
c         Ndtr: Integrate the Normal Distribution from -infinity to x.
c         Source: Adapted from ndtr in the IBM SSP pg78.

          ax = abs(x)
          t = 1.0/(1.0+.2316419*ax)
          d = 0.0
          if(ax.lt.6.0)d = 0.3989423*exp(-x*x/2.0)
          p = 1.0 - d*t*((((1.330274*t - 1.821256)*t + 1.781478)*t -
     *    0.3565638)*t + 0.3193815)
          if (x.lt.0.0) p = 1.-p
          ndtr = p
          END

          SUBROUTINE TSTATS (df,t1,t2,t3)
c         Tstats: Find Student's T-statistic for 95%, 97.5%, 99.5%
          integer df, dfs(34)
          real t950(34), t975(34), t995(34)
          fun(a,b,c) = a + (b-a)*c
          data dfs  / 1, 2, 3, 4, 5,
     1              6, 7, 8, 9,10,
     2             11,12,13,14,15,
     3             16,17,18,19,20,
     4             21,22,23,24,25,
     5             26,27,28,29,30,
     6             40,60,120,9999999/
          data t950 / 6.314, 2.920, 2.353, 2.132, 2.015,
     1              1.943, 1.895, 1.860, 1.833, 1.812,
     2              1.796, 1.782, 1.771, 1.761, 1.753,
     3              1.746, 1.740, 1.734, 1.729, 1.725,
     4              1.721, 1.717, 1.714, 1.711, 1.708,
     5              1.706, 1.703, 1.701, 1.699, 1.697,
     6              1.684, 1.671, 1.658, 1.645/
          data t975 /12.706, 4.303, 3.182, 2.776, 2.571,
     1              2.447, 2.365, 2.306, 2.262, 2.228,
     2              2.201, 2.179, 2.160, 2.145, 2.131,
     3              2.120, 2.110, 2.101, 2.093, 2.086,
     4              2.080, 2.074, 2.069, 2.064, 2.060,
     5              2.056, 2.052, 2.048, 2.045, 2.042,
     6              2.021, 2.000, 1.980, 1.960/
          data t995 /63.657, 9.925, 5.841, 4.604, 4.032,
     1              3.707, 3.499, 3.355, 3.250, 3.169,
     2              3.106, 3.055, 3.012, 2.977, 2.947,
     3              2.921, 2.898, 2.878, 2.861, 2.845,
     4              2.831, 2.819, 2.807, 2.797, 2.787,
     5              2.779, 2.771, 2.763, 2.756, 2.750,
     6              2.704, 2.660, 2.617, 2.576/

          IF (df.le.0) THEN
            print *,'TSTATS: Degrees of freedom must be > 0.'
            STOP
          ELSEIF (df.le.30) THEN
            t1 = t950(df)
            t2 = t975(df)
            t3 = t995(df)
          ELSE
            j = 30
            if (df.gt.40) j = 31
            if (df.gt.60) j = 32
            if (df.gt.120) j = 33
            factor = float (df-dfs(j+1))*dfs(j) /
     1             float((dfs(j)-dfs(j+1))*df)
            t1 = fun(t950(j+1),t950(j),factor)
            t2 = fun(t975(j+1),t975(j),factor)
            t3 = fun(t995(j+1),t995(j),factor)
          ENDIF
          END
```

9.2 Histo: Generate Counts for a Histogram

Histo counts the number of data points that fall into each of a set of equally-sized intervals. Just input the number of intervals desired and the lower and upper bounds of the range. The program divides the range and displays a histogram of the data.

Input. A histo input file requires the following information in the first line, separated by blanks or commas:

number of data points
number of intervals desired
lower bound of first interval
upper bound of last interval

If you enter the same value for both the lower and upper bounds, the program uses the lowest and highest values of the data as the bounds. You should try to select a number of intervals that will divide the range neatly.

Subsequent input lines contain the data in free format.

Sample Problem. Suppose you fire a tank gun 50 times and record the horizontal distance that it hits from the target each time. With this data file as input, histo generates a table giving interval boundaries, frequencies, and cumulative frequencies, as shown in Figure 9.2. The exhibit also demonstrates how the frequency distribution given by the program can be converted into its graphical interpretation, the histogram. The output for this example clearly shows that the gun is biased to the right.

Here's the input file:

```
50 5 -0.2 0.8
-0.10  0.25  0.51  0.72  0.01  0.12  0.25  0.43
 0.28  0.31  0.00 -0.20 -0.18  0.74  0.41  0.52
 0.02  0.81 -0.15 -0.20  0.23  0.45  0.58  0.27
 0.62  0.01 -0.12  0.38  0.39  0.80  0.80  0.32
 0.21  0.77  0.15  0.05  0.31  0.41  0.16  0.19
 0.09  0.33  0.25  0.38  0.22  0.55  0.46  0.32
 0.59  0.41
```

Program Output:

lower	higher	count	cum
$-\infty$	0.000	6	6
0.000	0.200	10	16
0.200	0.400	16	32
0.400	0.600	11	43
0.600	$+\infty$	7	50

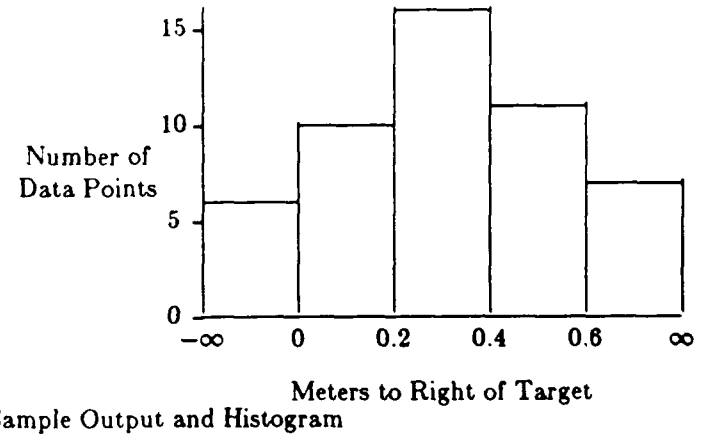


Figure 9.2 Sample Output and Histogram

Mathematics. This program simply generates counts for a histogram. It includes a data point in a specific interval if lower bound $\leq$ data point $<$ upper bound. A histogram and frequency distribution give discrete representations of a continuous sample. Thus, histo is useful for estimating probabilities and probability density functions.

Code.

```
      PROGRAM HISTO
c     Histo: Make counts for a histogram.
c     amin - lower boundary
c     amax - upper boundary
c     ni - #of intervals
c     np - #of points
c     x - vector of data points
      integer count(50), cum(50)
      real x(20000)
2     format(' -infinity',f10.3,2i8)
3     format(2f10.3,2i8)
4     format(f10.3,' +infinity',2i8)
5     format('    lower    higher   count    cum')

      read *, np, ni, amin, amax
      read *, (x(i),i=1,np)
      IF (amin.eq.amax) THEN
c     Find min & max
       amin=1e35
       amax=-amin
       DO 20 i=1,np
        if (x(i).lt.amin) amin=x(i)
        if (x(i).gt.amax) amax=x(i)
20     CONTINUE
      ENDIF
      size=(amax-amin)/ni
c     Tally into interval counters
       DO 40 i=1,np
        j=1+(x(i)-amin)/size
        if (j.le.0) j=1
        if (j.ge.ni) j=ni
        count(j) = count(j)+1
40     CONTINUE
c     Print results
       print 5
       alo=amin
       ahi=amin+size
       cum(1)=count(1)
       print 2, ahi, count(1), cum(1)
       DO 60 i=2,ni-1
        alo=ahi
        ahi=ahi+size
        cum(i)=cum(i-1)+count(i)
        print 3, alo, ahi, count(i), cum(i)
60     CONTINUE
       alo=alo+size
       ahi=ahi+size
       cum(ni)=cum(ni-1)+count(ni)
       print 4, alo, count(ni), cum(ni)
      END
```

9.3 Shell: Sort a list of numbers.

Shell sorts a list of numbers using the Shell sort method. The main routine of this program simply exercises the shell subroutine which does the actual sorting. The shell subroutine may be cut out and inserted in your program. With no change in the logic, integers may be sorted, and with some change, characters may be sorted.

There are many ways to sort. The classic reference by Donald Knuth[15] lists the following internal sorting methods:

METHOD	AVERAGE TIME
Insertion	
Straight insertion	$2N^2+$
Shell's method	$15N^{1.25}$
Exchange	
Bubble sort	$O(N^2)$
Quicksort	11.66N ln N
Selection	
Heapsort	23.08N log n +
Merging	
Distribution	

We see that the Shell sort is faster than some; furthermore, it is reasonably simple. Its advantage over a bubble sort is that it starts by exchanging pairs that are far apart, so it tends to place items near their final position quickly.

Table 9.3 Shell Sample Input / Output

Sample Input	8 45.	27.	82.	-4.	16.	66.	31.	30.
Sample Output	45. -4.	27. 16.	82. 27.	-4. 30.	16. 31.	66. 45.	31. 66.	30. 82.

15. *Sorting and Searching; The Art of Computer Programming*, Vol. 3, Addison Wesley, Reading, Massachusetts, 1973.

Code.

```
c       SHELL: Demonstrate shell sort.
        real a(0:99)
1       format(10f8.0)
c
        read *,n
        read *,(a(i),i=0,n-1)
        print 1,(a(i),i=0,n-1)
        call shell(a,n)
        print 1,(a(i),i=0,n-1)
        END

        SUBROUTINE SHELL (v,n)
c       Purpose: sort using Shell's method.
c       Ref - The C Programming Language, p58
        real v(0:99)
        integer gap

        gap = n/2
        DO 90 igap=1,15
          IF (gap.le.0) GOTO 99
            DO 80 i=gap,n-1
              j = i-gap
              DO 70 k=1,9999
                IF (j.lt.0 .or. v(j+gap).ge.v(j)) GOTO 71
                temp = v(j)
                v(j) = v(j+gap)
                v(j+gap) = temp
                j = j-gap
70            CONTINUE
71            CONTINUE
80          CONTINUE
          gap = gap/2
90      CONTINUE
99      CONTINUE
        END
```

INTENTIONALLY LEFT BLANK.

No of Copies	Organization
1	Office of the Secretary of Defense OUSD(A) Director, Live Fire Testing ATTN: James F. O'Bryon Washington, DC 20301-3110
2	Administrator Defense Technical Info Center ATTN: DTIC-DDA Cameron Station Alexandria, VA 22304-6145
1	HQDA (SARD-TR) WASH DC 20310-0001
1	Commander US Army Materiel Command ATTN: AMCDRA-ST 5001 Eisenhower Avenue Alexandria, VA 22333-0001
1	Commander US Army Laboratory Command ATTN: AMSLC-DL Adelphi, MD 20783-1145
2	Commander US Army, ARDEC ATTN: SMCAR-IMI-I Picatinny Arsenal, NJ 07806-5000
2	Commander US Army, ARDEC ATTN: SMCAR-TDC Picatinny Arsenal, NJ 07806-5000
1	Director Benet Weapons Laboratory US Army, ARDEC ATTN: SMCAR-CCB-TL Watervliet, NY 12189-4050
1	Commander US Army Armament, Munitions and Chemical Command ATTN: SMCAR-ESP-L Rock Island, IL 61299-5000
1	Commander US Army Aviation Systems Command ATTN: AMSAV-DACL 4300 Goodfellow Blvd. St. Louis, MO 63120-1798

No of Copies	Organization
1	Director US Army Aviation Research and Technology Activity Ames Research Center Moffett Field, CA 94035-1099
1	Commander US Army Missile Command ATTN: AMSMI-RD-CS-R (DOC) Redstone Arsenal, AL 35898-5010
1	Commander US Army Tank-Automotive Command ATTN: AMSTA-TSL (Technical Library) Warren, MI 48397-5000
1	Director US Army TRADOC Analysis Command ATTN: ATAA-SL White Sands Missile Range, NM 88002-5502
(Class. only) 1	Commandant US Army Infantry School ATTN: ATSH-CD (Security Mgr.) Fort Benning, GA 31905-5660
(Unclass. only) 1	Commandant US Army Infantry School ATTN: ATSH-CD-CSO-OR Fort Benning, GA 31905-5660
1	Air Force Armament Laboratory ATTN: AFATL/DLODL Eglin AFB, FL 32542-5000
	Aberdeen Proving Ground
2	Dir, USAMSAA ATTN: AMXSY-D AMXSY-MP, H. Cohen
1	Cdr, USATECOM ATTN: AMSTE-TD
3	Cdr, CRDEC, AMCCOM ATTN: SMCCR-RSP-A SMCCR-MU SMCCR-MSI
1	Dir, VLAMO ATTN: AMSLC-VL-D

No. of
Copies Organization

1 General Dynamics
Land Systems Division
ATTN: David Strimling
PO Box 2045
Warren, MI 48090

1 LTV Aerospace and Defense Company
ATTN: Phil Brown
Magdy Riscalla, Mailstop EM28
PO Box 225907
Dallas, TX 75265

1 Honeywell Defense Systems Division
ATTN: John Post
10400 Yellow Circle Drive
Minetonka, MN 55343

Aberdeen Proving Ground

Dir, USAMSAA
ATTN: AMXSY-G,
W. Brooks
B. Siegal
B. Goulet
R. Mezan
G. Comstock
L. Harrington
T. Ruth
M. Ritondo
P. Smyers
K. Tarquini
E. Walker
J. Graham
AMXSY-R, E. Hilkemeyer
AMXSY-A, J. Meredith
J. Hennessey
R. Mirabelle
AMXSY-C, R. Sandmeyer