

## **FFI RAPPORT**

### **A STOCHASTIC TWO-COMPONENT MATERIAL MODEL: DOCUMENTATION OF AN IMPLEMENTATION AS FORTRAN 90 SUBROUTINES IN AUTODYN**

SOLENG Harald H

**FFI/RAPPORT-2001/01089**



FFIBM/766

Approved

Kjeller 23 February 2001



Bjarne Haugstad

Director of Research

**A STOCHASTIC TWO-COMPONENT MATERIAL MODEL: DOCUMENTATION  
OF AN IMPLEMENTATION AS FORTRAN 90 SUBROUTINES IN AUTODYN**

SOLENG Harald H

FFI/RAPPORT-2001/01089

**FORSVARETS FORSKNINGSINSTITUTT**  
**Norwegian Defence Research Establishment**  
P O Box 25, NO-2027 Kjeller, Norway


**FORSVARETS FORSKNING SINSTITUTT (FFI)**  
**Norwegian Defence Research Establishment**

P O BOX 25  
 NO-2027 KJELLER, NORWAY

**UNCLASSIFIED**

SECURITY CLASSIFICATION OF THIS PAGE  
 (when data entered)

**REPORT DOCUMENTATION PAGE**

|  |   |                                      |
|--|---|--------------------------------------|
| 1) PUBL/REPORT NUMBER<br><br>FFI/RAPPORT-2001/01089<br><br>1a) PROJECT REFERENCE<br><br>FFIBM/766  | 2) SECURITY CLASSIFICATION<br><br>UNCLASSIFIED<br><br>2a) DECLASSIFICATION/DOWNGRADING SCHEDULE<br><br>--   | 3) NUMBER OF PAGES<br><br>51         |
| 4) TITLE<br>A STOCHASTIC TWO-COMPONENT MATERIAL MODEL: DOCUMENTATION OF AN IMPLEMENTATION AS FORTRAN 90 SUBROUTINES IN AUTODYN<br>EN STOKASTISK TOKOMPONENT MATERIALMODELL: DOKUMENTASJON AV EN IMPLEMENTASJON SOM FORTRAN 90 SUBROUTINER I AUTODYN  |   |                                      |
| 5) NAMES OF AUTHOR(S) IN FULL (surname first)<br>SOLENG Harald H   |   |                                      |
| 6) DISTRIBUTION STATEMENT<br>Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)   |   |                                      |
| 7) INDEXING TERMS<br>IN ENGLISH:<br>a) <u>Compound material</u><br>b) <u>Stochastic material model</u><br>c) <u>Simulation</u><br>d) _____<br>e) _____<br><br>IN NORWEGIAN:<br>a) <u>Komposittmateriale</u><br>b) <u>Stokastisk materialmodell</u><br>c) <u>Simulering</u><br>d) _____<br>e) _____<br><br>THESAURUS REFERENCE:                                     |   |                                      |
| 8) ABSTRACT<br><br>We implement a two-component material model using subroutines in Autodyn-2D version 4.1.13. The code described by this document has been written in noweb. Both the computer code in Fortran 90 and Matematica as well as the L <sup>A</sup> T <sub>E</sub> X source of this document are automatically generated from noweb source code files. |   |                                      |
| 9) DATE<br><br>23 February 2001  | AUTHORIZED BY<br><small>This page only</small><br><br>Bjarne Haugstad | POSITION<br><br>Director of Research |

ISBN 82-464-0517-9

**UNCLASSIFIED**

SECURITY CLASSIFICATION OF THIS PAGE  
 (when data entered)



## CONTENTS

|          | Page   |    |
|----------|--|----|
| 1        | Introduction                                     | 11 |
| 1.1      | Objective  | 11 |
| 1.2      | Organization of this document                    | 12 |
| 2        | Implementation in Autodyn                        | 12 |
| 2.1      | The initial distribution of rocks                | 12 |
| 2.2      | Setting material initial conditions using exedit | 12 |
| 2.2.1    | Overview of exedit                               | 13 |
| 2.2.2    | The initialization chunk                         | 15 |
| 2.2.3    | Reading input                                    | 15 |
| 2.2.4    | The kernel loop                                  | 16 |
| 2.2.5    | Inside the modification loop                     | 17 |
| 2.2.6    | File output for visualization                    | 20 |
| 2.3      | Implementation of rock properties                | 21 |
| 2.4      | Subroutines file layout                          | 21 |
| 3        | Running the program                              | 22 |
| 4        | Steel projectile penetrating into concrete       | 23 |
| 4.1      | Projectile material and geometric data           | 23 |
| 4.2      | Target data                                      | 24 |
| 4.3      | Results  | 26 |
| 4.3.1    | No gravel  | 26 |
| 4.3.2    | 5 mm gravel                                      | 26 |
| 4.3.3    | 10 mm gravel                                     | 28 |
| 5        | Conclusion                                       | 30 |
| <br>     |  |    |
| APPENDIX |  |    |
| A        | Global module                                    | 33 |
| B        | Random module                                    | 33 |
| B.1      | The uniform distribution                         | 33 |
| B.2      | The normal distribution                          | 34 |
| B.3      | The exponential distribution                     | 35 |
| B.4      | The Poisson distribution                         | 35 |
| B.4.1    | Case A: large expectation value                  | 37 |
| B.4.2    | Case B: small expectation value                  | 39 |

|     |                                    |    |
|-----|------------------------------------|----|
| B.5 | The Weibull distribution           | 40 |
| B.6 | Initialize random number generator | 40 |
| C   | Sorting module                     | 41 |
| D   | Generating seed file               | 43 |
| D.1 | Implementation                     | 43 |
| D.2 | Interactive usage                  | 43 |
| D.3 | Usage in batch                     | 44 |
| E   | The makefile                       | 44 |
| E.1 | Makefile definitions               | 44 |
| E.2 | Makefile rules                     | 46 |
| E.3 | Keywords to the makefile           | 47 |
|     | References                         | 49 |
|     | Distribution list                  | 51 |

**LIST OF TABLES**

|     |                                      |    |
|-----|--------------------------------------|----|
| 4.1 | Projectile data.                     | 24 |
| 4.2 | Target background material data.     | 25 |
| 4.3 | Target density data.                 | 25 |
| 4.4 | Target background yield stress data. | 25 |





**LIST OF FIGURES**

|      |  |    |
|------|--|----|
| 4.1  | Impact situation and penetrator geometry. The nose geometry is given by the radius of curvature $R$ and the angle $\alpha$ . | 23 |
| 4.2  | Pressure dependence of target properties.  | 26 |
| 4.3  | Initial situation and the final velocity with no gravel.   | 27 |
| 4.4  | 10% 5 mm gravel and the final velocity.  | 27 |
| 4.5  | 20% 5 mm gravel and the final velocity.  | 28 |
| 4.6  | 30% 5 mm gravel and the final velocity.  | 29 |
| 4.7  | 50% 5 mm gravel and the final velocity.  | 29 |
| 4.8  | 10% 10 mm gravel and the final velocity.   | 30 |
| 4.9  | 20% 10 mm gravel and the final velocity.   | 31 |
| 4.10 | 30% 10 mm gravel and the final velocity.   | 31 |
| 4.11 | 50% 10 mm gravel and the final velocity.   | 32 |



# A STOCHASTIC TWO-COMPONENT MATERIAL MODEL: DOCUMENTATION OF AN IMPLEMENTATION AS FORTRAN 90 SUBROUTINES IN AUTODYN

## 1 INTRODUCTION

Consider ordinary concrete. It consists of cement mixed with sand or gravel. It is well known that the properties of the concrete depends strongly on the mixture. It is also well known that for smaller calibers, the effects of inhomogeneties in the mixture can have strong effects on the penetration properties. For instance, we often find curved penetrator trajectories inconsistent with a symmetric impact into a homogeneous target.

### 1.1 Objective

In this report we are concerned with software which takes into account the inhomogeneity of concrete and its effects on penetration. In order to study such effects we have formulated a simplified two-component model of concrete in which the gravel is assumed to consisting of spherical objects with a given constant size. Both the cement and the gravel component are modelled using the *same* material. The gravel components differ only by having a rescaled yield strength. The target material is initialized by placing these “gravel” spheres randomly according to a predefined volume density. After this stochastic initialization we perform an ordinary deterministic penetration simulation. The algorithm we want to implement can be expressed in terms of the pseudocode in Algorithm 1.1.

---

#### Algorithm 1.1 Sketch of algorithm

---

```

1  draw number of rocks from a Poisson distribution
   randomly place rock centers
   for each cell
   {
5   decide material type and fill
   }
   evolve simulation

```

---

The model is admittedly very simple, and hence, it is not able to take into account the following known material properties:

1. the size and shape of gravel objects should follow a certain distribution
2. the gravel objects should be modelled using a different material model than used in the background, hence allowing also a different mass density in these objects
3. the interaction between gravel and cement could vary

Despite these shortcomings, we believe that the inhomogeneous stochastic concrete model serves to illustrate the importance of random inhomegeneities in concrete.

## 1.2 Organization of this document

This document is organized as follows. After sketching the general ideas in this introductory section, the details of the algorithm are worked out in section 2. The method is implemented in Fortran 90 as user-defined subroutines for Autodyn [1]. For further details about the literate programming style used in this paper and the use of Autodyn subroutines, please consult Ref. [2] where a stochastic flaw model was implemented to produce stochastic fractures. The current implementation shares much of its structure and coding details with the code of that report.

In section 3 we describe the user variables which have to be set interactively for the initialization file of Autodyn. In section 4 we present a test case used to verify the methods.

The implementation of the global variables, statistical functions and other utilities are explained and implemented as Fortran 90 modules in Appendices A–C. We use the *Mathematica* module described in Appendix D to seed the random number generator in the Fortran program.

## 2 IMPLEMENTATION IN AUTODYN

In this section we implement the details of Algorithm 1.1 sketched in section 1 as subroutines to Autodyn.

### 2.1 The initial distribution of rocks

Let the expected volume fraction of rocks in the compound be given by  $f_{\text{rocks}} \equiv \mathcal{V}_{\text{rocks}}/\mathcal{V}$  where  $\mathcal{V}_{\text{rocks}}$  is the total rock volume and  $\mathcal{V}$  is the total volume. For simplicity, we assume that all rocks are spherical balls with the same volume  $V_{\text{rock}}$ . Then the average number of rocks is

$$\mu = \frac{\mathcal{V}_{\text{rocks}}}{V_{\text{rock}}} = \frac{f_{\text{rocks}} \mathcal{V}}{V_{\text{rock}}}. \quad (2.1)$$

Not every target has the same number of rocks. We use a Poisson distribution to draw the number of rocks per target. Thus the number of rocks are drawn from the probability density function

$$p(n, \mu) = \frac{\mu^n e^{-\mu}}{n!}. \quad (2.2)$$

This is exactly the probability density function implemented in Section B.4.

When the number of rocks have been computed we have to distribute them randomly in the target geometry.

### 2.2 Setting material initial conditions using exedit

The standard way to implement custom initial conditions in Autodyn is to modify the `exval` subroutine. Unfortunately, this subroutine can only be called interactively through a

“fill” operation. Hence, in a statistical setting we would like to be able to run a large number of simulations in batch.

As a work-around Richard Clegg at Century Dynamics proposed to use the `exedit` function instead. This function is called at the end of a cycle, as specified under the `Modify Global Edit User Cycles` menu of Autodyn.

### 2.2.1 Overview of `exedit`

We develop the 2D and 3D implementations in parallel using C preprocessor directives to distinguish between the two versions.<sup>1</sup>

```
(exedit)≡
  subroutine exedit

    use sort
    use random
    use kindf
    use mdgrid
    use rundef
    use global ! Global variables

    implicit none

    integer (int4) :: i, imn, imx, &
                    j, jmn, jmx, ijk, ios, &
                    n, m, p, qq

    #ifdef threeD
    integer (int4) :: k, kmn, kmx, r
    real (real8)   :: z, znew
    #endif
    logical        :: stochastic
    real           :: mu, rockvolume
    real (real8)   :: density, radius, &
                    x, y, xnew, ynew
    character(len=10) :: material_name
    character(len=1024) :: comment

    at_end_of_zeroth_cycle : &
    if (ncycle==0) then
        <initialize>
        <read material data>
        <write log>
        <do if rocks>
            write(*,*)
```

<sup>1</sup>At the time of writing Autodyn 3D is at version 3 whereas Autodyn 2D is at version 4. The two versions are implemented in Fortran 77 and Fortran 90, respectively. We decided to wait for version 4 of Autodyn 3D and implement the two cases as a single source code.

```

write (*,*) " Modifying ",namsub(nsub)
write(*,*)

! Compute expectation mu=<V>
! Note that subvl(nsub) does not contain
! the volume of the void.

mu=density*subvl(nsub)
write (*,*) " Expected rock volume: ", mu

! Compute actual rock volume
rockvolume=real(random_Poisson(mu, .true.))
write (*,*) " Target rock volume: ", &
           rockvolume

mu=0.0
do
  ! First draw random position in i, j,
  ! and k (if 3D)
  i=random_uniform_integer(1,imax)
  j=random_uniform_integer(1,jmax)

  #ifdef threeD
    k=random_uniform_integer(1,kmax)
    ijk=ijkset(i,j,k)
  #else
    ijk=ijset(i,j)
  #endif

  ! Now we have the ijk index and we can
  <modify cell no n and its neighbours>

  if (mu>rockvolume) exit ! 'cause we're done
end do
write (*,*) " Simulated rock volume:", mu
write (*,'(af5.2a)') &
           " Simulated rock density: ", &
           100.0*mu/subvl(nsub), "%"
write (*,*)
  <write file "initial.dat">
<end if rocks>
end if at_end_of_zeroth_cycle
write(*,'(a)') " Starting simulation."
write(*,*) ""
return
end subroutine exedit

```

The above pseudo-code implementation will now be spelled out in more detail.

### 2.2.2 The initialization chunk

The purpose of this code chunk is simply to write out a welcome screen and to initialize the random number generator. Depending on the compiler, this initialization requires different number of integers in the file "seed.dat".

```

<initialize>≡
write(*, '(a)') " "
write(*, '(aa)') " *****", &
  "*****"
write(*, '(aa)') " * " , &
  " * "
write(*, '(aa)') &
  " * " Autocomp v. 2.0 " , &
  " * "
write(*, '(aa)') &
  " * " by Harald H. Soleng " , &
  " * "
write(*, '(aa)') &
  " * " Harald.Soleng@adinfinitum.no " , &
  " * "
write(*, '(aa)') " * " , &
  " * "
write(*, '(aa)') &
  " * " March, 2001 " , &
  " * "
write(*, '(aa)') " * " , &
  " * "
write(*, '(aa)') &
  " * " This is an extension of Autodyn. It " , &
  " * " contains a " * "
write(*, '(aa)') &
  " * " stochastic compound material model." , &
  " * "
write(*, '(aa)') " * " , &
  " * "
write(*, '(aa)') " *****", &
  "*****"
write(*, *)
write(*, *)
stochastic=.true. ! initial value
call seed_random_number() ! read file "seed.dat"

```

### 2.2.3 Reading input

After initialization of the random number generator, we read material data from a file. The file should contain



1. the relative volume density of rocks
2. the expected radius of the rocks
3. the
4. the strength ratio of rocks to background
5. the name of the subgrid to be filled with rocks

```

<read material data>≡
open(50,file='material.dat',status="old",iostat=ios)
if (ios==0) then
  read(50,*) comment, density, radius, strength, material_name
  density = density/100.0
else
  write(*,*)
  write(*,*) "FATAL ERROR: Cannot read ", &
             "'material.dat'. Aborting."
  write(*,*)
  stop
end if
close(50)

<write log>≡
write(*,'(aa)') &
  "   Material:      ", &
  material_name
write(*,'(af10.4a)') &
  "   density of: ", &
  density*100.0,"%"
write(*,'(af10.4)') &
  "   radius:      ", &
  radius
if (density==0.0) then
  stochastic=.false.  ! Nothing more to do
  write(*,*)
  write(*,'(aa)') " WARNING: Density ",&
    "vanishes in 'material.dat'. No"
  write(*,'(aa)') "   stochastic ",&
    "rocks are present, and the model is"
  write(*,'(aa)') "   equivalent ",&
    "to a standard Autodyn model."
  write(*,*)
end if

```

#### 2.2.4 The kernel loop

The beginning logical compound expressions is:

```

<do if rocks>≡
  stochastic_model : &
  if (stochastic) then
    do nsub = 1, numsub
      call getsub
      SPH: &
      if (nproc==7) then ! SPH
        write(*,*)
        write(*,*) "FATAL ERROR: Program ", &
          "not written for SPH"
        write(*,*) "Aborting."
        stop
      else
        if (namsub(nsub).eq.material_name) then
and the end is:
<end if rocks>≡
      end if
    end if SPH
  end do
end if stochastic_model

```

### 2.2.5 Inside the modification loop

Now we put in the gravel. Since two materials cannot be put in the same subgrid, during the `exedit` process, we tag the rock cells using a user-variable as a label. In the code chunk below we perform this tagging for the kernel cell and its neighbours where the neighbourhood is computed using the rock radius.

First let us define `<rock>` as a shorthand for user variable 01:

```

<rock>≡
  var01(ijk)

```

Next we need a test to check the Euclidean distance from the kernel node to the neighbour node.

```

<if greater than radius>≡
  xnew = xn(ijk)
  ynew = yn(ijk)
  #ifdef threeD
    znew=zn(ijk)
  #endif
  if (sqrt((x-xnew)**2.0+(y-ynew)**2.0 &
    #ifdef threeD
      +(z-znew)**2.0 &
    #endif
    ).gt.radius)

```

It returns true if the distance is greater than the rock radius.

```

<modify cell no n and its neighbours>≡
! First check that we are not in the void
if (den(ijk) > 0.0 .and. &
    <rock> < 0.5) then
    <rock> = 1.0
    x=xn(ijk)
    y=yn(ijk)
#ifdef threeD
        z=zn(ijk)
#endif

! Finding min in i-direction
m=i
do
    #ifdef threeD
        ijk=ijkset(m,j,k)
    #else
        ijk=ijset(m,j)
    #endif
    <if greater than radius> exit    ! rock too big
    if (m==1) exit                  ! at grid boundary
    m=m-1
end do
imn=m

! Finding max in i-direction
m=i
do
    #ifdef threeD
        ijk=ijkset(m,j,k)
    #else
        ijk=ijset(m,j)
    #endif
    <if greater than radius> exit    ! rock too big
    if (m==imax) exit              ! at grid boundary
    m=m+1
end do
imx=m

! Finding min in j-direction
m=j
do
    #ifdef threeD
        ijk=ijkset(i,m,k)
    #else
        ijk=ijset(i,m)
    #endif

```

```

        <if greater than radius> exit    ! rock too big
        if(m==1) exit                  ! at grid boundary
        m=m-1
    end do
    jmn=m

    ! Finding max in j-direction
    m=j
    do
        #ifdef threeD
            ijk=ijkset(i,m,k)
        #else
            ijk=ijset(i,m)
        #endif
        <if greater than radius> exit    ! rock too big
        if (m==jmax) exit              ! at grid boundary
        m=m+1
    end do
    jmx=m

    #ifdef threeD
    ! Finding min in k-direction
    m=k
    do
        ijk=ijkset(i,j,m)
        <if greater than radius> exit    ! rock too big
        if (m==1) exit                  ! at grid boundary
        m=m-1
    end do
    kmn=m

    ! Finding max in k-direction
    m=k
    do
        ijk=ijkset(i,j,m)
        <if greater than radius> exit    ! rock too big
        if (m==kmax) exit              ! at grid boundary
        m=m+1
    end do
    kmx=m
    #endif

    ! Loop over neighbours
    do p=imn, imx
        do qq=jmn, jmx
            #ifdef threeD
                do r=kmn, kmx

```

```

        ijk=ijkset(p,qq,r)
    #else
        ijk=ijset(p,qq)
    #endif
    <if greater than radius> then
        continue
    else
        <rock> = 1.0
        mu=mu+voln(ijk)
    end if
    #ifdef threeD
    end do
    #endif
end do
end do
end if

```

### 2.2.6 File output for visualization

This code chunk is included with external visualization in mind. If Autodyn's own visualizations are sufficient, this chunk can be omitted without loss of functionality. The node positions and rock values are written to an ascii file for later processing in visualization programs.

```

<write file "initial.dat">≡
open(unit=100,file="initial.dat", status="replace")
#ifdef threeD
    write(100,'(4i8)') imax, jmax, kmax, 0
#else
    write(100,'(3i8a)') imax, jmax, 0
#endif
do i=1, imax
    do j=1, jmax
        #ifdef threeD
            do k=1, kmax
                ijk=ijkset(i,j,k)
                write(100,'(3f10.5i3)') xn(ijk), &
                    yn(ijk), zn(ijk), int(<rock>)
            #else
                ijk=ijset(i,j)
                write(100,'(f10.5af10.5i3)') xn(ijk), &
                    " ",yn(ijk), int(<rock>)
            #endif
        #endif
    #ifdef threeD
    end do
    #endif
end if

```

```

    end do
  end do
close(100)

```

### 2.3 Implementation of rock properties

Cells with the nonzero  $\langle rock \rangle$  value gets an increased yield strength depending on the rock strength.

```

 $\langle exyld \rangle \equiv$ 
  subroutine exyld(pres, tt1, tt2, tt3, xmut, &
                 epst, epsd, tempt, yieldt, ifail)

  use mdgrid
  use ijknow
  use global

  implicit none

  integer (int4), intent(inout):: ifail
  integer (int4) :: ijk
  real (real8), intent(in):: pres, tt1, tt2, tt3, &
    xmut, epst, epsd, tempt
  real (real8), intent(out):: yieldt

  #ifdef threeD
  ijk=ijkset(inow, jnow, know)
  #else
  ijk=ijset(inow, jnow)
  #endif

  yieldt = yieldt+ $\langle rock \rangle$ *(strength-1.0)*yieldt

  return
end subroutine exyld

```

### 2.4 Subroutines file layout

The aforementioned subroutines (as well as unmodified dummy subroutines which are described in the Autodyn User Subroutines Tutorial [1]) come together in a single Fortran 90 source file.

```

 $\langle autosub.f90 \rangle \equiv$ 
   $\langle exedit \rangle$  ! replaces #include "exedit.f90"
   $\langle exyld \rangle$  ! replaces #include "exyld.f90"

```

! The following files contain the default usersub functions

```

#include "exale.f90"
#include "exbulk.f90"
#include "excomp.f90"
#include "excrck.f90"
#include "exdam.f90"
#include "exeos.f90"
#include "exerod.f90"
#include "exfail.f90"
#include "exfails.f90"
#include "exflow.f90"
#include "exload.f90"
#include "expor.f90"
#include "exsave.f90"
#include "exshr.f90"
#include "exsie.f90"
#include "exstif.f90"
#include "exstr.f90"
#include "extab.f90"
#include "exval.f90"
#include "exvel.f90"
#include "exzone.f90"

```

### 3 RUNNING THE PROGRAM

When compiling Autodyn with the user subroutines described in this paper using the Makefile given in Appendix E, we get an executable file called `autocomp-2D-1.0`. Autocomp is a customized version of Autodyn containing a stochastic flaw model.

In order to run Autodyn with the subroutines specified in this report, the user has to set up

1. a computational model containing a two-component target (*i.e.*, cement and rock). In the input model, only the cement should be included. The rocks are added by the program.
2. an ascii file named `seed.dat` containing the number of random seed integers required by the particular version of the Fortran 90 compiler used by his or her computer platform. This file should be updated before calling the program again.
3. an ascii file names `material.dat` containing the following material data
  - (a) rock volume ratio in per cents
  - (b) the rock radius in milli meters
  - (c) the rock strength factor
  - (d) the two-component subgrid name

In the present implementation only one two-component material type can be used in a single run.

In addition, the following user variables must be activated `var01` user variable must be activated. It represents the rock fraction of a cell. This user variable is activated by using the **Modify Global Options UserVar. Add/Mod** menu of Autodyn. Next, the `exedit` subroutine must be called at the end of the zeroth cycle. This must be specified under the **Modify Global Edit User Cycles** menu of Autodyn.

#### 4 STEEL PROJECTILE PENETRATING INTO CONCRETE

The problem is the following. A long rod steel projectile hits a large block of concrete. The situation short before impact is depicted in Fig. 4.1 (a). In this report we limit ourselves to a

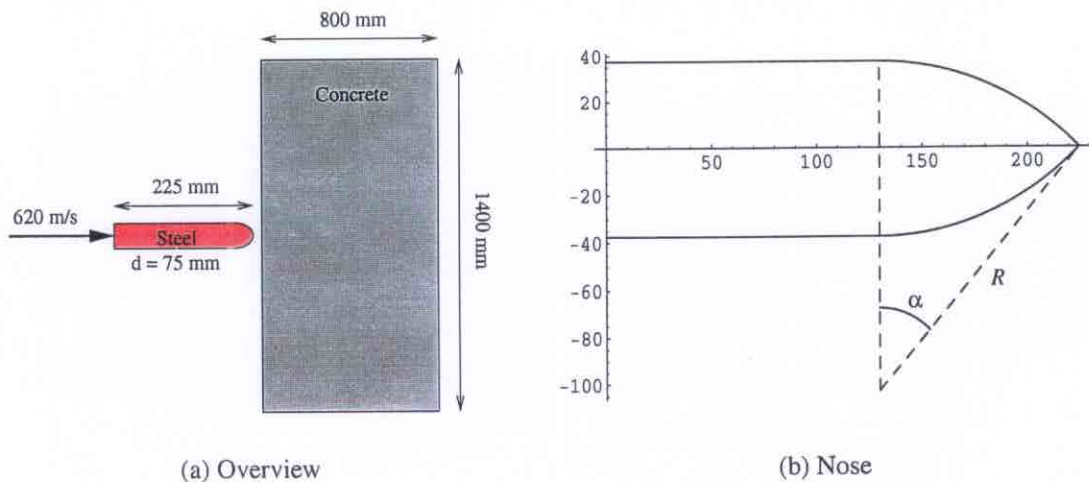


Figure 4.1: Impact situation and penetrator geometry. The nose geometry is given by the radius of curvature  $R$  and the angle  $\alpha$ .

two-dimensional test case, *i.e.*, we assume a plane symmetric impact situation. Obviously, due to the inhomogeneity in the target, an axial symmetric simulation is no good in this problem. Hence, the illustration in Fig. 4.1 shows a horizontal cross section of the situation. For the penetrator we use a Lagrange grid. The target is modelled using an Euler grid.

##### 4.1 Projectile material and geometric data

The projectile has the geometrical parameters as shown in Fig. 4.1 (b). Thus the projectile has a  $\ell/d$  of 3. Its nose is ogive. As expected, the experiments gave very little deformation of the projectiles. A full account of the relevant projectile data is given in Table 4.1.



Table 4.1: Projectile data.

| <i>Steel penetrator</i>                        |                         |
|--|-------------------------|
| <i>Equation of state</i>                       | linear                  |
| <i>Reference density <math>\rho</math></i>     | 7.324 g/cm <sup>3</sup> |
| <i>Bulk modulus <math>K</math></i>             | 175 GPa                 |
| <i>Shear modulus <math>G</math></i>            | 80.8 GPa                |
| <i>Yield stress <math>Y_0</math></i>           | 1.00 GPa                |
| <i>Impact velocity</i>                         | 620.00 m/s              |
| <i>Length</i>                                  | 225.00 mm               |
| <i>Diameter</i>                                | 75.00 mm                |
| <i>Nose radius of curvature <math>R</math></i> | 140.25 mm               |
| <i>Angle <math>\alpha</math></i>               | 42.8935°                |

## 4.2 Target data

The target consists of 150 MPa concrete with no reinforcement. It is modelled by a porous equation of state where both the density and the yield limit depend on pressure. Concrete is a mixture of cement and sand or gravel. In the current simulations, the mixture is modelled as a homogeneous background into which we place a certain density of spherical objects. For simplicity we use the same material model for both the cement and the gravel components of the concrete. The gravel objects differ from the background by having an increased yield limit. These objects are placed randomly in the target during initialization. In the simulations below, the gravel has a yield limit which is *four* times higher than that of the background. Their radius is set to 2.5 mm or 5.0mm. The volume density is set to values between 0 and 50%. Alternatively, the yield limit could have been reduced for the background as we add gravel so as to keep the average yield limit constant. In that way one would separate the effect of target inhomogeneity from the effect of increasing the mean yield limit of the target.

Obviously this material model is an oversimplification, yet it is far more realistic than a homogeneous material model since it captures the essential stochastic inhomogeneity in concrete. As a result of these inhomogeneities, the model is able to reproduce the stochastic deviations from straight projectile trajectories seen in experiments.

All material data in Tables 4.2–4.4 refer to the properties of the background. The gravel is assumed to have exactly the same material data, except for the yield limit which is a factor of four higher.

The concrete is modelled by a porous equation of state where the density depends on the pressure, *cf.* Table 4.3. Due to the internal friction, also the yield limit increases with increasing pressure, *cf.* Table 4.4.

For easier visualization, we have plotted the contents of Tables 4.3 and 4.4 in Figure 4.2.

Table 4.2: Target background material data.

| <b>C150 material data</b>      |                        |
|--------------------------------|------------------------|
| Equation of state              | porous                 |
| Reference density $\rho_{ref}$ | 3.05 g/cm <sup>3</sup> |
| Initial density $\rho_0$       | 2.77 g/cm <sup>3</sup> |
| Solid sound speed              | 3000 m/s               |
| Porous sound speed             | 2720 m/s               |
| Bulk modulus $K$               | 28.43 GPa              |
| Shear modulus $G$              | 25.00 GPa              |
| Young's modulus $E$            | 58.00 GPa              |
| Diameter                       | 1400 mm                |
| Thickness                      | 800 mm                 |

Table 4.3: Target density data.

| <b>C150 density data</b> |          |                       |
|--------------------------|----------|-----------------------|
| Density $\rho(p)$        | pressure | density               |
| $p_1$ and $\rho_1$       | 150 MPa  | 2.7 g/cm <sup>3</sup> |
| $p_2$ and $\rho_2$       | 250 MPa  | 2.8 g/cm <sup>3</sup> |
| $p_3$ and $\rho_3$       | 500 MPa  | 2.9 g/cm <sup>3</sup> |
| $p_4$ and $\rho_4$       | 800 MPa  | 3.0 g/cm <sup>3</sup> |
| $p_5$ and $\rho_5$       | 1100 MPa | 3.1 g/cm <sup>3</sup> |
| $p_6$ and $\rho_6$       | 1500 MPa | 3.2 g/cm <sup>3</sup> |
| $p_7$ and $\rho_7$       | 2000 MPa | 3.3 g/cm <sup>3</sup> |

Table 4.4: Target background yield stress data.

| <b>C150 yield stress data</b> |          |              |
|-------------------------------|----------|--------------|
| Yield stress $Y(p)$           | pressure | yield stress |
| $p_1$ and $Y_1$               | 0 MPa    | 20.0 MPa     |
| $p_2$ and $Y_2$               | 25 MPa   | 113.1 MPa    |
| $p_3$ and $Y_3$               | 50 MPa   | 150.9 MPa    |
| $p_4$ and $Y_4$               | 100 MPa  | 202.5 MPa    |
| $p_5$ and $Y_5$               | 200 MPa  | 273.0 MPa    |
| $p_6$ and $Y_6$               | 300 MPa  | 326.5 MPa    |
| $p_7$ and $Y_7$               | 400 MPa  | 369.3 MPa    |
| $p_8$ and $Y_8$               | 600 MPa  | 441.2 MPa    |
| $p_9$ and $Y_9$               | 800 MPa  | 500.8 MPa    |
| $p_{10}$ and $Y_{10}$         | 1000 MPa | 552.6 MPa    |

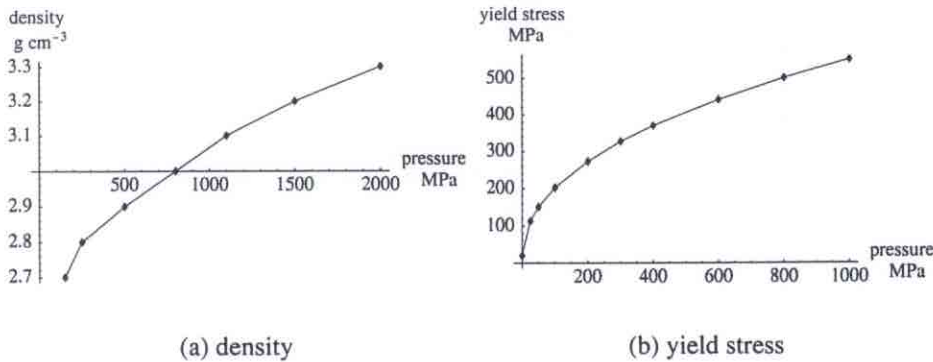


Figure 4.2: Pressure dependence of target properties.

### 4.3 Results

We have carried out a number of simulation of this situation varying the gravel volume densities between 0 and 50% and the gravel rock diameter between 5 and 20 mm. As expected, adding material with a higher yield limit reduces the penetration depth. From the principle of minimal action, it is clear that a projectile hitting an inhomogeneous material will seek the path of least resistance. Hence, the projectile trajectory should deviate from a straight line as the projectile is perturbed by the gravel.

#### 4.3.1 No gravel

Let us first look at the case with no gravel at all. In the case with no gravel depicted in Figure 4.3 (a), the projectile perforates the target. The slight deviation from perfect symmetry seen in Figure 4.3 (b) can only be explained by numerical perturbations (rounding errors). Due to an inherent instability initial rounding errors may grow into visible effects later in the simulation.

The immense size of the crater is due to the use of planar symmetry. By a simple volume scaling argument [2] where the volume of the projectile trajectory is compared to the target volume, it can be seen that the planar symmetric case corresponds (approximately) to an axial symmetric case with a small target. In addition, the target has no support at the boundaries in our example.

#### 4.3.2 5 mm gravel

Let us now look at various densities of 5 mm gravel. In Figure 4.4 (a), we see a random distribution of 10% gravel in the background. Initially, the rock density is discrete. Hence, in each cell the rock parameter should be either zero or unity. Unfortunately, we have not managed to turn off the smoothing of the Autodyn plots. Another unphysical smoothing occurs because the rock variable is transported between cells in the Euler grid. The transport effect is more serious because it affects the computations. It can be seen in the pictures of the initial rock distribution which are taken at about 400 cycles; in front of the projectile one can see a semi-circular blurred region. After adding 10% gravel by

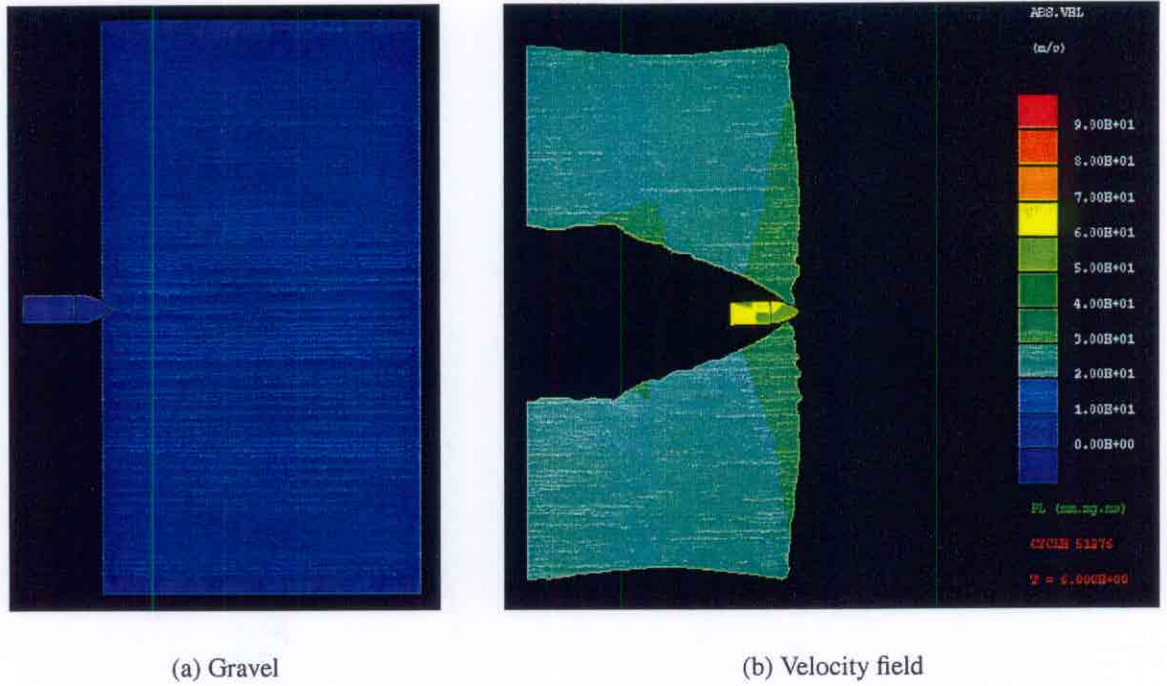


Figure 4.3: Initial gravel distribution of 0% by volume and the final absolute velocity.

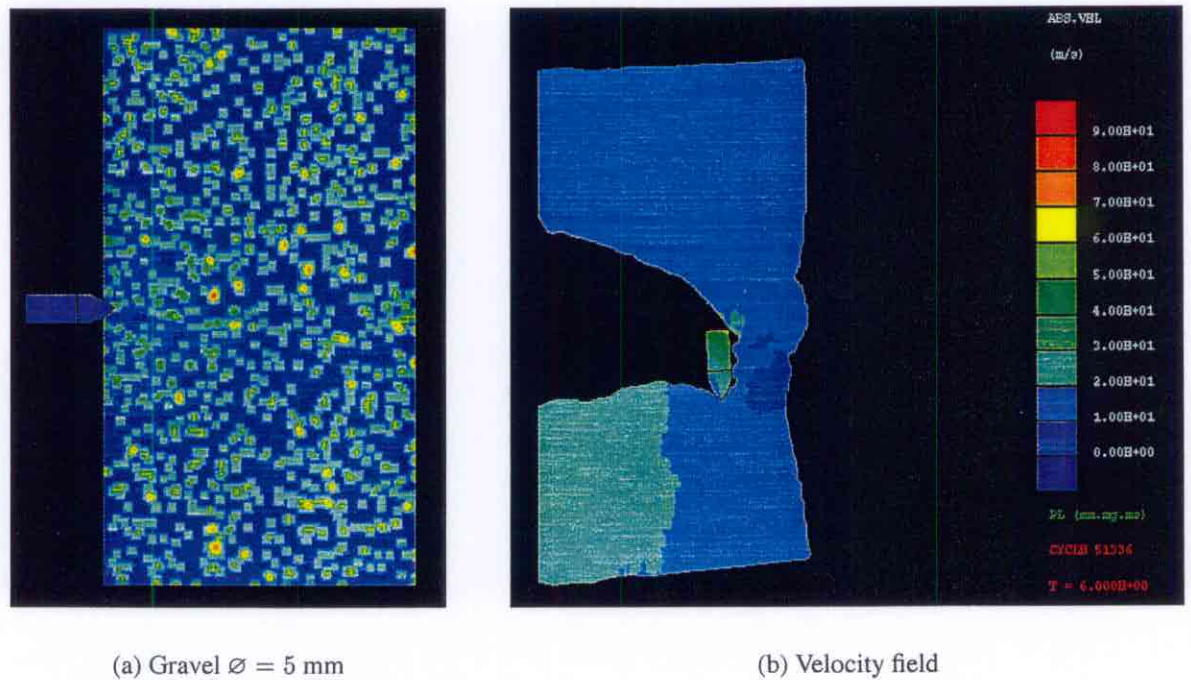
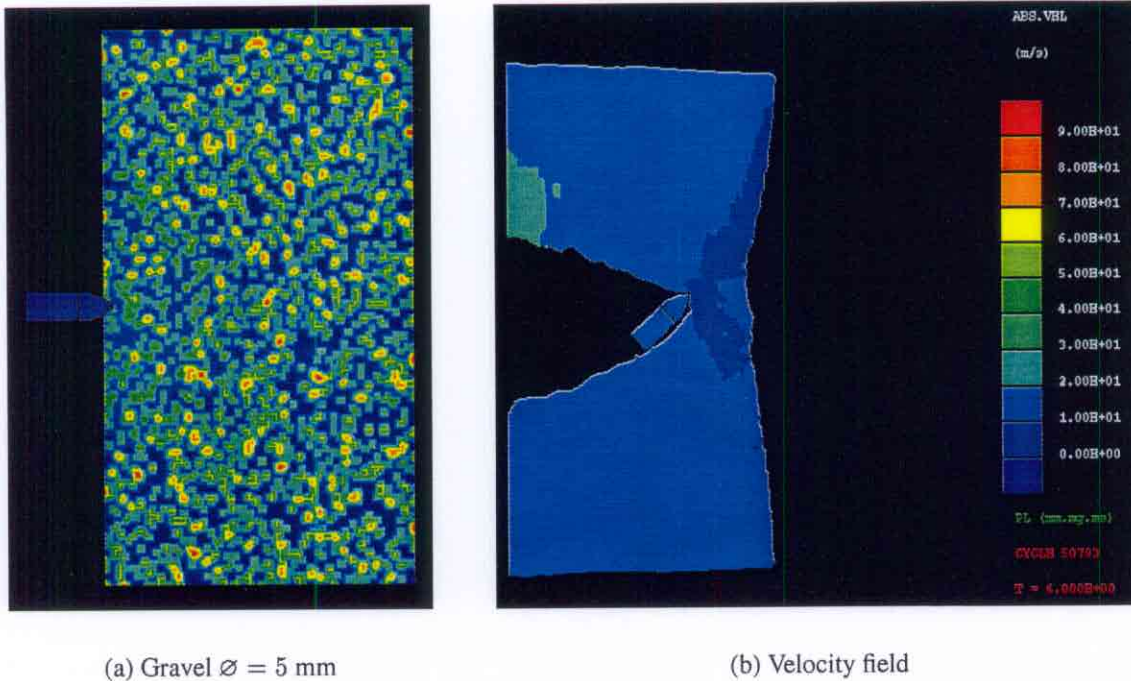


Figure 4.4: Initial gravel distribution of 10% by volume and the final absolute velocity.

volume, the target is not perforated. Instead the projectile is strongly deflected. A minute analysis of the final state in Figure 4.4 revealed that the penetrator is about to ricochet. With a single simulation it is impossible to say how likely this result is. Perhaps the strong deflection happened just by coincidence?



(a) Gravel  $\varnothing = 5$  mm

(b) Velocity field

Figure 4.5: Initial gravel distribution of 20% by volume and the final absolute velocity.

Doubling the fraction of gravel as seen in Figure 4.5 did not decrease the penetration depth significantly. This may be a coincidence. In the 10% case, the penetrator was strongly deflected. Its sideways motion certainly helped stopping the projectile faster than if the deflection had been more modest. Without a statistical analysis of a large number of such simulations we cannot say much about the expected penetration depths for different proportions of cement and gravel.

In Figure 4.6 we have increased the gravel volume fraction to 30%. Here the penetration depth is significantly reduced and the penetrator deflection is modest.

In the last simulation (Figure 4.7) the gravel fraction was 50%. Here we see that the penetrator only reaches halfway through the target. This is just what one should expect: adding a significant portion of objects with a high yield limit has a strong effect on the resistance of the target.

#### 4.3.3 10 mm gravel

Let us now increase the diameter of the gravel rocks to 10 mm. On the one hand, the expected number of rocks in the projectile trajectory decreases. On the other hand, each rock produces a larger resistance to penetration. It is not obvious what the net result turns out to be.

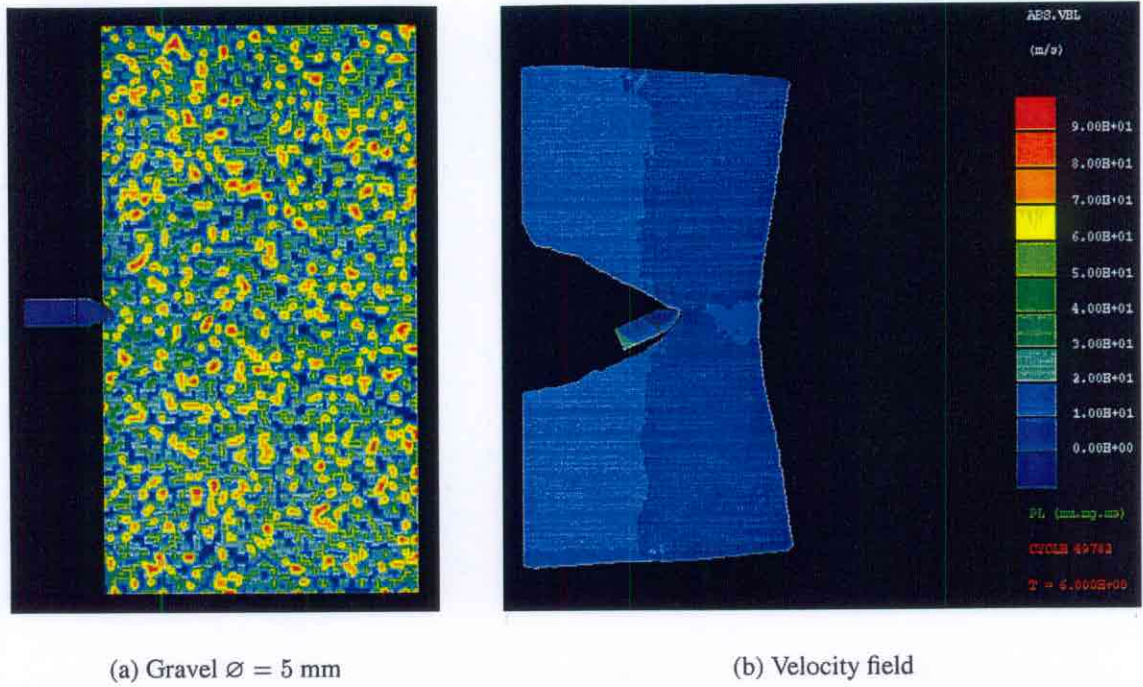


Figure 4.6: Initial gravel distribution of 30% by volume and the final absolute velocity.

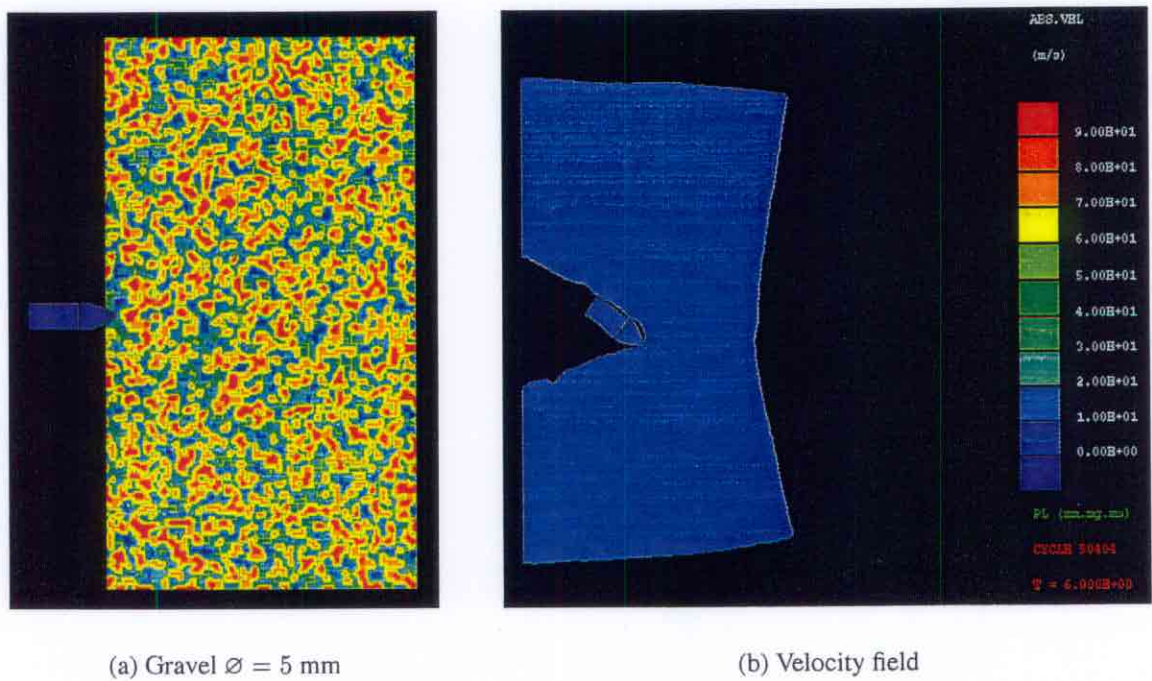


Figure 4.7: Initial gravel distribution of 50% by volume and the final absolute velocity.

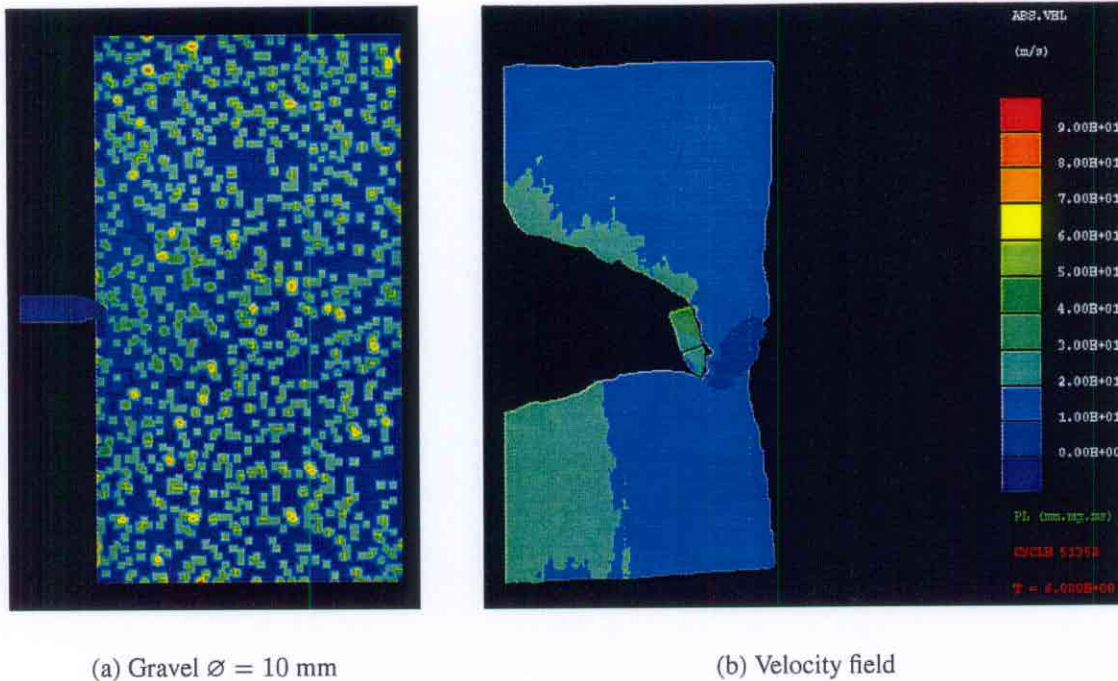


Figure 4.8: Initial gravel distribution of 10% by volume and the final absolute velocity.

Figure 4.8 shows the result of adding 10% 10 mm gravel. Comparing the result to that of 5% 10 mm gravel in Figure 4.4 we find that contrary to the expectation the penetration depth appears to be slightly deeper for the larger rocks. However, the data set is not large enough to say if the difference is significant.

Going up to 20% gravel, the difference in penetration depth is small for 5 mm (Fig. 4.5) and 10 mm gravel (Fig. 4.9).

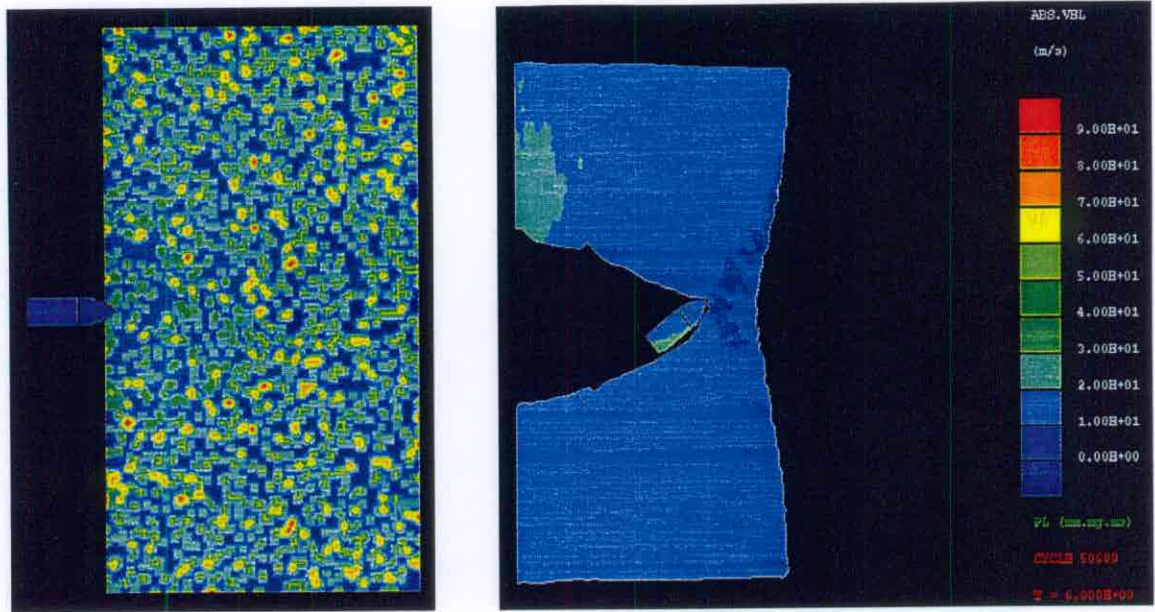
Also at 30%, we still don't see a significant difference between 5 mm (Fig. 4.6) and 10 mm gravel (Fig. 4.10).

At 50%, the high yield limit of the gravel really kicks in. We find that the projectile is deformed. It seems to be a larger difference between the cases with 5 mm (Fig. 4.7) and 10 mm gravel (Fig. 4.11), but the difference in penetration depth could be a consequence of random deflections.

## 5 CONCLUSION

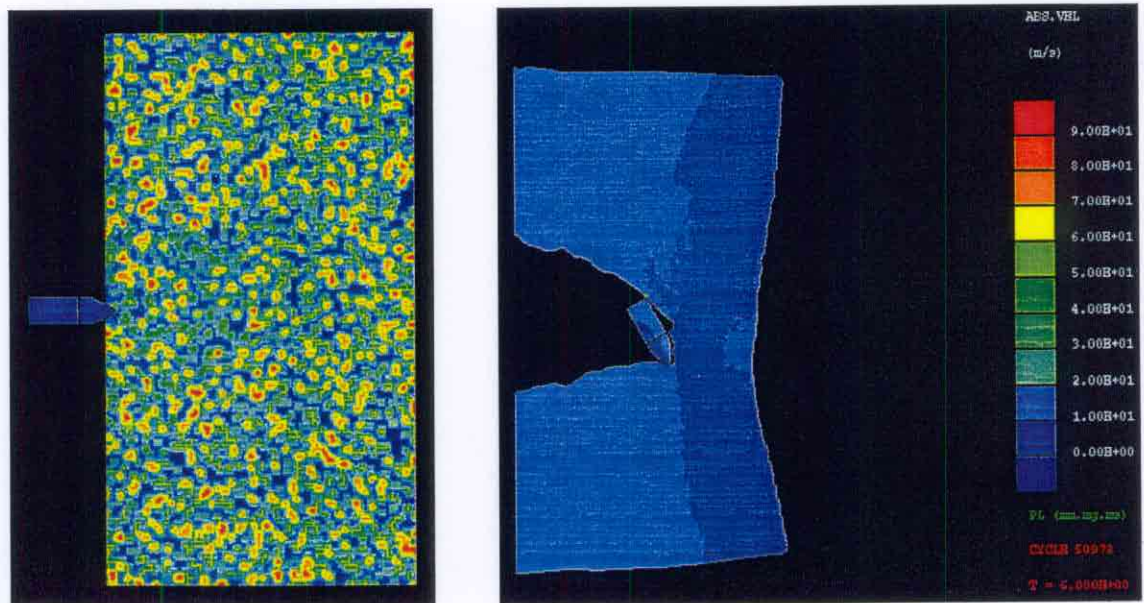
A very simple stochastic two-component model of concrete has been implemented. It captures the essential inhomogeneity property of concrete. The plane symmetric 2D simulations have shown that inhomogeneities with a diameter of 5 mm or 10 mm (where yield strength is increased by a factor of four) has a strong effect on the deflection of 75 mm ammunition.

This is in agreement with experience. Projectile trajectories tend to be complex in concrete. Of course, the model needs to be refined before it can be used on realistic problems, but

(a) Gravel  $\varnothing = 10$  mm

(b) Velocity field

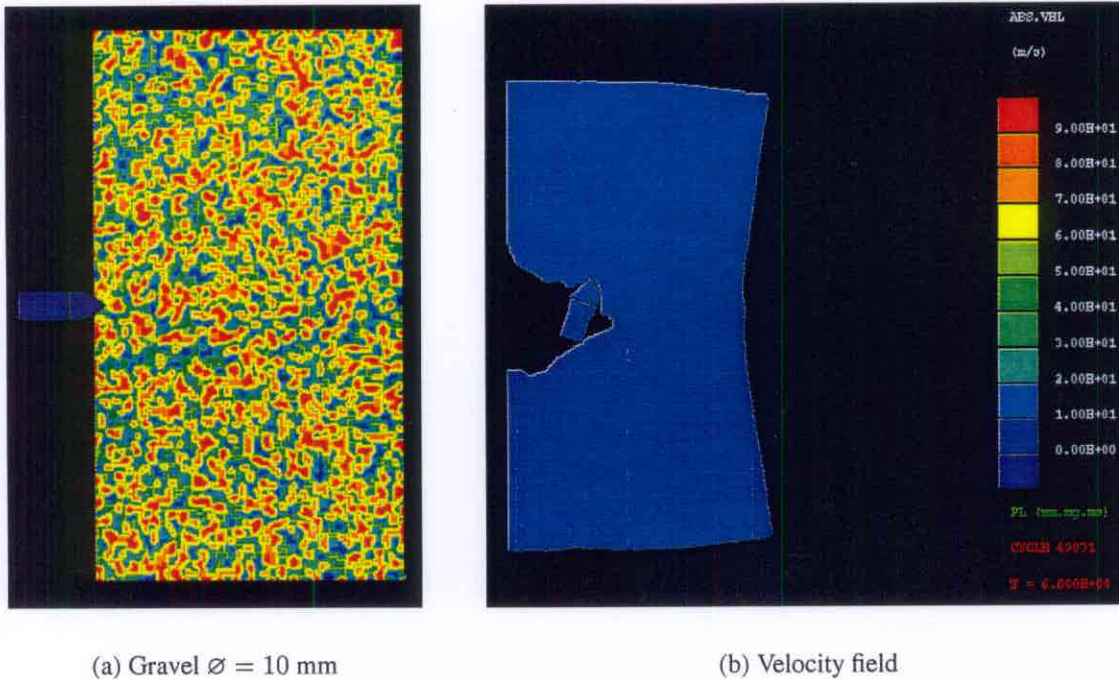
Figure 4.9: Initial gravel distribution of 20% by volume fraction and the final velocity.

(a) Gravel  $\varnothing = 10$  mm

(b) Velocity field

Figure 4.10: Initial gravel distribution of 30% by volume and the final velocity.



(a) Gravel  $\varnothing = 10$  mm

(b) Velocity field

Figure 4.11: Initial gravel distribution of 50% by volume fraction and the final velocity.

already at this preliminary stage, effects that are masked in a continuum material model are illustrated.

The main steps to refine this approach should be as follows:

1. Include the possibility to use two different material models for the cement and concrete. To achieve this, the `exva1` subroutine in Autodyn must be used for the initialization.
2. Be able to include stones with different size
3. Run the code in full 3D. This will need version 4 of Autodyn, since version 3 is not coded in Fortran 90. Also, parallel processing may be necessary to avoid very long run-times (also available in version 4).
4. Finally, implementation in SPH may be needed to avoid certain inherent numerical problems in the Euler or Lagrange processors.

## APPENDIX

### A GLOBAL MODULE

In this appendix we define a global module. It is used to hold a global variable to be accessed by different user subroutines.

```
<global.f90>≡
  module global
  implicit none
  real :: strength
  end module global
```

### B RANDOM MODULE

In this appendix we implement a simple random number module. It can draw pseudo-random numbers from the normal (Gaussian), the exponential, the Poisson, and the Weibull distributions.

```
<random.f90>≡
  module random
  implicit none
  real, private, parameter :: zero = 0.0, half = 0.5, &
                                one = 1.0

  contains
  <draw from the uniform distribution>
  <draw from the normal distribution>
  <draw from the exponential distribution>
  <draw from the Poisson distribution>
  <draw from the Weibull distribution>
  <initialize the random number generator>
  end module random
```

#### B.1 The uniform distribution

We define the `random_uniform()` function in the random module. It returns a uniformly distributed pseudo-random real number.

```
<draw from the uniform distribution>≡
  function random_uniform(lower,upper) result(fu_val)

  real, intent(in) :: lower, upper
  real              :: fu_val, ran

  call random_number(ran)
  fu_val= (upper-lower)*ran+lower
```

```

return
end function random_uniform

```

The `random_uniform_integer()` function returns a uniformly distributed pseudo-random integer.

```

<draw from the uniform distribution>≡
function random_uniform_integer(lower,upper) result(fu_val)

integer, intent(in) :: lower, upper
integer              :: fu_val
real                :: ran

call random_number(ran)
fu_val= int((1+upper-lower)*ran+lower)
#ifdef debug
if (fu_val<lower) then
    write(*,*) "Random number below boundary. Aborting."
    stop
else if (fu_val>upper) then
    write(*,*) "Random number above boundary. Aborting."
    stop
end if
#endif
return
end function random_uniform_integer

```

## B.2 The normal distribution

The normal distribution (normalized Gaussian) distribution is given by the probability density function

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}. \quad (\text{B.1})$$

In the random module we define the function `random_normal()`. It returns a normally distributed pseudo-random number with zero mean and unit variance. The algorithm uses the ratio of uniforms method of A. J. Kinderman and J. F. Monahan augmented with quadratic bounding curves.

```

<draw from the normal distribution>≡
function random_normal() result(fn_val)

real                :: fn_val, u, v, x, y, q
real, parameter    :: s = 0.449871, t = -0.386595, &
                    a = 0.19600, b = 0.25472,    &
                    r1 = 0.27597, r2 = 0.27846

```

```

do
  call random_number(u)
  call random_number(v)
  v = 1.7156 * (v - half)
  x = u - s
  y = abs(v) - t
  q = x**2 + y*(a*y - b*x)
  if (q < r1) exit
  if (q > r2) cycle
  if (v**2 < -4.0*log(u)*u**2) exit
end do

fn_val = v/u

return
end function random_normal

```

### B.3 The exponential distribution

The exponential distribution with scale parametre equal unity is given by the probability density function

$$p(x) = e^{-x}. \quad (\text{B.2})$$

The function `random_exponential()` returns a pseudo-random number  $x \in [0, \infty)$  from a negative exponential distribution using inversion.

*(draw from the exponential distribution)*≡

```

function random_exponential() result(fn_val)

real :: fn_val, r

do
  call random_number(r)
  if (r > zero) exit
end do

fn_val = -log(r)

return
end function random_exponential

```

### B.4 The Poisson distribution

The Poisson distribution is defined by the discrete probability density function

$$p(n, \mu) = \frac{\mu^n e^{-\mu}}{n!}. \quad (\text{B.3})$$

$\mu$  is the mean value.

The `random_Poisson(mu, first)` returns a pseudo-random number from the Poisson distribution with mean  $\mu$ . This implementation is based on an algorithm of Ahrens and Dieter [3]. It was translated to Fortran 90 by Alan Miller from RANLIB.

The interface consists of

```
<i/o variables>≡
  real, intent(in)      :: mu
  logical, intent(in)  :: first
  integer               :: ival
```

where `mu` is the expectation, `first` is a Boolean which is true for the first call. `ival` is the return value.

```
<draw from the Poisson distribution>≡
  function random_Poisson(mu, first) result(ival)
```

```
  <i/o variables>
  <local variables>
  <parametres>
```

```
  if (mu > 10.0) then
    <case A>
  else
    <case B>
  end if
```

```
  return
end function random_Poisson
```

```
<local variables>≡
  real                :: b1, b2, c, c0, c1, c2, c3, &
                       del, difmuk, e, fk, fx, fy, g, &
                       omega, px, py, t, u, v, x, xx
  real, save         :: s, d, p, q, p0
  integer            :: j, k, kflag
  logical, save     :: full_init
  integer, save     :: l, m
  real, save        :: pp(35)
```

```
<parametres>≡
  real, parameter :: a0 = -.5, a1 = .3333333, &
                       a2 = -.2500068, a3 = .2000118, &
                       a4 = -.1661269, a5 = .1421878, &
                       a6 = -.1384794, &
                       a7 = .1250060
  real, parameter :: fact(10) = &
    (/ 1., 1., 2., 6., 24., &
      120., 720., 5040., &
      40320., 362880. /)
```

B.4.1 Case A:  $\mu > 10$ 

If  $\mu$  has changed, we recalculate  $s$ ,  $d$ , and  $l$ . The Poisson probabilities  $p_k$  exceed the discrete normal probabilities  $f_k$  whenever  $k \geq m(\mu)$ .  $l = \text{ifix}(\mu - 1.1484)$  is an upper bound to  $m(\mu)$  for all  $\mu \geq 10$ .

```

<case A>≡
  if (first) then
    s = sqrt(mu)
    d = 6.0*mu*mu
    l = mu - 1.1484
    full_init = .false.
  end if

```

Get a normal sample  $g$

```

<case A>+≡
  g = mu + s*random_normal()
  g_positive: &
  if (g > 0.0) then
    ival = g

```

and accept immediatly if it is large enough

```

<case A>+≡
  if (ival>=l) return

```

Squeeze acceptance and draw a uniform sample  $u$ .

```

<case A>+≡
  fk = ival
  difmuk = mu - fk
  call random_number(u)
  if (d*u >= difmuk*difmuk*difmuk) return
end if g_positive

```

Recalculation of parametres if necessary. The quantities  $b_1$ ,  $b_2$ ,  $c_3$ ,  $c_2$ ,  $c_1$  and  $c_0$  are for the discrete normal probabilities  $f_k$ .

```

<case A>+≡
  if (.not. full_init) then
    omega = .3989423/s           ! .3989423 = (2*pi)**(-.5)
    b1 = .4166667E-1/mu         ! .416667E-1 = 1./24.
    b2 = .3*b1*b1
    c3 = .1428571*b1*b2         ! .1428571 = 1./7.
    c2 = b2 - 15.*c3
    c1 = b1 - 6.*b2 + 45.*c3
    c0 = 1. - b1 + 3.*b2 - 15.*c3
    c = .1069/mu               ! majorization by
                                ! the 'hat' function

    full_init = .true.
  end if
  if (g<0.0) go to 50

```

Calling 'subroutine F' (setting the flag for correct return):

```
<case A>+≡
  kflag = 0
  go to 70
```

Implementing the rare quotient acceptance case.

```
<case A>+≡
  40 if (fy-u*fy <= py*exp(px-fx)) return
```

Exponential sample and sample  $t$  from the LaPlace 'hat' (if  $t \leq -0.6744$  then  $p_k < f_k$  for all  $\mu \geq 10$ ).

```
<case A>+≡
  50 e = random_exponential()
  call random_number(u)
  u = u + u - one
  t = 1.8 + sign(e, u)
  if (t <= (-.6744)) go to 50
  ival = mu + s*t
  fk = ival
  difmuk = mu - fk
```

Calling 'subroutine F' (setting the flag for correct return):

```
<case A>+≡
  kflag = 1
  go to 70
```

Hat acceptance (going back to exponential sample on rejection):

```
<case A>+≡
  60 if (c*abs(u) > py*exp(px+e) - fy*exp(fx+e)) go to 50
  return
```

### Subroutine F

This 'subroutine' calculates  $p_x$ ,  $p_y$ ,  $f_x$  and  $f_y$ . For values below 10 it uses factorials from the table.

```
<case A>+≡
  70 if (ival>=10) go to 80
  px = -mu
  py = mu**ival/fact(ival+1)
  go to 110
```

For  $ival \geq 10$  we use polynomial approximation  $a_0$ - $a_7$  for accuracy when advisable.

```
<case A>+≡
  80 del = .8333333E-1/fk ! .8333333E-1=1./12.
  del = del - 4.8*del*del*del
  v = difmuk/fk
  if (abs(v)>0.25) then
```

```

    px = fk*LOG(one + v) - difmuk - del
else
    px = fk*v*v* ((((((a7*v+a6)*v+a5) &
        *v+a4)*v+a3)*v+a2)*v+a1)*v+a0) - del
end if
py = .3989423/sqrt(fk)      ! .3989423=(2*pi)**(-.5)
110 x = (half - difmuk)/s
xx = x*x
fx = -half*xx
fy = omega* (((c3*xx + c2)*xx + c1)*xx + c0)
if (kflag <= 0) go to 40
go to 60

```

#### B.4.2 Case B: $\mu \leq 10$

Start new table and calculate  $p_0$  is necessary.

```

<case B>≡
  if (first) then
    m = max(1, int(mu))
    l = 0
    p = exp(-mu)
    q = p
    p0 = p
  end if

```

Draw a random number  $u$  from the uniform distribution.

```

<case B>+≡
  random_u: &
  do
    call random_number(u)
    ival = 0
    if (u <= p0) return

```

Do a table comparison until the end  $pp(l)$  of the  $pp$ -table of cumulative Poisson probabilities (for  $\mu = 10$ ,  $pp(9) = 0.458$ ).

```

<case B>+≡
  if (l == 0) go to 150
  j = 1
  if (u > 0.458) j = min(1, m)
  do k = j, l
    if (u <= pp(k)) go to 180
  end do
  if (l == 35) cycle

```

Create new Poisson probabilities  $p$  and their cumulatives  $q = pp(k)$ :

```

<case B>+≡
  150 l = l + 1

```



```

do k = 1, 35
  p = p*mu / k
  q = q + p
  pp(k) = q
  if (u <= q) go to 170
end do
l = 35
end do random_u
170 l = k
180 ival = k
return

```

### B.5 The Weibull distribution

The probability density function of the Weibull distribution is defined as

$$p(x, m) = mx^{m-1}e^{-x^m}. \quad (\text{B.4})$$

Given a positive definite input parameter  $m$ , the function `random_Weibull(m)` returns a pseudo-random number from the Weibull distribution

```

<draw from the Weibull distribution>≡
function random_Weibull(m) result(fn_val)

real, intent(in) :: m
real              :: fn_val

fn_val = random_exponential() ** (one/m)

return
end function random_Weibull

```

### B.6 Initialize random number generator

```

<initialize the random number generator>≡
subroutine seed_random_number()

integer              :: k, i, ios
integer, allocatable :: seed(:)

call random_seed(size=k)
allocate(seed(k))
write(*,*) ""
write(*,'(aa)') " Initializing random number ",&
  "generator using 'seed.dat'."

open(99,file="seed.dat",status="old",iostat=ios)

```

```

file_ok: &
if (ios==0) then
  read(99,*) seed
  write(*,'(a, i2, aa)') " Reading ", &
    k, " random number seeds ",&
    "from the file:"
  do i=1, k
    write(*,'(i8)',advance='no') seed(i)
    if (mod(i,7)==0 .or. i==k) then
      write(*,*)
    end if
  end do

  write(*,*) ""
  close(99)
  call random_seed(put=seed)
else
  write(*,*) ""
  write (*,'(aa)') " FATAL ERROR: Could not ",&
    " read file 'seed.dat'"
  write (*,'(a)') " Program aborts."
  write(*,*) ""
  stop
end if file_ok
deallocate(seed)
return
end subroutine seed_random_number

```

## C SORTING MODULE

This module implements a pedestrian sorting function for float arrays. Be warned: if you need a very effective sorting algorithm, then please look elsewhere!

```

<sort.f90>≡
  module sort
  implicit none
  contains
  <sort array of floats>
  end module sort

```

We define two sorting function to sort float numbers in positive:

```

<sort array of floats>≡
  subroutine floatsort(a, n)

```

```
integer, intent(in) :: n
real, intent(inout) :: a(n)
```

```
! Local variables
integer i, counter
real    x
```

```
do i = 2, n
  x = a(i)
  counter = i-1
  loop : &
  do
    if (counter < 1 .or. x >= a(counter)) then
      exit loop
    else
      a(counter+1) = a(counter)
      counter = counter-1
    end if
  end do loop
  a(counter+1) = x
end do
return
end subroutine floatsort
```

or negative order

*(sort array of floats)*+≡

```
subroutine floatsortr(a, n)
```

```
integer, intent(in) :: n
real, intent(inout) :: a(n)
```

```
! Local variables
integer i, counter
real    x
```

```
do i = 2, n
  x = a(i)
  counter = i-1
  loop : &
  do
    if (counter < 1 .or. x <= a(counter)) then
      exit loop
    else
      a(counter+1) = a(counter)
      counter = counter-1
    end if
  end do loop
```

```

    a(counter+1) = x
end do
return
end subroutine floatsortr

```

## D GENERATING SEED FILE

In the previous section we implemented an initialization function for the random number generator. This function reads a set of seeds from a file. Thus, we need a method to generate a new set of seeds prior to calling the initialization function.

### D.1 Implementation

Here we shall implement such a function in *Mathematica*.

```

⟨makeseed.m⟩≡
Module[{NumberOfSeeds, SeedRange,
        write, seeds, out},
        ⟨platform dependent code⟩
        ⟨general code⟩
        seeds (* For logging purposes *)]

```

The number of seed numbers as well as the minimal and maximal integers are platform dependent:

```

⟨platform dependent code⟩≡
NumberOfSeeds = 17;
SeedRange     = { -30000, 30000 };

```

Now we are ready to write down the general code

```

⟨general code⟩≡
write[x_][y_] := WriteString[x, ToString[y], " "];
seeds = Table[Random[Integer, SeedRange],
              {NumberOfSeeds}];
out = OpenWrite["seed.dat"];
Map[write[out], seeds];
Close[out];

```

This completes the *Mathematica* module needed to write a list of random integers to the file `seed.dat`.

### D.2 Interactive usage

Let us now see how to use this *Mathematica* module. If you are running *Mathematica* interactively, it can be called by giving the command

```
Get["makeseed`"]
```

at the `In[n]` prompt of *Mathematica*. This, however, is not very practical for our purposes so we shall describe the use in batch mode.

### D.3 Usage in batch

In batch mode, we call it by the shell command

```
math < makeseed.m > /dev/null
```

Here we have directed output to the Unix “black hole” at `/dev/null`. If one would like to keep a log file of all seed numbers one could instead use

```
math < makeseed.m >> seed.log
```

where `>>` means that output is appended to the file each time the command is called.

## E THE MAKEFILE

The program is compiled and linked using `make` and a `Makefile`. Using the `make` program we are able to make fortran source files, object files and linked executables from the `noweb` source files. We can also make `LaTeX`, `postscript` and `pdf` files for the documentation. The rules needed to perform these tasks are defined in the `Makefile`. It has the following layout:

```
<Makefile>≡
  <definitions>
  <rules>
  <keywords>
```

### E.1 Makefile definitions

```
<definitions>≡
COMPILER = fortran # Script calling f90 on "mammut"
CVERSION = 90
AUTODYN_2D_VERSION = 4.1.13
AUTODYN_3D_VERSION = 3.2.05
AUTOCOMP_VERSION = 1.0
PROGRAM_2D = autocomp-2D-$(AUTOCOMP_VERSION)
PROGRAM_3D = autocomp-3D-$(AUTOCOMP_VERSION)
LINKER = $(COMPILER) $(CVERSION)
FLAGS = +save +noshared +O3 +Oaggressive\
        +Oall +DA2.0 +cpp=yes +cpp_keep\
        -Ddebug -Dshearmod -Dbulkmod
2DFLAGS= $(FLAGS)
3DFLAGS= $(FLAGS) -DthreeD
LINKFLAGS=
```

```

MAIN      = compound
DOCDIR   = ../doc
MATHDIR  = ../math
BINDIR_2D = ../autodyn-$(AUTODYN_2D_VERSION)
BINDIR_3D = ../autodyn-$(AUTODYN_3D_VERSION)

INCDIR_2D = ../usrsub-2d
LIBDIR_2D = /user/hso/autodyn/2d4113_dp/usrsub

INCDIR_3D =
LIBDIR_3D = /user/hso/autodyn/3dv3205_1000k/usrsub
GKSDIR   = /user/hso/autodyn/gks

INCLUDE_2D = -I$(INCDIR_2D) -I$(LIBDIR_2D)
INCLUDE_3D = -I$(INCDIR_3D)

LIB_2D    = $(LIBDIR_2D)/admain2.o \
            $(LIBDIR_2D)/autodyn2.a \
            -L$(GKSDIR) -lgksflb -lgksw5300 -lgkswiss \
            -lgksgksm -lgksmsc -L/usr/lib/X11R5 -lX11 -lm

LIB_3D    = $(LIBDIR_3D)/admain3.o \
            $(LIBDIR_3D)/autodyn3.a \
            -L$(GKSDIR) -lgksflb -lgksw5300 -lgkswiss \
            -lgksgksm -lgksmsc -L/usr/lib/X11R5 -lX11 -lm

FORTRANSRC= global.nw random.nw sort.nw autosub.nw

MATHSRC    = makeseed.nw

MAINSRC    = $(MAIN).nw

TEXSRC     = running.nw\
            simulations.nw\
            makefile.nw\
            autosub.nw\
            $(MAINSRC)\
            $(MATHSRC)\
            $(FORTRANSRC)

TEXFILES   = ${TEXSRC:.nw=.tex}

FORTRANFILES = ${FORTRANSRC:.nw=.f90}

MATHFILES  = ${MATHSRC:.nw=.m}

```

```
O2DFILES = ${FORTRANSRC:.nw=.o2D}
O3DFILES = ${FORTRANSRC:.nw=.o3D}
.SUFFIXES: .nw .tex .dvi .ps .f90 .o2D .o3D .m .pdf
```

## E.2 Makefile rules

Let us now define the rules.

```
<rules>≡
.nw.f90:
    notangle -t8 -R$@ $< | \
    perl -e 'while(<>) { s/\s*#/#/; print; }' \
    | cpif $@
```

The program code is extracted using notangle. We are going to use a C preprocessor on the source before feeding it to the fortran compiler. Unlike the gnu C processor, the HP version of the C preprocessor suffers from the illness of requiring all preprocessor directives to start in the first column. To satisfy this need we pipe the code through a small perl program.

The *Mathematica* program and the  $\text{\LaTeX}$  sources are easily obtained by means of notangle and noweave::

```
<rules>+≡
.nw.m:
    notangle -t8 -R$@ $< | cpif $@

.nw.tex: ; noweave -n -delay $< | cpif $@; \
    rm -f $(MAIN).dvi
```

From the  $\text{\LaTeX}$  source files, we get the documentation by running latex and bibtex a few times:

```
<rules>+≡
.tex.dvi: $(TEXFILES)
    latex '\scrollmode \input "$$(MAIN)"; \
    if (\
        grep -s 'Citation' $(MAIN).log || \
        grep -s 'No file $(MAIN).bbl' $(MAIN).log ); \
    then \
        bibtex $(MAIN); \
        latex '\scrollmode \input "$$(MAIN)"; \
    fi; \
    while (\
        grep -s 'No file $(MAIN).toc' $(MAIN).log || \
        grep -s 'Rerun to get cross-references right'\
            $(MAIN).log ); \
    do latex '\scrollmode \input "$$(MAIN)"; \
    done
```

Equivalently for pdflatex:

```

<rules>+≡
.tex.pdf: $(TEXFILES)
    pdflatex '\scrollmode \input "$$(MAIN)"; \
    if (\
        grep -s 'Citation' $(MAIN).log || \
        grep -s 'No file $(MAIN).bbl' $(MAIN).log ); \
    then \
        bibtex $(MAIN); \
        pdflatex '\scrollmode \input "$$(MAIN)"; \
    fi; \
    while (\
        grep -s 'No file $(MAIN).toc' $(MAIN).log || \
        grep -s 'Rerun to get cross-references right'\
            $(MAIN).log ); \
    do pdflatex '\scrollmode \input "$$(MAIN)"; \
    done

```

With the dvi file in hand, we can make the post script using dvips:

```

<rules>+≡
.dvi.ps: $(TEXFILES) $(MAIN).dvi
    dvips -f $< | cpif $@

```

Compilation of the same code (distinguished by preprocessor directives) results in 2D and 3D object files.

```

<rules>+≡
.f90.o2D: $(FORTRANFILES)
    $(COMPILER)$ (CVERSION) $< -c \
    $(2DFLAGS) $(INCLUDE_2D) -o $@

.f90.o3D: $(FORTRANFILES)
    $(COMPILER)$ (CVERSION) $< -c \
    $(3DFLAGS) $(INCLUDE_3D) -o $@

```

### E.3 Keywords to the makefile

Linking the two sets of object codes leads to the 2D and 3D version of the program. The following keywords can be used in the make command:

```

<keywords>≡
2d: makefile $(O2DFILES)
    $(LINKER) $(LINKFLAGS) $(O2DFILES) $(LIB_2D) \
    -o $(PROGRAM_2D)
    cp $(PROGRAM_2D) ../autodyn-$(AUTODYN_2D_VERSION)/

3d: makefile $(O3DFILES)
    $(LINKER) $(LINKFLAGS) $(O3DFILES) $(LIB_3D)\

```



```

-o $(PROGRAM_3D)
cp $(PROGRAM_3D) ../autodyn-$(AUTODYN_3D_VERSION) /

program: 2d

all:    dir math ps fortran object program

dir:
    mkdir -p $(MATHDIR) $(DOCDIR)

tex:    $(TEXFILES)
    rm -f *.dvi

dvi:    $(TEXFILES) $(MAIN).dvi

ps:     tex $(MAIN).dvi $(MAIN).ps
    cp $(MAIN).ps $(DOCDIR) /

pdf:    tex rmpdf $(MAIN).pdf
    cp $(MAIN).pdf $(DOCDIR) /

rmpdf:
    rm -f $(MAIN).pdf

makefile:
    notangle -t8 -RMakefile makefile.nw \
    | cpif Makefile

fortran: $(FORTRANFILES)

math:    $(MATHFILES)
    cp $(MATHFILES) $(MATHDIR) /

object: $(O3DFILES) $(O2DFILES)

cleandoc: rmpdf
    rm -f *.dvi *.ps

clean:  cleandoc
    rm -f *.tex *.aux *.log *~ *.bbl *.blg *.toc \
    *.brf .out *.o2D *.o3D *.mod *.f90 *.i90

```

**References**

- [1] Century Dynamics Inc. (1999): *Autodyn, User Subroutines Tutorial*, 2333 San Ramon Valley Blvd., Suite 185, San Ramon, CA 94583, USA, fourth edn.
- [2] Soleng H H (2001): A stochastic SPH flaw model and the evolution of fractures: Documentation of an implementation as Fortran 90 subroutines in Autodyn, Tech. Rep. FFI/RAPPORT-2001/01090, Forsvarets Forskningsinstitut (Norwegian Defence Research Establishment), P.O. Box 25, NO-2027 Kjeller, Norway.
- [3] Ahrens J H, Dieter U (1982): Computer generation of Poisson deviates from modified normal distributions, *ACM Trans. Math. Software* **8**, no. 2, pp. 163–179.



## DISTRIBUTION LIST

**FFIBM**
**Dato: 23 February 2001**

|  |                           |  |                                    |
|--|---------------------------|--|------------------------------------|
| RAPPORTTYPE (KRYSS AV)<br><input checked="" type="checkbox"/> RAPP <input type="checkbox"/> NOTAT <input type="checkbox"/> RR                    | RAPPORT NR.<br>2001/01089 | REFERANSE<br>FFIBM/766/130   | RAPPORTENS DATO<br>23 februar 2001 |
| RAPPORTENS BESKYTTELSESGRAD<br><br>Unclassified  |                           | ANTALL EKS<br>UTSTEDT<br><br>40  | ANTALL SIDER<br><br>51             |
| RAPPORTENS TITTEL<br>A STOCHASTIC TWO-COMPONENT MATERIAL<br>MODEL: DOCUMENTATION OF AN<br>IMPLEMENTATION AS FORTRAN 90<br>SUBROUTINES IN AUTODYN |                           | FORFATTER(E)<br>SOLENG, Harald   |                                    |
| FORDELING GODKJENT AV FORSKNINGSSJEF:<br>                       |                           | FORDELING GODKJENT AV AVDELINGSSJEF:<br> |                                    |

**EKSTERN FORDELING**
**INTERN FORDELING**

| ANTALL | EKS NR | TIL              | ANTALL | EKS NR | TIL                        |
|--------|--------|------------------|--------|--------|----------------------------|
| 1      |        | FBT/S            | 14     |        | FFI-Bibl                   |
| 1      |        | v/Helge Langberg | 1      |        | Adm direktør/stabssjef     |
|        |        |                  | 1      |        | FFIE                       |
|        |        |                  | 1      |        | FFISYS                     |
|        |        |                  | 5      |        | FFIBM                      |
|        |        |                  | 7      |        | Eirik Svinsås, FFIBM       |
|        |        |                  | 3      |        | Harald Soleng, FFIBM       |
|        |        |                  | 1      |        | Jan Arild Teland, FFIBM    |
|        |        |                  | 1      |        | Henrik Sjøel, FFIBM        |
|        |        |                  | 1      |        | John F Moxnes, FFIBM       |
|        |        |                  | 1      |        | Ove Dullum, FFIBM          |
|        |        |                  | 1      |        | Svein E Martinussen, FFIBM |
|        |        |                  | 1      |        | Svein Rollvik, FFIS        |
|        |        |                  |        |        | FFI-veven                  |

FFI-K1

Retningslinjer for fordeling og forsendelse er gitt i Oraklet, Bind I, Bestemmelser om publikasjoner for Forsvarets forskningsinstitutt, pkt 2 og 5. Benytt ny side om nødvendig.