# Physical modelling of dynamic recrystallization of the cyclic peak type

by
Corentin Kastenbaum

Directed by
Dr.Daniel Crespo

Co-directed by
Dr.Eloi Pineda

Dr. José María Cabrera

Material science

November, 2015
Barcelona, Spain

Based on the Avrami equation a model to predict the cyclic behavior of dynamic re-crystallization of metals (particularly Cu) undergoing high temperature deformation at relatively low strain rates must be obtained.

(ABSTRACT)

Dynamic re-crystallization is a process of discontinuous crystallization during hot-working. This is a very interesting process that has been modeled in several ways. In this project I've tried to model the birth, increment, and extinction of crystallites by using only the Avrami equation. The Avrami equation is quite simple to write. It describes the occurrence of newborn crystallites during nucleation and growth processes. The equation establishes that the ratio of the real volume and the extended volume is equal to the fraction of free space. We will get more in detail in the chapter 2. While in a crystallization process the free space is simply the uncrystallized space, in re-crystallization processes it must be redefined as the space available for growth, regardless if it was previously crystallized. The project consisted in write the adequate model and a code to check its validity. I then made several tests with the final code, with variations on each parameters, and I gathered it in the chapter 3.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Theory of DRX

## 1.1 Occurrence of DRX in the experience

Dynamic re-crystallization (DRX) is a phenomenon that occurs in some metals during hot working. We consider that there is hot working for $T_H > 0.5$. This is the homologous temperature: $T_H = \frac{T}{T_f}$. Where $T_f$ stands for the temperature of fusion.

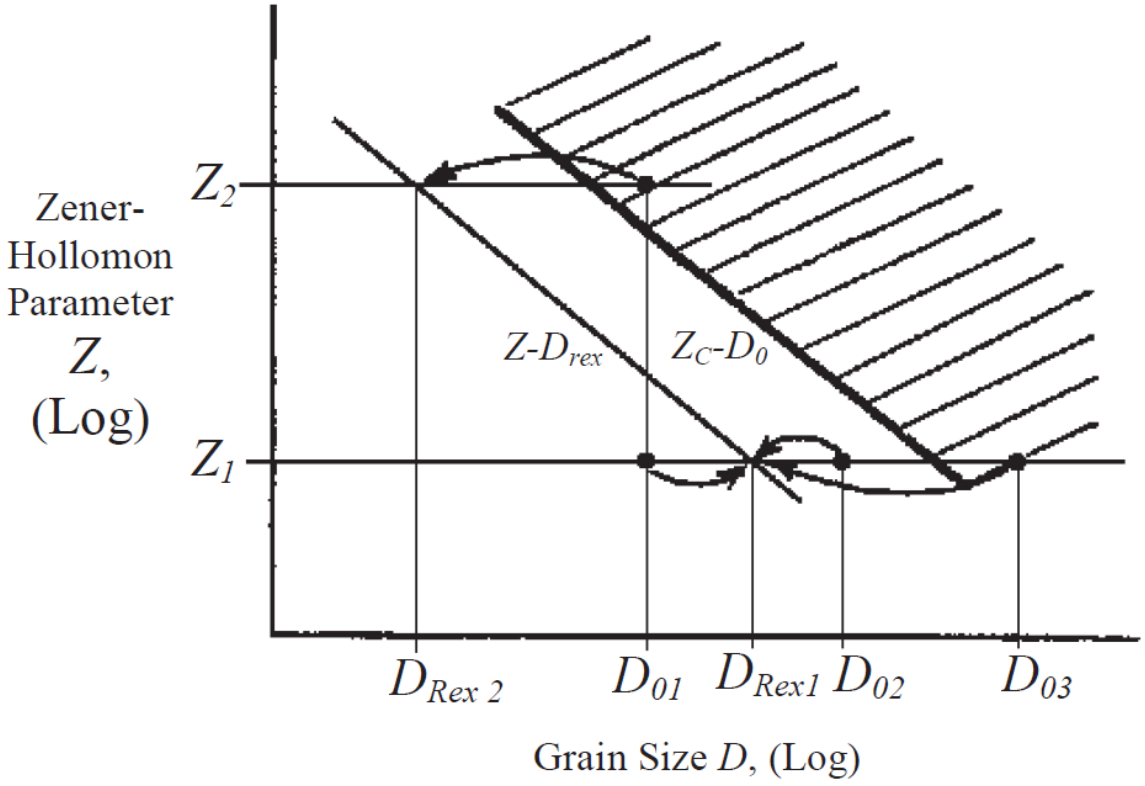
In hot working, there are two phenomenons that face each other: hardening with deformation, and softening due to DRX. Both phenomenons of hardening and softening are functions of strain rate $\dot{\varepsilon}$, temperature $T$, and stacking fault energy of the material $Q$.

The hardening comes from the athermal mechanism, that lead the dislocation to be stored. Indeed, dislocation motion permits the deformation to progress. After dislocation accumulation, some softening occurs by dynamic recovery, i.e. the reordering of dislocations within the material. Hwn materials posses low stacking fault energy a second softening mechanism, i.e, dynamic recrystallization, DRX, appears. Dynamic recovery appears to be the leading mechanism when stacking fault energy is high and deformation is quick. On the stress-strain curve, we can see that the curve always increase to an asymptotic value but never reach a local maximum. On the other side, when the stacking fault energy is low and the deformation rate is low enough, we get a DRX. It begins when the other mechanisms cannot store energy anymore.

There are two types of DRX: the one-peak type, and the cyclic-peak type. The two determining parameters are initial mean grain size $D_0$ , and Zener-Hollomon parameter:

$$Z = \dot{\varepsilon} \cdot exp(\frac{Q}{RT})$$

Figure 1.1: Line between multiple-peak type and one-peak type [1]

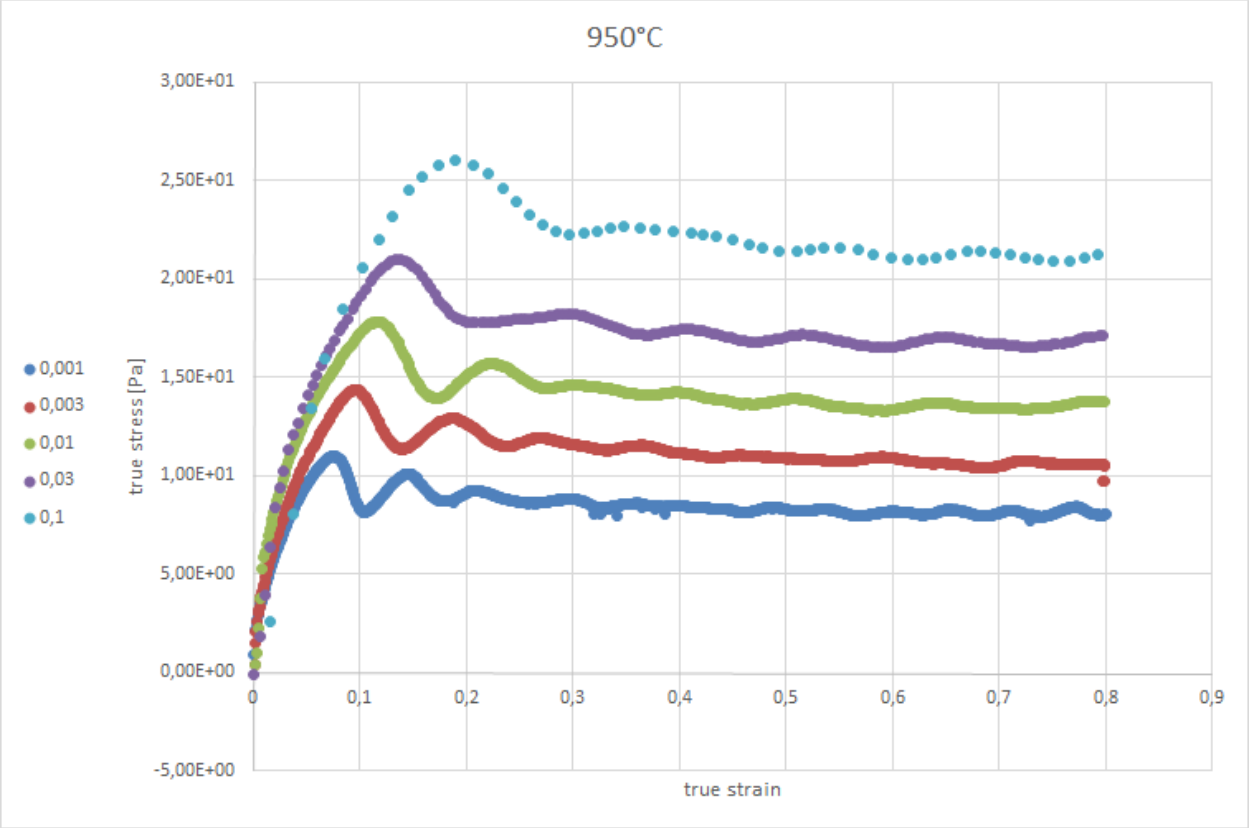



## 1.2 Determining multiple-peak DRX

Previously, we said that the DRX would occur when the energy couldn't be stored by strain hardening nor recovery. Then the grain becomes saturated in dislocation, that will agglomerate until new grain is formed. The multiple-peak type occurs when there is too much energy to store. In the figure 1.1, the medium initial grain size is confronted to Zener-Hollomon parameter. On the logarithmic scale, we put the conditions of the re-crystallisation and we observe that the medium size of the crystal will tend to align with the $Z - D_{rex}$ curve. When the initial point is in the shadow area, i.e. above the $Z_c - D_0$ curve, we observe a single-peak type. The $Z_c - D_0$ curve puts the limit between the single-peak type and the multiple-peak type. Above the line, the equilibrium of the crystal is reached really quick and only need one cycle.

On the other hand, when he conditions are not in the shadowed area, it correspond to a recrystallized grain size too big to store all of the energy in one cycle. This approach is the Relative-Grain-Size model and was developed by Montheillet and Jonas [1].

Figure 1.2: Effect of strain rate on DRX



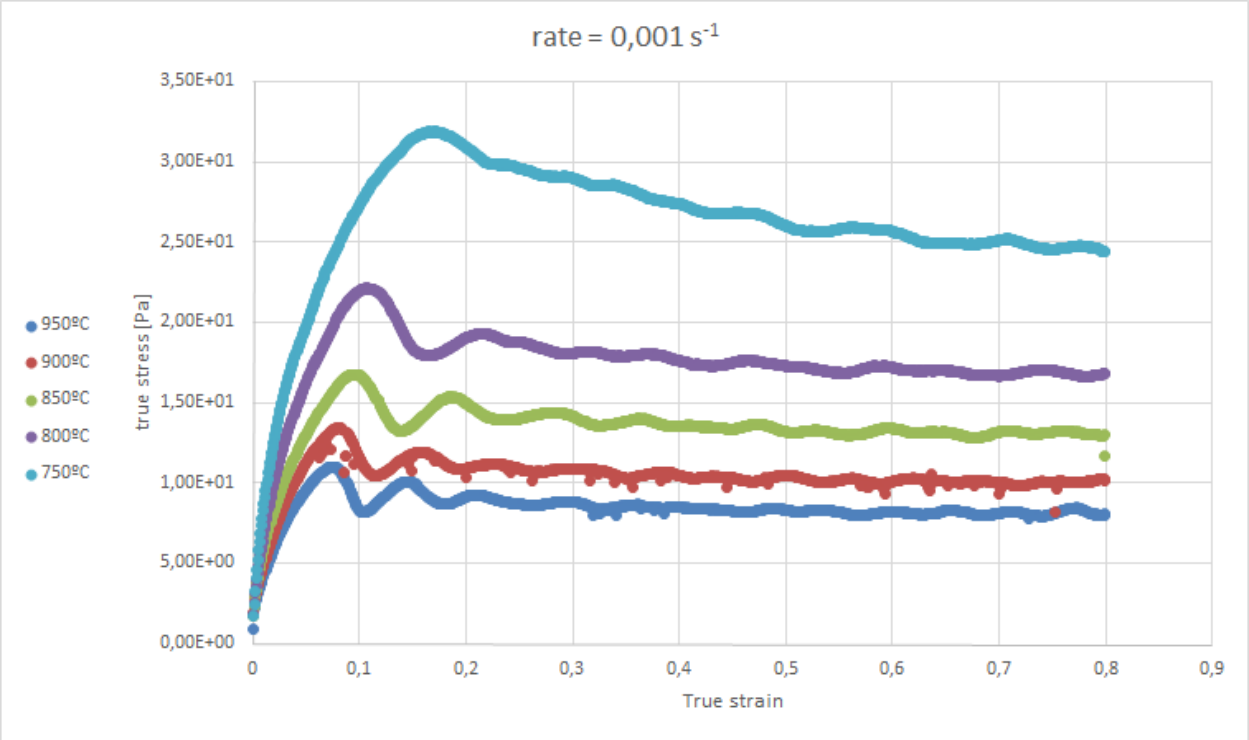

## 1.3 Stress-strain data

Here we will have a look at the data we got experimentally. In a first time, when we compare the stress-strain curves for different strain rate on the figure 1.2, we observe a higher stress limit for higher strain rates. Moreover, the number of cycles is more important toward the end of the curve.

In the figure 1.3, we emphasize the effect of temperature on the DRX curves. We see that the temperature acts in the same way as the strain rate. As the Zener-Hollomon is decreasing with the temperature, the oscillations decrease with temperature and the final stress value gets higher.

Those experimental observations will guide us through the modeling and the interpretations of the results.

Figure 1.3: Effect of temperature on DRX



# Chapter 2

# Avrami model

The Avrami equation describes the way the space is covered in a nucleation and growth process. Let's $V(t)$ be the volume of the crystallized phase at time $t$, and $V_{ext}(t)$ the volume of the extended phase at the same time. This volume correspond to the volume that would be occupied by the crystalline grains neglecting overlapping. The increase of actual and extended volumes during a time interval $dt$ is given by

$$\frac{dV/dt}{dV_{ext}/dt} = 1 - \frac{V(t)}{V_{ext}(t)}$$

Some attempts have been previously performed to describe the process of dynamic re-crystallization in the framework of the Avrami formulation. However, it is not directly suitable because the process of dynamic crystallization is in fact a "multiple-phase" process. At the beginning of the process a given texture exist, consisting in grains with a given grain size distribution; let's call this phase $f_0$. As strain is initiated, the density of dislocations in $f_0$-grains increases, and eventually new, free of dislocation grains, nucleate and start growing; let's call this phase $f_1$. $f_1$ grows and eventually occupy all the space previously covered by $f_0$. This process is well described by the Avrami formulation. However, as they grow, $f_1$-grains increase also their dislocation density; eventually, they reach the critical dislocation density and a new free of dislocation phase $f_2$ starts nucleating and growing; that is, in this second re-crystallization cycle $f_1$ and $f_2$ play the role of $f_0$ and $f_1$ in the first cycle. The Avrami formulation may then be applied to this process, provided $f_1$ had already replaced completely $f_0$. But this cannot be assured, as due to its exponential behavior some remaining $f_0$ space may still exist in the material.

To deal with this problem, we propose a "multiple-phase" approach. At every moment along the dynamic crystallization problem we will consider the following phases:

- $f_i$: phase recrystallized in the $i$-cycle, which is simultaneously growing following an

Avrami protocol related to its corresponding extended volume and vanishing due to a dislocation density above the critical value.

- $f_{i,c}$: phase recrystallized in the $i$-cycle who already overcome the critical dislocation density. This phase is in fact being replaced by the $f_{i+1}$. However, the growth of $f_{i+1}$ does not cover all the $f_{i,c}$ at the same speed that grains in $f_i$ reach the critical dislocation density. So $f_{i,c}$ takes into account the available space for growth of $f_i$, as well as it continues increasing its dislocation density due to the continuous strain.

The total transformed fraction (in the Avrami sense) is just the addition of all the $f_i$ phases:

$$x(t) = \frac{\sum_0^{i+1} V_i(t)}{\sum_0^{i+1} V_{i,ext}(t)}$$

From the above, it is seen that $f_{i,c}$ does not follow an Avrami protocol. In fact, the space occupied by $f_{i,c}$ is in fact free space for the growth of $f_{i+1}$. A simple volume conservation equation can be written for $f_{i,c}$:

$$V_{i,c}(t+dt) = (V_{i,c}(t) + dV_{i,d}) \cdot \frac{1 - x(t)}{\sum_0^i V_{i,c}(t)}$$

where $\frac{dV_{i,d}}{dt}$ accounts for the volume reduction of $f_i$ due to reaching the critical dislocation density. This equation ensures an homogeneous reduction of all the $V_{i,c}$ as well as makes the sum of them equal to the volume available for re-crystallization. And, additionally, it establishes that the phase re-crystallizing in the $i$-th cycle can crystallize in the space previously occupied by any previous recrystallized phase.

This approach allows one to apply successive Avrami protocols for each of the re-crystallization cycles. Given the succession of re-crystallization events, the Avrami equation becomes

$$\frac{\sum_0^{i+1} dV_i/dt}{\sum_0^{i+1} dV_{i,ext}/dt} = 1 - x(t)$$

## 2.1 From continuous to discrete description

While the above formalism can describe the amount of the different re-crystallization phases, it is unable to describe the grain size evolution. A discrete description, focusing in the grain populations of similar grain will thus be used [2].

The discrete description is inspired in the way that grain size distributions are obtained experimentally from optical or electron microscopy. Grains have an irregular shape, but in

order to classify them an effective radius is defined. Here we will assume that, regardless its shape, the effective radius of a grain is

$$r = \left(\frac{3V}{4\pi}\right)^{1/3}$$

The population of grains nucleated between $kt$ and $kt + 1$ has an initial grain size, and a counterpart extended population which initially has the same grain size. As experimentally, a discrete grain radius scale is adopted, based on a fixed grain size interval $dr$; then we define $kr_j = j \cdot dr$. Thus, the population of grains $P[kt][kr]$ accounts for the number of grains per unit volume nucleated at time $kt$ who have a radius $kr$ such as $kr_{j-1} < kr \leq kr_j$. The discrete description is completed by establishing a discrete timescale $t_i = i \cdot dt$, such as $G = \frac{dr}{dt}$, $G$ being the growth rate.

The new population for a given ratio is the frequency of birth times the fraction of free space. Then, we have to make some calculations to find the coefficient $\alpha$ that will split the population of each radius to complete this equation. We start with the expression of the volumes V at the time t and V' at t+1:

$$V = \sum_i P(t, r_i) * V(r_i)$$

$$V' = \sum_i [(1-\alpha)P(t_0, r_i) + \alpha P(t_0, r_{i-1})] * V(r_i)$$

The difference of the two gives us:

$$dV = V - V' = \alpha \sum_i [P(t_0, r_{i-1}) - P(t_0, r_i)]V(r_i)$$

We do the same with the extended population, which doesn't suffer from birth rate reduction:

$$V_{ext} = \sum_i P(t, r_{ext,i})V(r_{ext,i})$$

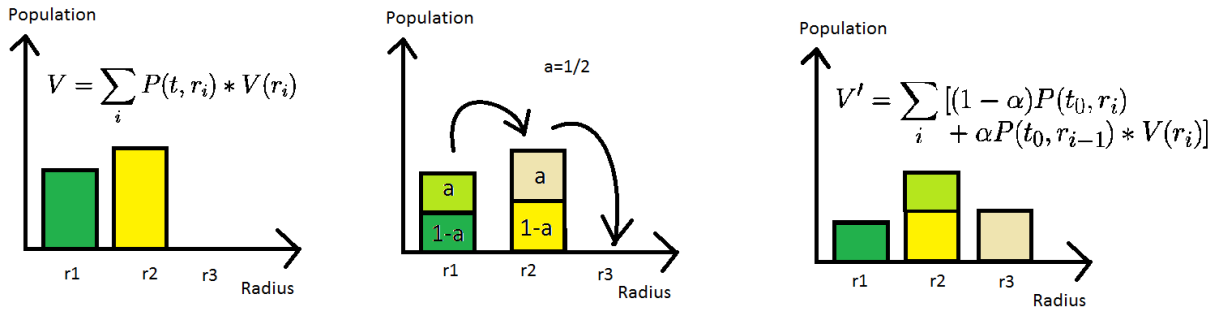
$$dV_{ext} = \sum_i P(t, r_{ext,i})[V(r_{ext,i} + \delta r) - V(r_{ext,i})]$$

With the Avrami equation, we have the following:

$$(1 - x) = \frac{dx}{dx_{ext}} = \frac{dV}{dV_{ext}}$$

That becomes, with the previous considerations on the volumes:

Figure 2.1: Example of real population growth with three radius $r_1 r_2 r_3$ and an $\alpha = \frac{1}{2}$




$$(1-x) = \frac{\alpha \sum_i [P(t_0, r_{i-1}) - P(t_0, r_i)] V(r_i)}{dV_{ext}}$$

That is equivalent to:

$$\alpha = (1-x) \frac{\sum_i P(t, r_{ext,i}) [(r_{ext,i} + \delta r)^3 - r_{ext,i}^3]}{\sum_i [P(t_0, r_{i-1}) - P(t_0, r_i)] r_i^3}$$

The $\alpha$ is what indicates us the fraction of growing cells, as drawn in the figure 2.1. We can see that for a $\alpha = \frac{1}{2}$ half of a population will grow to the next radius. In the end, we obtain each population:

$$P_{new}(t_0, r_i) = (1-\alpha) * P(t_0, r_i) + \alpha * P(t_0, r_{i-1})$$

## 2.2 Program Methodology

In this part I will explain the code variables ans algorithms. I will also describe the role of each function.

The program code was written in C++. The main variables used are:

- dr, dt: radius and time discrete intervals; default values 1 and 1.
- MAXR, MAXT: maximum discrete radius and time. Used to size the corresponding matrices; default values, 1000 and 3000.
- tMax, rMax: maximum time and radius to be simulated; must be tMax < dt·MAXT and rMax < dr·MAXR; default values 1 and 1.
- rate: deformation rate, in $[s^{-1}]$; default value $0.001s^{-1}$.
- p[MAXT][MAXR]: actual population. At time $t_k > t_i$, p[i][j] stores the amount of grains nucleated at $t_i$ which have a radius $r_j$ at time $t_k$.
- freqn[MAXR]: nucleation rate. Population density of grains which are born with radius r[j] at each time step.
- stopn: time at which nucleation should be turned off.
- K: it's the constant in the Monte Carlo formula $\sigma_i = K * \sqrt{\rho_i}$ where $K = M * \alpha * \mu * b$ [2]; default value 1.
- pobext[MAXT][MAXR]: extended population. At time $t_k > t_i$, pobext[i][j] stores the amount of grains nucleated at $t_i$ for every radius $r_j$, reduced as a consequence of the grains vanished when they reached the critical dislocation density.
- rext[MAXT][MAXR]: extended radius. At time $t_k > t_i$, rext[i][j] stores the extended radius of pobext[i][j] at time $t_k$
- zombi[MAXT][MAXR]: actual population who already reached the critical dislocation density. At time $t_k > t_i$, zombi[i][j] stores the amount of grains nucleated at $t_i$ which reached the critical dislocating density and have a radius $r_j$ at time $t_k$.

For the sake of flexibility two ways of reaching the critical dislocation density were considered. On the one side, it was considered that the critical dislocation density is related to the grain size. On the other side, it was considered that the critical dislocation density is related to the time since the grain nucleated. Accordingly, two vectors were defined, namely fMr[MAXR] and fMt[MAXT], which store the probability of reaching the critical dislocation density for a grain or radius r[j] and for a grain nucleated k time steps before, respectively.

Appropriate units were used. A volume of 1 $m^3$ is simulated, and that way the population densities are naturally in grains per $m^3$. Units for radius and time were chosen to be $\mu m$ and and s, respectively.

**Input**

Initial conditions are read from the file "condinit.txt". The file format is:

dt, tMax, dr, rMax, stopn, rate, K:
-1
j, freqn[j] (repeated as many times as needed)
-1
j, fMr[j] (repeated as many times as needed)
-1
j, fMt[j] (repeated as many times as needed)
-1
j, p[0][j] (repeated as many times as needed)
-1

The $\delta t$ and $\delta r$ are the step sizes of time and radius in second and micrometer. $t_{max}$ and $r_{max}$ are the maximum values they will reach. We can then calculate $G = \frac{\delta t}{\delta r}$ and the maximum indexes of the two dimensions of the matrix. The maximum indexes must not overcome the size of the matrix, if so, the program stops. We also read the variables stopn that is the time at which we want the births to stop and K of the Mont Carlo formula.

**Workflow**

After initialization we read the input file in the order:

- freqn[j] which is the frequency of birth of the $r_j$-sized crystals,
- fMr[j], and fMt[i] which are the fractions of extinction of crystals when they reach respectively the size $r_j$ or the age $t_i$, and pob[0][j] the initial population.

Second, we write

- the values of $r_j = (1 + j) * \delta r$, the initial extended population which is the real population.
- the $x_{ext} = \sum P_{ext}(r) * V(r)$ with $V(r) = 4/3 * \pi * r^3$
- initialization of zombi to zero.

The zombie matrix stock the population of dead cells, that will be useful to the calculus of total dislocation density.

We get the initial real fraction of recrystallized space with the Avrami formula. Then, we can launch the time loop that aims to calculate discretely the real and extended fraction of DRX and populations, and the corresponding stress. To calculate the fractions, we start with (1) the decrease of real and (2) extended population, only then we launch the increase of (3) real, then (4) extended population.

1. Real population matrix has two indexes: age index and radius index. To calculate the fraction of extinction, we first need to calculate the age, which is the present time, in the main loop, minus the date of birth kt+1. So, for the population P[kt][kr] we have the frequency of extinction: $fM = fMt[kt] + fMr[age]$. Consequently, the population becomes:

   $P_{new}[kt][kr] = (1 - fM) * P[kt][kr]$

   The zombie population will host the fraction $fM * P[kt][kr]$ that has died.

2. The decrease of extended population is similar, except the radius are not the same since they grow without affection of the volume occupied:

   $P_{ext,new}[kt][kr] = (1 - fM) * P_{ext}[kt][kr_{ext}]$

3. The growth of real population begins with the birth at the present time index, let's call it $iTime$. We call then the index of the matrix the *date of birth*. The new population for a given ratio is the frequency of birth times the fraction of free space: $Pnew[iTime][kr] = freqn[kr] * (1 - x)$

   In a second place, one must also take into account the evolution of the sizes of the older populations, using the $\alpha$ parameter we explained in the previous chapter:

   $Pnew[kt][kr] = P[kt][kr] * (1 - \alpha)$

   There, we measure the evolution of the global size in the volume.

4. There is no notion of $\alpha$ with he extended population. In other words, the extended population of a radius $kr$ at a time $kt$ will always grow to the radius $kr + 1$ at the following time $kt + 1$:

   $P_{ext,new}[kt + 1][kr + 1] = P_{ext}[kt][kr]$

**Output**

The values of $x, x_{Ext}, zvol, and zfactor$ are stored in *x.dat*, next to the corresponding time $kt$.

- $x$ is the fraction of real occupied space by the real population.
- $x_{ext}$ is the virtual occupied space by the extended population.

- $zvol$ is the total population of zombies from the first date to the present date.
- $zfactor$ is the ratio that gives the evolution of the zombie population.

In the file *dyncrist.dat* we stock the population $P[kt][kr]$ according to the radius $kr$ at each time $kt$.

In the file *stress.dat*, we store the values of $sigma[kt], epsilon[kt], rho, zrho, rho + zrho$

- $sigma[kt]$ is the $\sigma_t$, the stress
- $epsilon[kt]$ corresponds to $\varepsilon$, the strain
- $rho$ corresponds to $\rho_{rx}$ the dislocation density of the recrystallized population. As a consequence it doesn't take into account the dislocation of the rest of the volume.
- $zrho$ corresponds to $\rho_z$ the dislocation density of the zombie population, that takes into account the population left.
- $rho + zrho$ is the sum of the two previous values and is the $\rho$ we use to compute the pseudo-stress according to [2] $\sigma = K * \sqrt{\rho}$

# Chapter 3

# Results

## 3.1 Natural scales

The main governing parameters of the problem are the nucleation and growth rates. Appropriate time and length scales can be built from them by using dimensional analysis. These scales give us the order of magnitude the time needed to reach the steady state and the radius of the crystallites at the the steady state.

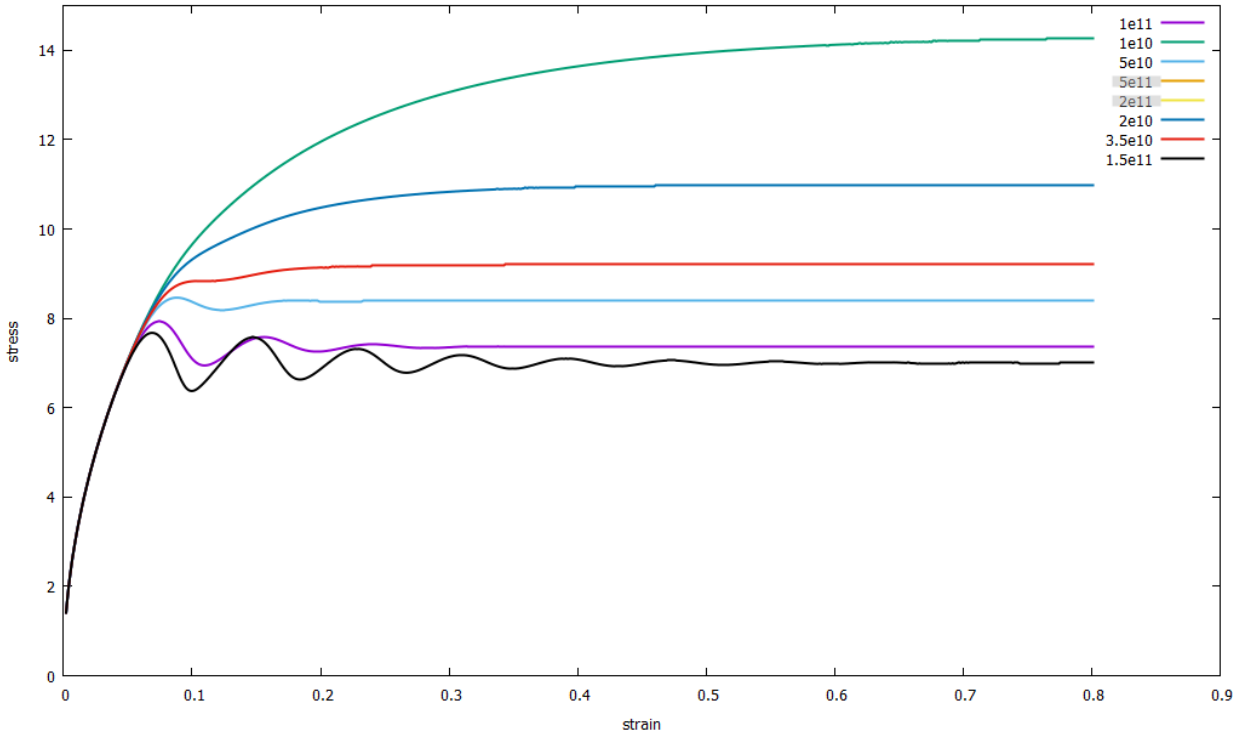
- time: $\tau = G^{-3/4} * I^{-1/4}$
- length: $\lambda = G^{1/4} * I^{-1/4}$

The main utility of these scales is to help us to select the proper sizes of the dimensions of the matrices used in our code. Table 3.1 gives their values for an example case.

Table 3.1: Scales of time and length.

| Name | definition | value | dimension | conversion |
|---|---|---|---|---|
| $\delta r$ | . | $1.10^{-6}$ | m | $1\mu m$ |
| $\delta t$ | | 1 | s | . |
| $G$ | $\delta r/\delta t$ | $1.10^{-6}$ | $m/s$ | $1\mu m/s$ |
| $I$ | $\sum f_n(r)/(V_{tot} * \delta t)$ | $10^9$ | $at/m^3/s$ | . |
| $\tau$ | $G^{-3/4} \cdot I^{-1/4}$ | 177 | s | . |
| $\lambda$ | $G^{1/4} \cdot I^{-1/4}$ | $1.77{\cdot}10^{-6}$ | m | $177\mu m$ |
| $\delta r/\lambda$ | | 177 | | |
| $\delta t/\tau$ | | 177 | | |

Figure 3.1: comparison of stress curves for different rates of birth




## 3.2 Model validation

The code was ran with different parameters to check its stability and the effect of the different parameters, as well as to be compared to experimental data.

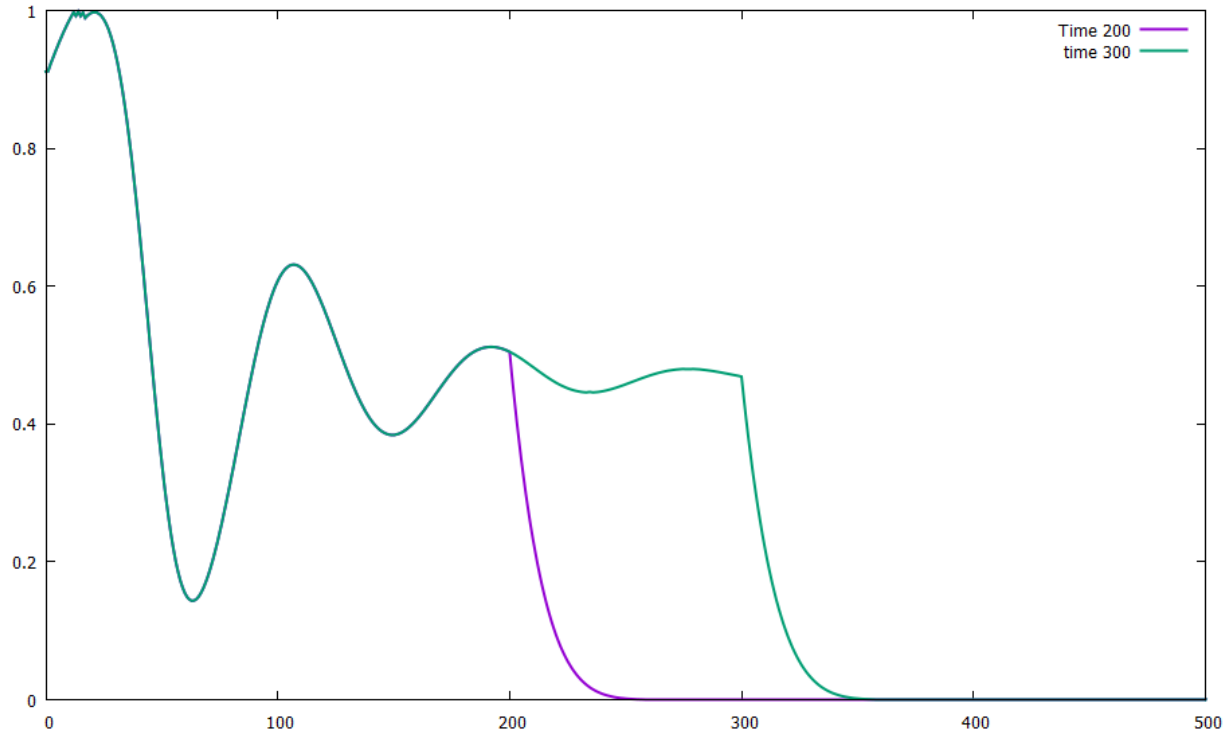
### 3.2.1 Birth rate

The rate of birth is measured by $I$. Its change will have a great impact on the oscillation of the stress curve. Indeed, if the birth rate is slow enough, the flow of living population inside the volume will stabilize before that any oscillation occurs. We can see it on the figure 3.1. For a birth rate of 1.5e11 crystals per second and cubic meter, we obtain the black curve. The oscillations are clear, and the final stress is low. In purple, there is the curve that has a rate of 1e11 birth per second. It still is a multiple peak-type DRX.

### 3.2.2 Birth stop

The birth can be stopped to observe the evolution of population only lead by extinction. Here we compare the fraction $x$ when birth stop is done at 200 and at 300 seconds. We then

Figure 3.2: effect of stopping birth



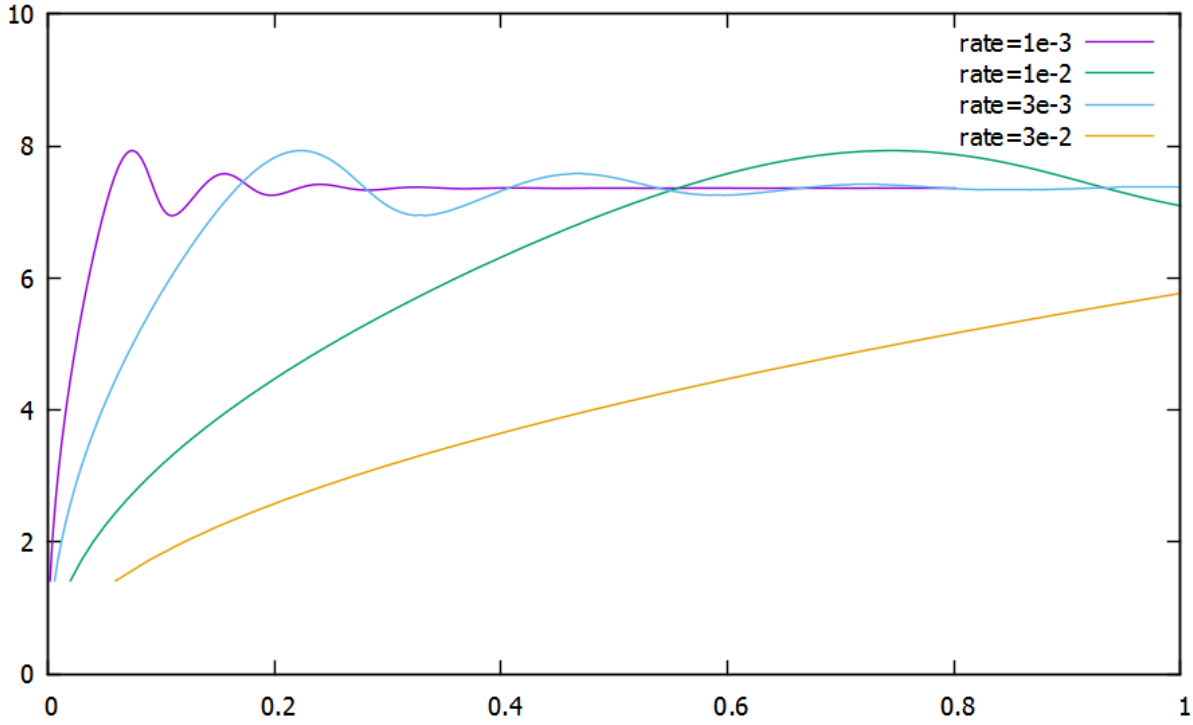

observe that the fraction of re-crystallized volume decrease as soon as the time is reached. We can see the effect of stopping birth on the figure 3.2.2. There are compared the effect of stopping birth after 200 seconds and after 300 seconds.

### 3.2.3 Deformation rate

The deformation rate $\dot{\varepsilon}$ appears in the Zener-Hollomon parameter. It plays a key role in the DRX as we saw, because if it's done too quick, no DRX will occur. However, it doesn't appear directly on our model. It only serves to get the scale of strain in the epsilon matrix.

The result we obtain is in the figure 3.3. Unfortunately, the curves obtained don't fit with the curves obtained with the experiment. It could mean that there is an incoherence in the code shown. Indeed, in the documents there are no clear link between the deformation rate and the stress curve. All we have is the scheme of the chapter 1. The scheme gives us a relation between the initial grain size and the Zener Hollomon parameter, it tells that for the same Z we get the same $D_{rex}$. At this point in the developing of the program, we couldn't solve this issue.

Figure 3.3: stress-strain curves comparing deformation rate




## 3.3 Initial population

Here we are looking at the different results according to the distribution of the initial population. We studied different initial distributions, namely a delta distribution, a plateau, a delta-binomial, and a double binomial distributions.

### Delta distribution

Here we consider that the original population has a unique radius. The figure 3.4 shows the distribution of the population at the initial time. The values of the population were chosen to cover the whole volume, that is x(0)=1. This is the easiest way to write the initial conditions.

### Plateau

Here is another form of initial population, that takes a different shape. Here, we consider that the population is constant through a certain window of radius. We see the distribution at the figure 3.3. This distribution also filled the whole volume, that is x(0)=1.

Figure 3.4: population distribution of the peak type

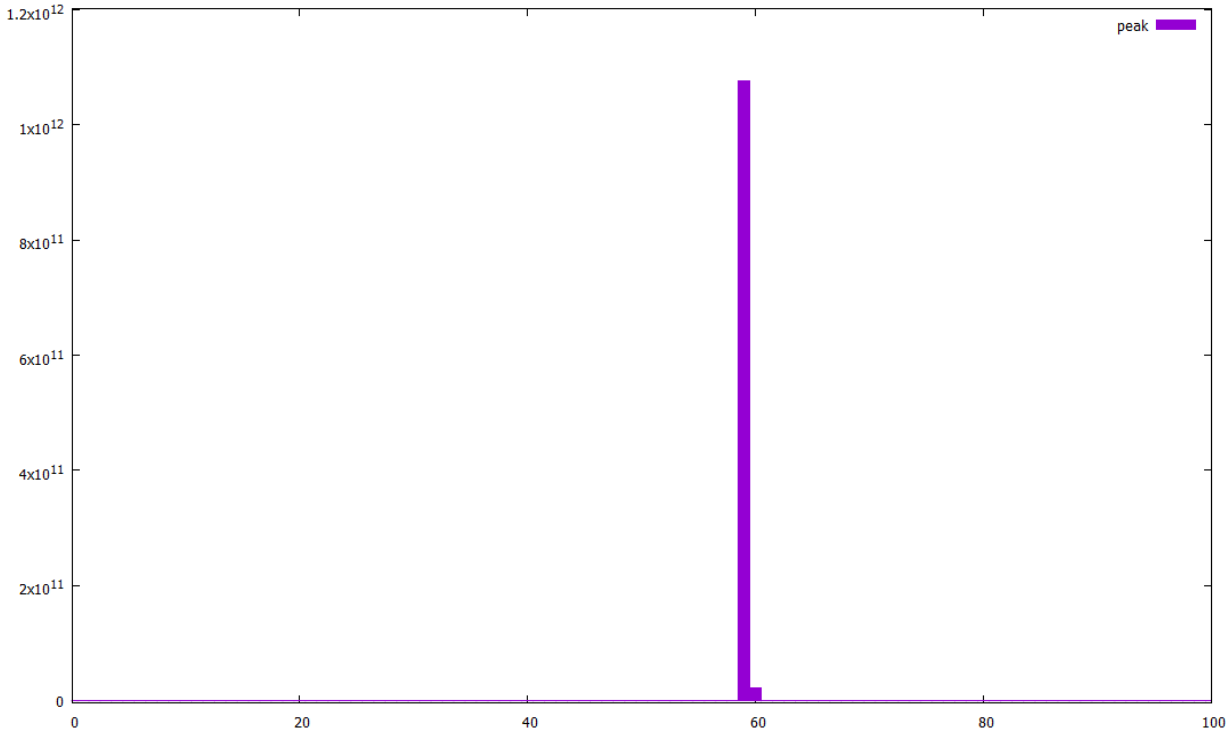



Figure 3.5: population distribution of the plateau type

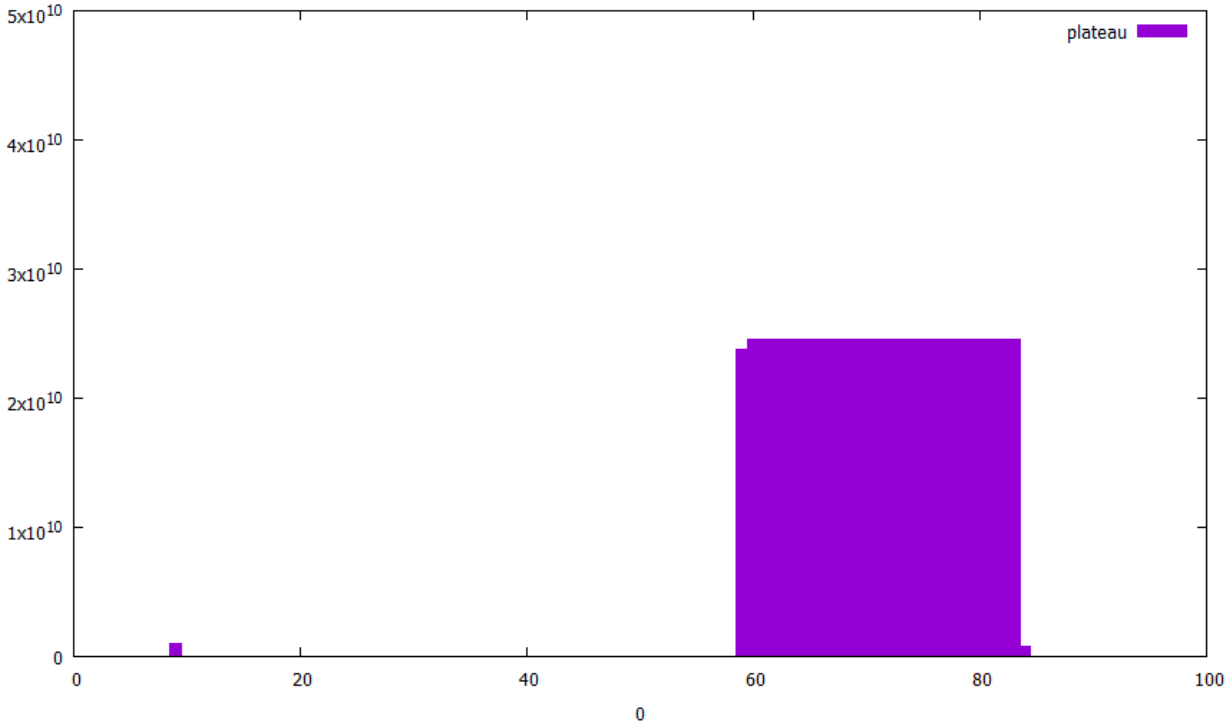
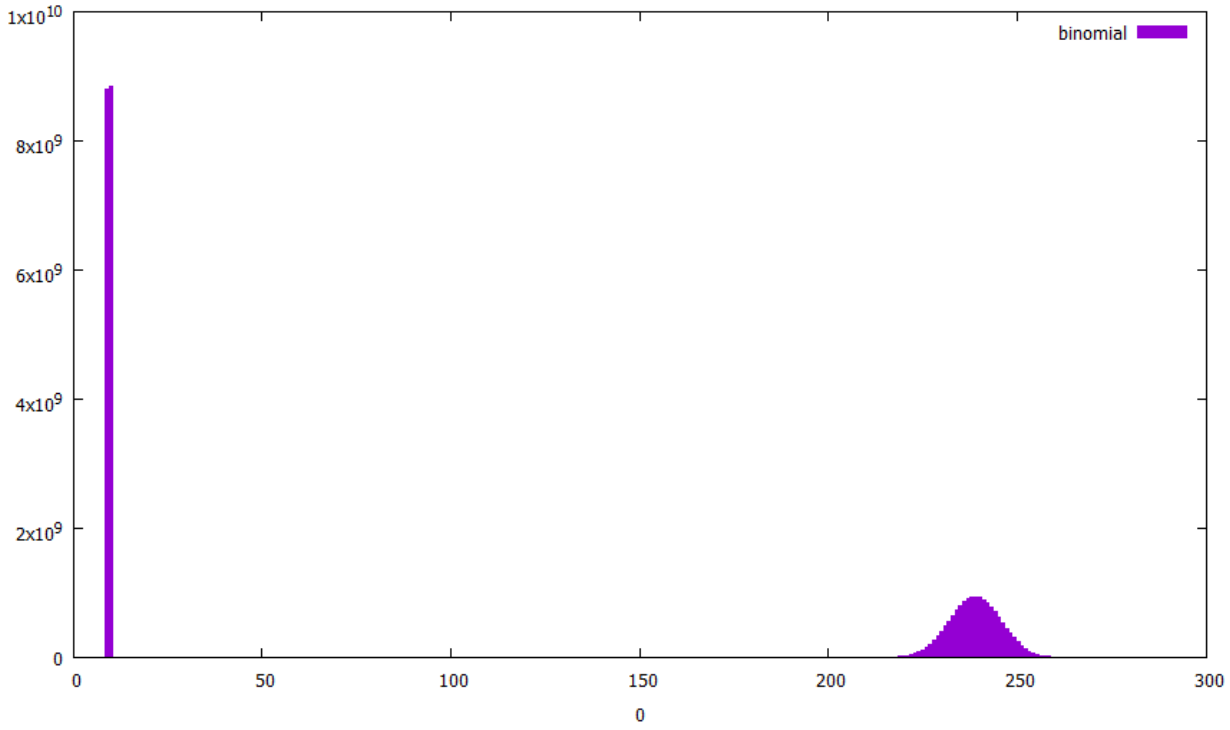


Figure 3.6: binomial population distribution




### Delta-binomial

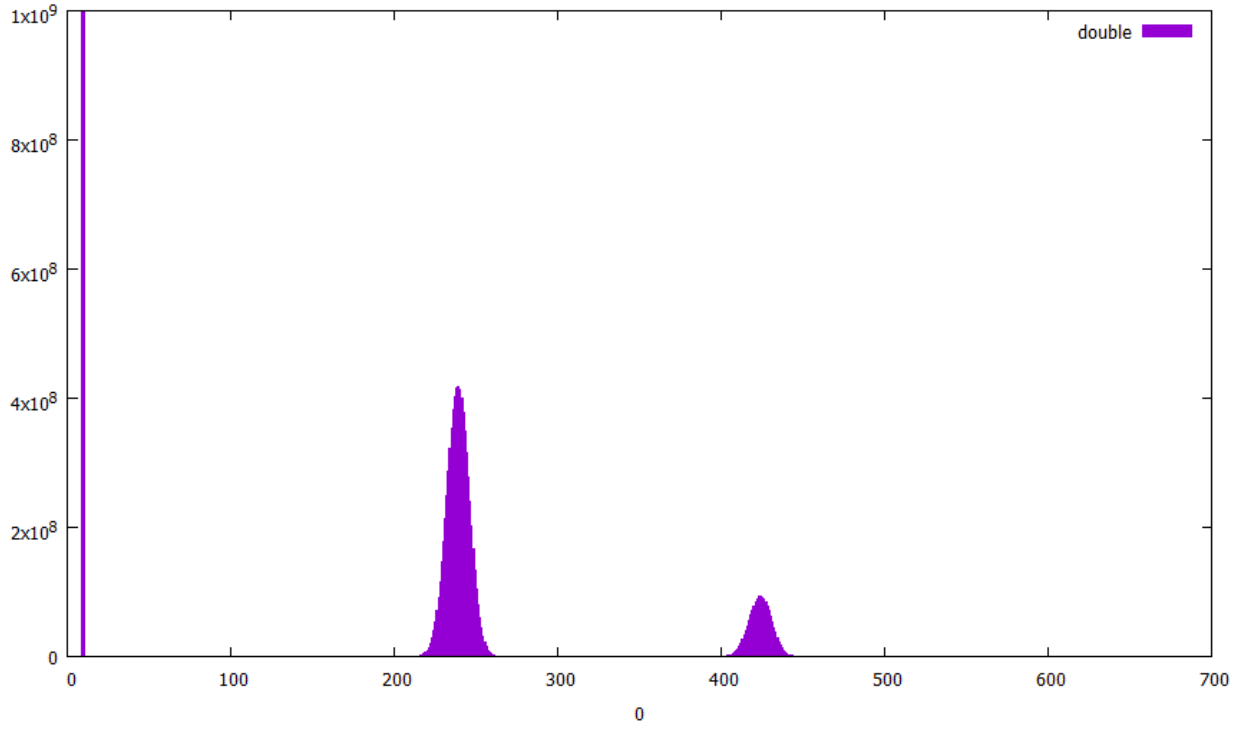
The delta-binomial distribution is interesting to test, because it is toward that distribution that the DRX aims. Indeed, at every step, when we take a fraction of population of radius at every step of time, as shown in the chapter 2.1, we finally get to a binomial distribution. In this case, the initial volume filled by the populations was 90%, that is x(0)=0.9. We can see on the figure 3.6 the initial distribution of this population.

### Double binomial

On this case, we made it double to compare with the rest, so we added a binomial peak to the previous one, as we see on the figure 3.3. As in the previous case, the initial volume filled by the populations was 90%, that is x(0)=0.9.

Figure 3.3 shows the strain-stress curves of each initial distribution. They do not show a strong effect in the strain-stress response. Additional work is needed to understand the influence of initial distribution, but we can expect that the binomial shaped distribution will involve softer variations in the stress.

Figure 3.7: double-binomial population distribution




## 3.4 Extinction protocols

Figures changing the intensity of the extinction and/or the time where the extinction starts/ends. The equation that modulates most adequately the extinction of the population is a sigmoïd of the type:

$fMt[x] = \frac{C}{1+\exp(\frac{A-x}{B})}$ It means that it takes values near 0 for low $x$, then there is a transition zone around $A$ and it reaches $C$ right after. $C = 20\%$ and $A = 50$ are our default values. We can see it represented on the figure 3.9 with all of the other values we tested.

First, we tried different values for $C$, and the results of the modelings are gathered in the figure 3.10. Here we can see that the lower the extinction rate is, the higher the stress will be. Moreover, we can say that for extinction amplitudes lower than ten percent, we don't observe the desired DRX phenomenon.

Then, we compared different radius of death, as we can see on the figure 3.11. We can see that from a radius of extinction of 100, the curve takes a strange shape. It must be due at the fact that the space is filled too quickly compared to extinction. Anyway, for an early extinction, we observe, as the previous test, what could be a dynamic restoration. This means that at no moment, the stress decreases. It is associated with a high Zener-Hollomon parameter. We can imagine that the extinction is though has parameters $A, B, andC$ proportionals to

Figure 3.8: comparison of the stress-strain curves obtained for different initial distribution

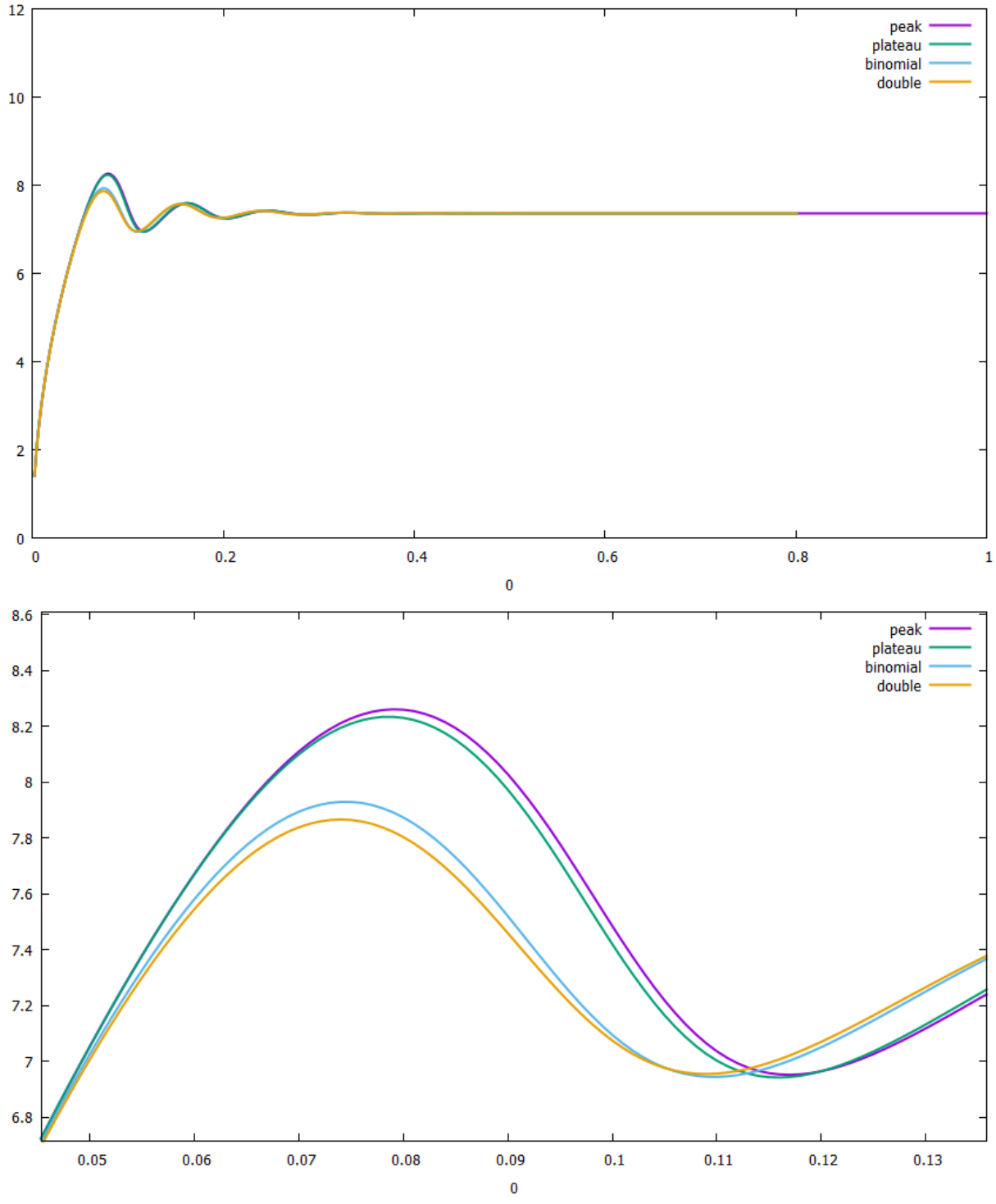
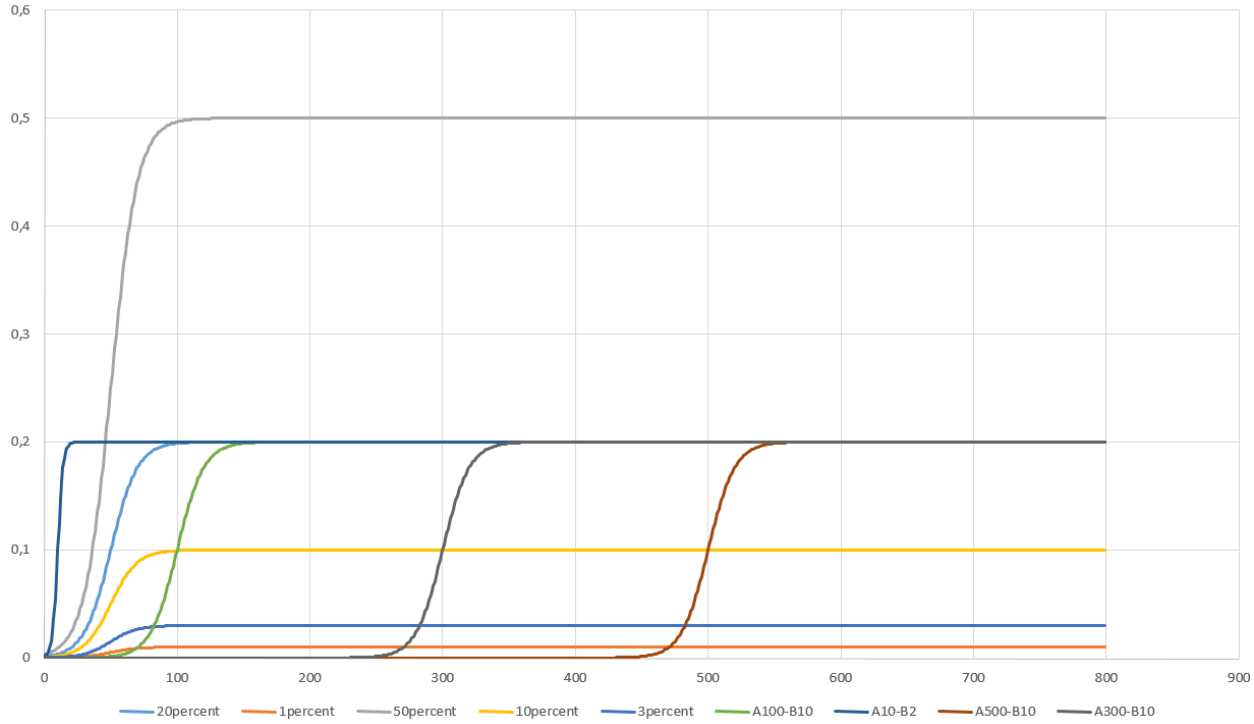


Figure 3.9: different types of extinction




the deformation rate.

## 3.5 Discussion

We have seen all along the third chapter all of the changes that enhanced each parameter. On the figure 3.12 we have a modeled curve compared to experimental data. We can see that they have similar undulations, nevertheless, they don't fit to each other since they don't reach the same mean final value.

We have also seen with the different tests, that there were parameters depending from each other. In other words, one should really set precise rules to define the initial conditions. Always keeping in mind that the final objective is to fit the most accurately the experimental curve.

Figure 3.10: effect of different amplitudes of extinction

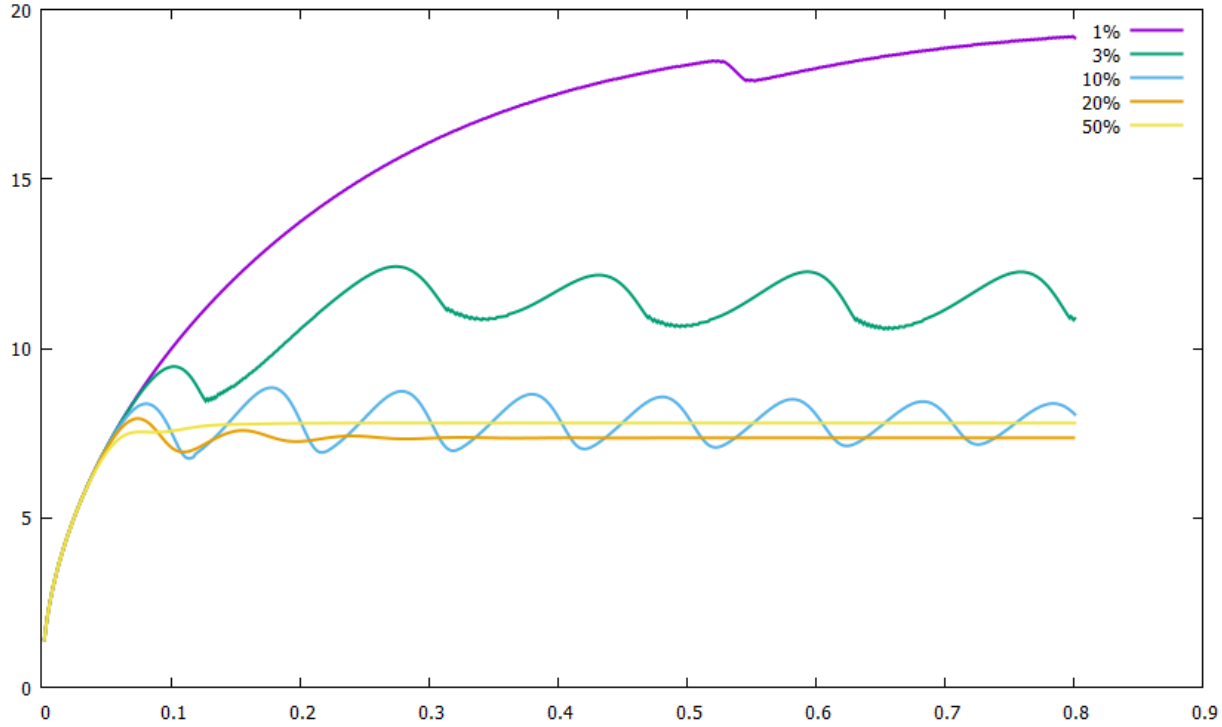



Figure 3.11: effect of different radius until extinction

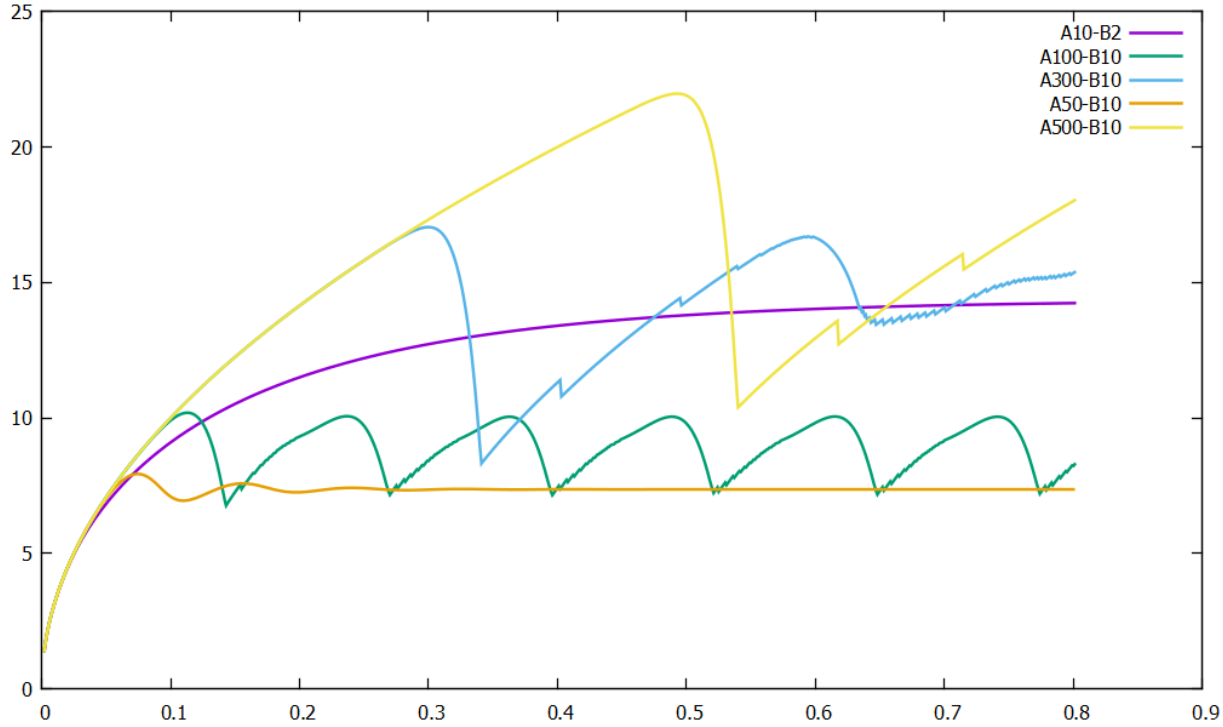
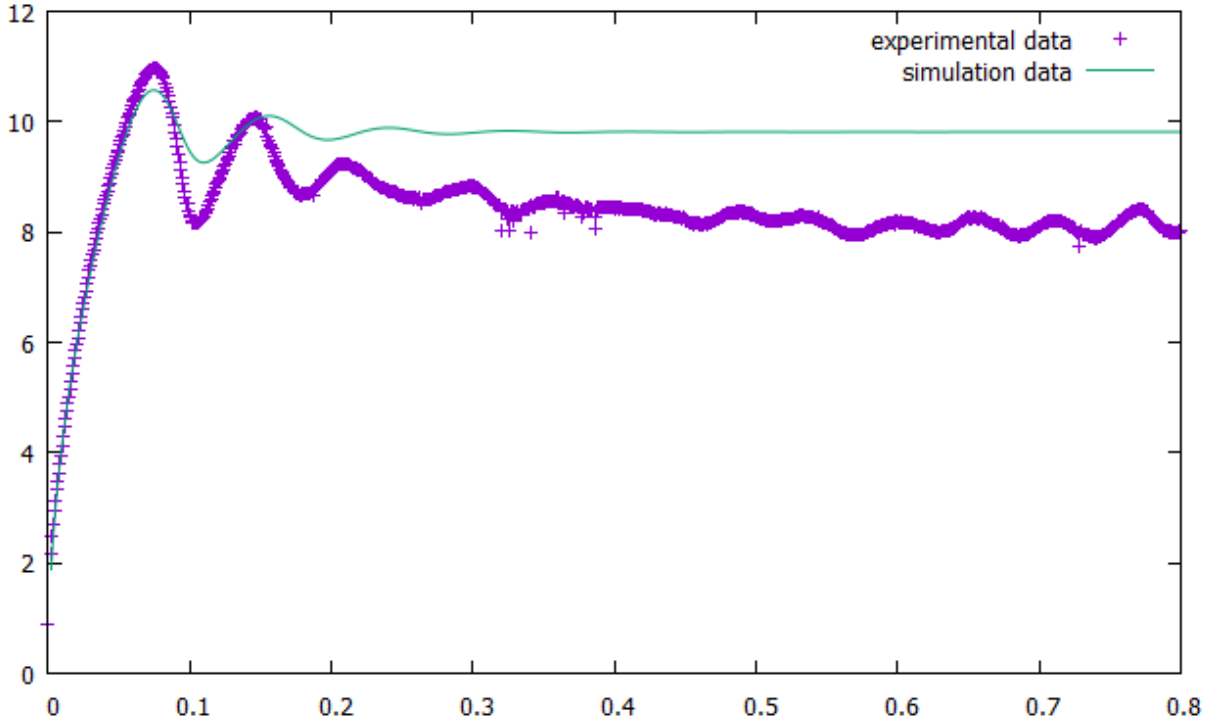


Figure 3.12: comparison of experimental data with simulated curve




## 3.6 Conclusion

The model developed captured the essential features of dynamic re-crystallization processes. The developed code allows to model the process in different materials by properly tuning the initial conditions. A wide survey on the parameters was out of the scope of this project, due to its limited duration, but it is expected that future work may allow to simulate the re-crystallization on processes on different metals. This would allow to have a unified picture of the re-crystallization process, allowing to relate the different responses to the specific parameters of the simulation.

# Bibliography


1. constitutive relations to model the hot flow of commercial purity copper, Victor Gerardo Garcia Lopez, Barcelona 2004, Chapter 6

2. Microstructural simulation of dynamic recrystallization, A.D.Rollett, M.J.Luton, D.J.Srolovitz, Los Alamos National laboratory, 1991

# Appendix A

# Program Source

```
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define PI 3.1415923565
#define DIMT 3000
#define DIMR 1000
#define VOL_TOT 1 //unidad m^3
using namespace std;

#define MIN(a,b) a < b ? a : b
#define MAX(a,b) a > b ? a : b
#define NDXR(a) (int) a/deltaR - 1

static double xPob[DIMT];
float deltaR=0.0;
FILE *pfr, *pfx;

float volumen(float r)
{
        return(4.0/3.0*PI*pow(r,3.0)*1e-18);
}

double sumavol(int maxR, int iTime, float p[DIMT][DIMR], float r[DIMR], int tMin)
{
    double xnew;
    int kr, kt;
```

```
    for(kr=0,xnew=0.0;kr<maxR;kr++)
        for (kt=tMin;kt<=iTime;kt++) xnew+=p[kt][kr]*volumen(r[kr]);
//    printf("%f",xnew);
//    xnew=MIN(xnew,1.0);
    return xnew;
}

double decrExt(int maxR,int iTime,float p[DIMT][DIMR],
               float rext[DIMT][DIMR],float fMr[DIMR],
               float fMt[DIMT], int tMin)
// En esta función se calcula solo la disminución
//de poblaciones extendidas por muerte
{
        double xExt, fM;
        int kr,kt,krAct,age,kPropio;

    for(kt=tMin,xExt=0.0; kt<iTime; kt++)
    // Radio en el momento del nacimiento
    {
            for(kr=0, kPropio=0; kr<maxR; kr++)
            {
                if (p[kt][kr]>0)
                {
                    // kPropio                  = kr;
                    // krAct         = MIN( kr+(iTime-kt), maxR-1 );
                    krAct         = MIN( NDXR( rext[kt][kr] ), maxR-1 );
                    age           = 1+(iTime-kt);
                    fM            = MIN( fMr[krAct] + fMt[age], 1.0 );
                    p[kt][kr]     = p[kt][kr]*(1.0-fM);
//                    xExtPropio          = p[kt][kr]*volumen(rext[kt][kr]);
//                    xExt         += xExtPropio;
                      }
        }
        xExt+=sumavol(maxR, kt, p, rext[kt], kt);
        if (kPropio>0 && kt==2+tMin)
        fprintf(pfr,"%g %g\n", p[kt][kPropio], rext[kt][kPropio]);
    }

    return xExt;
}

double incrExt(int maxR,int iTime,float G,
```

```
                float p[DIMT][DIMR],float r[DIMR],
                float rext[DIMT][DIMR],float freqn[DIMR],
                float deltaT,float rMax, int tMin)
// en esta función se calcula solo el nacimiento
//y crecimiento de granos extendidos
{
    double xExt;
    int kr,kt;

//    for( kt=tMin, xExt=0.0; kt<iTime; kt++ )
    for( kt=1, xExt=0.0; kt<iTime; kt++ )
            // Radio en el momento del nacimiento
    {
        for( kr=0; kr<maxR; kr++ )
        {
            rext[kt][kr] = (p[kt][kr]>0) ? MIN( rext[kt][kr] + G*deltaT,  rMax ) : 0.0;
        }
        xExt+=sumavol(maxR, kt, p, rext[kt], kt);
    }

    for( kr=0; kr<maxR; kr++ )
    {
        p[iTime][kr]   = freqn[kr];
        rext[iTime][kr] = (p[iTime][kr]>0) ? r[kr] : 0.0;
    }
    xExt+=sumavol(maxR, iTime, p, rext[iTime], iTime);
    return xExt;
}

double decrReal(int maxR, int iTime, float r[DIMR],
                float p[DIMT][DIMR],float fMr[DIMR],
                float fMt[DIMT], int tMin, float zombi[DIMT][DIMR])
// en esta función se calcula únicamente el decrecimiento
//de poblaciones reales por muerte
{
        int kr,kt,age;
        float fM;
        double xNew=0.0;

        for(kr=0;kr<maxR;kr++)
        {
            for (kt=tMin;kt<=iTime;kt++)
```

```
            {
                age         = 1+(iTime-kt);
                fM          = MIN((fMr[kr]+fMt[age]),1.0);
                zombi[kt][kr]   = zombi[kt][kr] + fM*p[kt][kr];
                // x = p[kt][kr];
                p[kt][kr]   = (1.0-fM)*p[kt][kr];
                // desaparición de poblaciones
                // xNew       += p[kt][kr]*volumen(r[kr]);
            }
        }
        xNew=sumavol(maxR, iTime, p, r, tMin);
    //printf("decr %f\n",xNew);
        return xNew;
}

double incReal(int maxR, int iTime, float p[DIMT][DIMR],
             float pnew[DIMT][DIMR],float freqn[DIMR],
             float x, float r[DIMR], float rext[DIMT][DIMR],
             int tMin, float zombi[DIMT][DIMR], float pobExt[DIMT][DIMR], double G, flo
{
        int kr,kt;
        double Gdt, pr3, prExt2, alpha, xNew, xExt;

           for (kt=tMin;kt<iTime;kt++) for(kr=0; kr<maxR; kr++) pnew[kt][kr] = p[kt][kr]

        for(kr=0; kr<maxR; kr++)
        {
        pnew[iTime][kr]=freqn[kr]*(1.0-x);        // nacimiento
        zombi[iTime][kr]=0.0;
        }

        xNew=sumavol(maxR, iTime, pnew, r, tMin);

    Gdt = G*dt;

    for( kt=tMin, xExt=0.0; kt<=iTime; kt++ )
        xExt+=sumavol(maxR, kt, pobExt, rext[kt], kt);

    for (kt=tMin;kt<iTime;kt++) // tiempo de nacimiento
        {
            pr3    = ( - p[kt][0] )* pow(r[0],3.0);
                prExt2  = pobExt[kt][0] * (pow(rext[kt][0]+Gdt,3.0) - pow(rext[kt][0],3.
```

```
                for(kr=1; kr<maxR; kr++)
            {
                pr3    += ( p[kt][kr-1] - p[kt][kr] )* pow(r[kr],3.0);
                       prExt2  += pobExt[kt][kr] * (pow(rext[kt][kr]+Gdt,3.0) - pow(rext
        }
        }
        if(pr3==0.0) pr3=1E-15;
        alpha   = prExt2 / pr3 * (1.0-xNew);
        fprintf(pfx,"%d %g\n", kt, alpha);
    alpha   = MIN( alpha, 1.0 );
    for (kt=tMin;kt<iTime;kt++) // tiempo de nacimiento
                 for(kr=0, xPob[kt]=0.0; kr<maxR; kr++)
            {
                    pnew[kt][kr] = (1.0-alpha) * p[kt][kr];                    //no ha c
            if (kr>0) pnew[kt][kr] += alpha * p[kt][kr-1];  //ha crecido
            xPob[kt]  += pnew[kt][kr] * volumen( r[kr] );
            }

        xNew=sumavol(maxR, iTime, pnew, r, tMin);
        if (xNew>=1.0)
        {
            alpha=0.0;
               for (kt=tMin;kt<iTime;kt++) // tiempo de nacimiento
            for(kr=0; kr<maxR; kr++)
                pnew[kt][kr] = p[kt][kr];                    //no ha crecido
        }
        xNew=sumavol(maxR, iTime, pnew, r, tMin);
        return xNew;
}

int main()
{
    int i=0,j=0,kt=0,k=1,maxT=0,maxR=0,stopn, tMin=0;
    float deltaT=0.0,rMax=0.0,tMax=0.0;
    float rate,K=1.0, zvol, zfactor;
    double G=0.0,I=0.0;
    static float pob[DIMT][DIMR],r[DIMR];
    static float pobExt[DIMT][DIMR],rext[DIMT][DIMR];
    static float freqn[DIMR], fMr[DIMR],fMt[DIMT],pobTemp[DIMT][DIMR];
    static float zombi[DIMT][DIMR];
    static double escalaT=0.0,escalaR=0.0, xext0, corrxext;
    //velocidad de deformacion=d epsilon/d t
```

```
//y constante de formula de monte carlo
static double sigma[DIMT],epsilon[DIMT], rho, zrho;
double xdeseado=0.0, xExt=0, xnew,sumapob=0.0, age, x=0.0, xx;

FILE * cond_init,
     * dyncrist,
     * ftrans,
     * stress;

    pfr=fopen("pfr.dat","w");
    pfx=fopen("pfx.dat","w");

/* lectura de los condiciones iniciales */
cond_init=fopen("cond_init.txt","r+");
if (cond_init==NULL){
    printf("error abriendo cond_init.txt");
    exit(-7);
}
fscanf(cond_init,"%f",&deltaT);
fscanf(cond_init,"%f",&tMax);
fscanf(cond_init,"%f",&deltaR);
fscanf(cond_init,"%f",&rMax);
fscanf(cond_init,"%d",&stopn);
fscanf(cond_init,"%f",&rate);
fscanf(cond_init,"%f",&K);
fscanf(cond_init,"%d",&k);
if(k>0){
    printf("error de lectura, k = %d > 0\n",k);
    exit(-1);
}

/* Calculo de condiciones iniciales */
G=deltaR/deltaT;
if ((rMax/deltaR<DIMR)&&(tMax/deltaT<DIMT)){
    maxT=(int) tMax/deltaT;
    maxR=(int) rMax/deltaR;
    }
else{
    printf("error con los valores de delta t, max t, delta r, max r\n");
    exit(-2);
}
```

```
/*seguida de la lectura*/
fscanf(cond_init,"%d",&j);
for(k=0;(j>=0)&&(k<maxR);k++){
    fscanf(cond_init,"%f",&freqn[j]);
    fscanf(cond_init,"%d",&j);
}
if(k>maxR){
    printf("error: demasiado valores para freqn \n");
    exit(-3);
}
fscanf(cond_init,"%d",&j);
for(k=0;(j>=0)&&(k<maxR);k++){
    fscanf(cond_init,"%f",&fMr[j]);
    fscanf(cond_init,"%d",&j);
}
if(k>maxR){
    printf("error: demasiado valores para fMr \n");
    exit(-4);
}
fscanf(cond_init,"%d",&i);
for(k=0;(i>=0)&&(k<maxT);k++){
    fscanf(cond_init,"%f",&fMt[i]);
    fscanf(cond_init,"%d",&i);
}
fscanf(cond_init,"%d",&j);
for(k=0;(j>=0)&&(k<maxR);k++){
    fscanf(cond_init,"%f",&pob[0][j]);
    fscanf(cond_init,"%d",&j);
}
if(k>maxR){
    printf("error: demasiado valores para pob[t=0][r] \n");
    exit(-5);
}
fclose(cond_init);

/*calculos iniciales*/
for (j=0;j<maxR;j++) r[j]=deltaR*(1.0+j);
x=sumavol(maxR, 0, pob, r, 0);
corrxext=pow( -log(1.0-x)/x, 1.0/3.0 );
for (j=0;j<maxR;j++)
{
    pobExt[0][j]=pob[0][j];
```

```
        rext[0][j] = pobExt[0][j]>0 ? corrxext*r[j] : 0.0;
        zombi[0][j]=0.0;
    }

    xext0=sumavol(maxR, 0, pobExt, rext[0], 0);
    xx=1.0-exp(-xext0);
    for(k=0;k<maxT;k++)
    {
        epsilon[k]=rate*(1.0+k*deltaT);
    }
    for(j=0,I=0.0;j<maxR;j++)
    {
//          I+=freqn[j]/(deltaT*volumen(r[j]));//unidad 1/m3*t
        I+=freqn[j]/(deltaT);//unidad 1/m3*t
    }

    /*calculos de escala*/
    //I=I*pow(10.0,-18.0);
    escalaR=pow(G*1e-6,0.25)*pow(I,-0.25);
    escalaT=pow(G*1e-6,-0.75)*pow(I,-0.25);
    printf("I=%g [m^-3*s^-1], G=%g [m*s^-1]\n"
           ,I,G*1e-6);
    printf("escala t = %g [s] , escala r = %g [m]\n"
         ,escalaT,escalaR);
    printf("delta t/escala t = %g , delta r/escala r = %g \n"
           ,deltaT/escalaT,(deltaR*pow(10,-6))/escalaR);

    /* ficheros donde se escriben los resultados*/
    dyncrist=fopen("dyncrist.dat","w+");
    ftrans=fopen("x.dat","w+");
    stress=fopen("stress.dat","w");
    fprintf(stress,"#stress / strain / dislocation density\n\n ");
    fprintf(ftrans,"%d %g %g\n",kt,x,xExt);

    /*inicio de la modelisacion*/
    for(kt=1;kt<maxT;kt++)
    {
        if(stopn==kt)
        {
            for(j=0;j<maxR;j++) freqn[j]=0.0;
        }
         // Primero reducimos las poblaciones real y extendida por muerte
```

```
            xnew=decrReal(maxR, kt, r, pob, fMr, fMt, tMin,zombi);
        xExt=decrExt(maxR, kt,pobExt, rext, fMr, fMt, tMin);
                xnew=incReal(maxR, kt, pob, pobTemp, freqn, x, r, rext, tMin, zombi, po
                xExt=incrExt(maxR,kt,G,pobExt,r,rext,freqn,deltaT,rMax, tMin);

        /*calculos de la fraccion real*/
        xdeseado=1-exp(-xExt);
        xnew=sumavol(maxR, kt, pobTemp, r, tMin);
        //suma del volumen de pobTemp por cada radio, desde k=0 hasta k=kt-1
        x=xnew;
        zvol=sumavol(maxR, kt, zombi, r, tMin);
        zfactor=1.0;
        if (zvol != 0.0) zfactor=(1.0-xnew)/zvol;
//        printf("%4d %6.4lg %6.4lg %6.4lg %6.4lg\n",
//               kt, x, xdeseado, xnew, xExt);
        for(j=0;j<maxR;j++){        // r index
            for (k=0;(k<=kt);k++) pob[k][j]=pobTemp[k][j];
        }
        xext0=xExt;
        for(j=0,rho=0.0,zrho=0.0;j<maxR;j++)
        {
            for(k=0;k<=kt;k++)
            {
                zombi[k][j]=zombi[k][j]*zfactor;
                age=1.0+kt-k;
                rho+=pob[k][j]*volumen(r[j])*age;
                zrho+=zombi[k][j]*volumen(r[j])*age;
            }
        }
        sigma[kt]=K*sqrt(rho+zrho);

    /* escritura de los resultados*/
//        printf("time = %d, rho = %f \n",kt,rho);
         if(kt%10==0)
            printf("|");
        fprintf(dyncrist,"#time %d \n",kt);
        for(j=0;j<maxR;j++) // r index
        {
            for(k=0, sumapob=0.0;k<=kt;k++)
                sumapob+=pob[k][j];
            fprintf(dyncrist,"%d %f \n",j,sumapob);
//            fprintf(pfx,"%d %g\n", j, xPob[j]);
```

```
        }
        fprintf(ftrans,"%d %g %g %g %g\n",kt,x,xExt,zvol,zfactor);
        fprintf(dyncrist,"\n\n");
        fprintf(pfx,"\n\n");
        fprintf(stress,"%f %f %f %f %f\n",sigma[kt],epsilon[kt],rho,zrho,rho+zrho);
    }
    fclose(ftrans);
    fclose(dyncrist);
    fclose(stress);

    return 0;
}
```