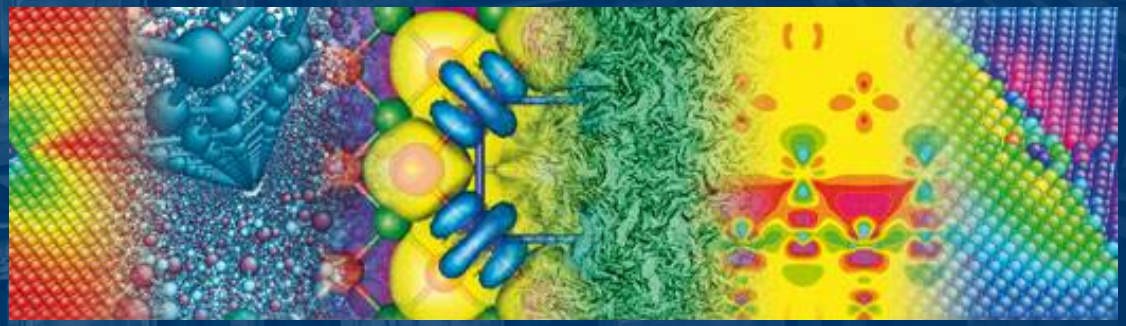
Multiscale Simulation Methods I (MuSim I)

Amorphous plasticity



David Fernandez Castellanos

Department für Werkstoffwissenschaften

Lehrstuhl für Werkstoffsimulation (WW8)

Dr.-Mack-Str. 77, 90762 Fürth

David.fernandez.castellanos@fau.de

www.matsim.techfak.uni-erlangen.de

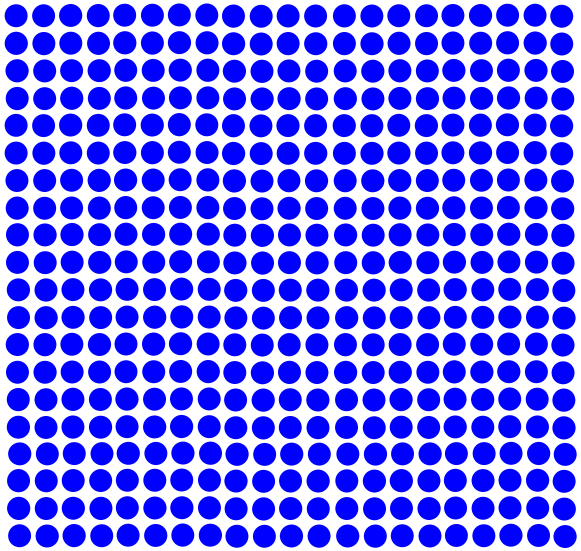
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

1. Bulk Metallic Glasses

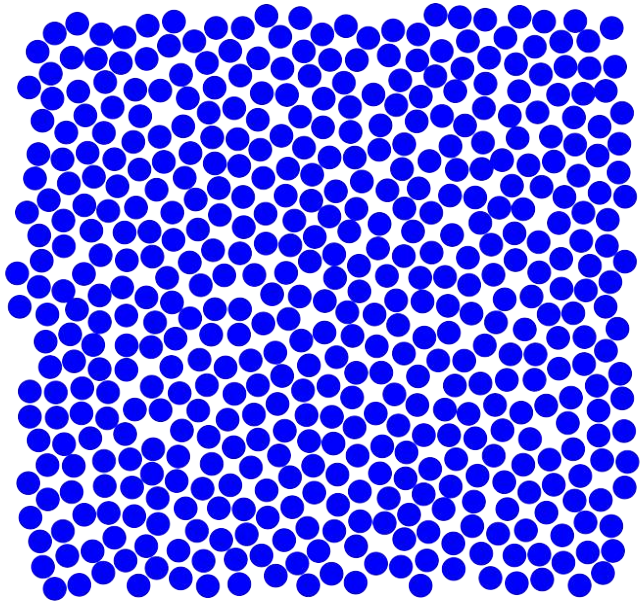
1. Bulk Metallic Glasses

Crystalline structure



- Periodic and ordered
- Helpful to make crystal plasticity theory
- Well-defined dislocations

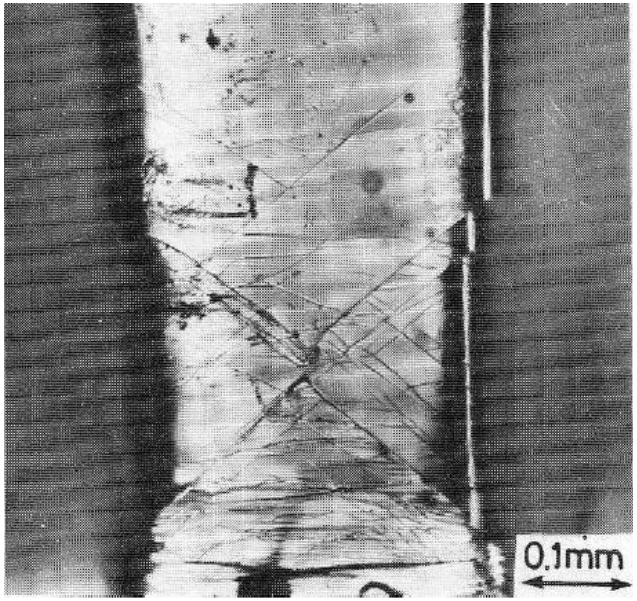
Amorphous structure

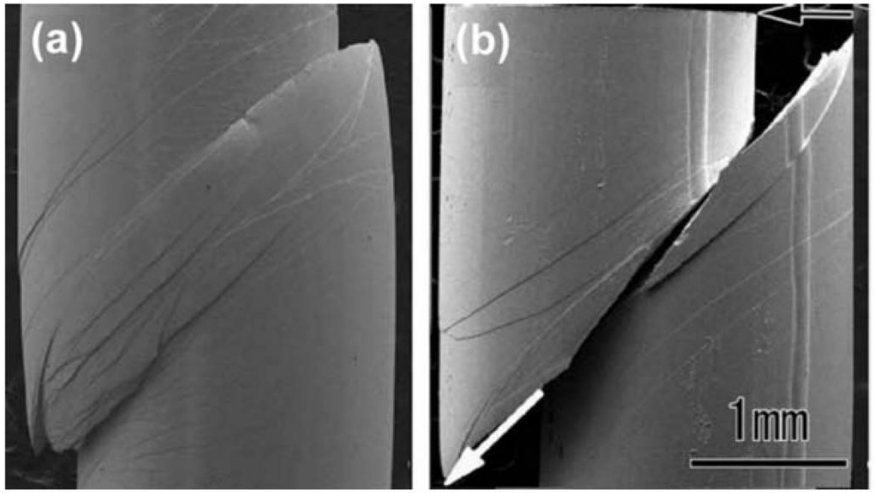


- Disordered
- Structure similar to a liquid, but without atomic mobility
- Dislocations don't exist
- Crystal plasticity not applicable

Plastic deformation of BMGs happens through highly **localized shear bands**.

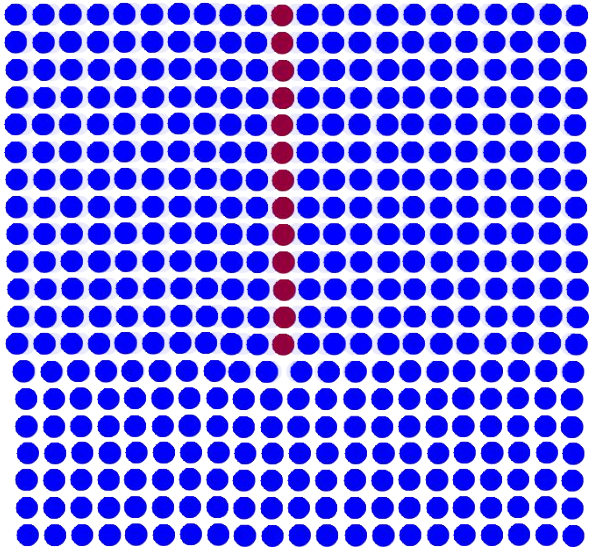
Strain patterns are formed, with **45 degrees orientation** with respect to applied stress axis.





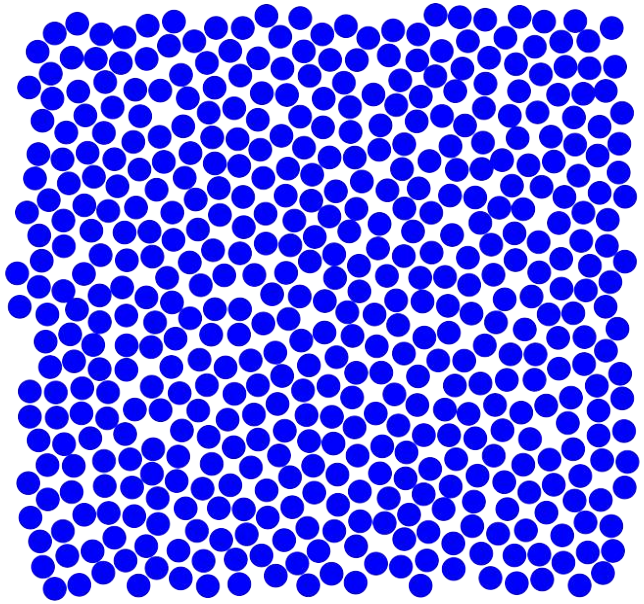
1. Bulk Metallic Glasses

Crystalline structure



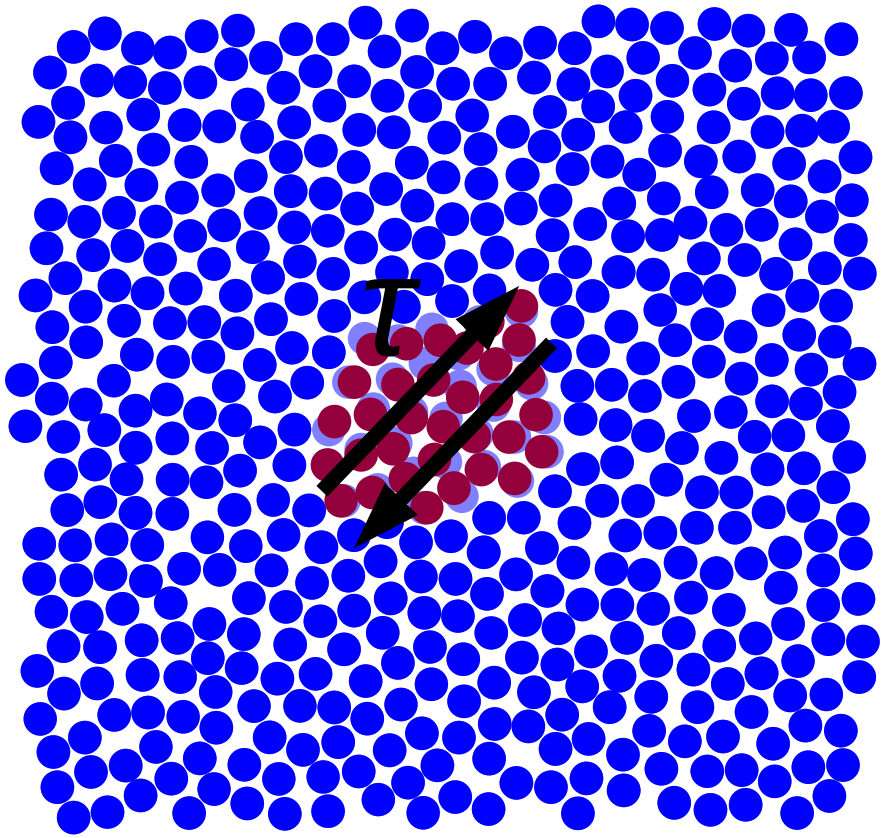
- Periodic and ordered
- Helpful to make crystal plasticity theory
- Well-defined dislocations

Amorphous structure



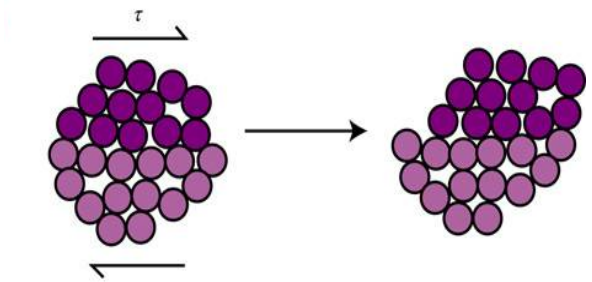
- Disordered
- Structure similar to a liquid, but without atomic mobility
- Dislocations don't exist
- Crystal plasticity not applicable

1. Bulk Metallic Glasses



Individual plastic events in BMG are called **shear transformations** (ST)

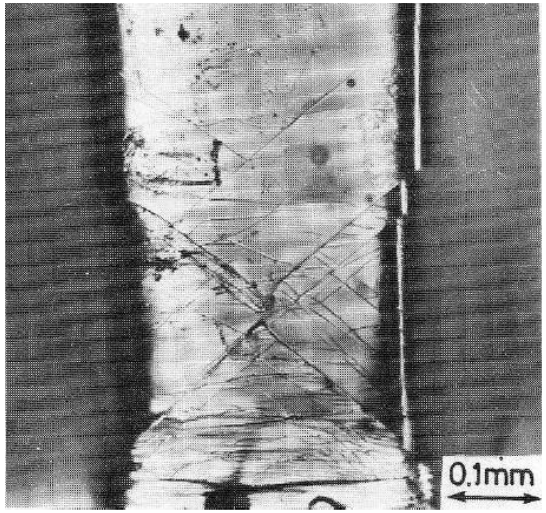
- local r**earrangement** of atoms
- Triggered by **shear stress**
- Not a feature of the structure
- Not well defined volume nor shape

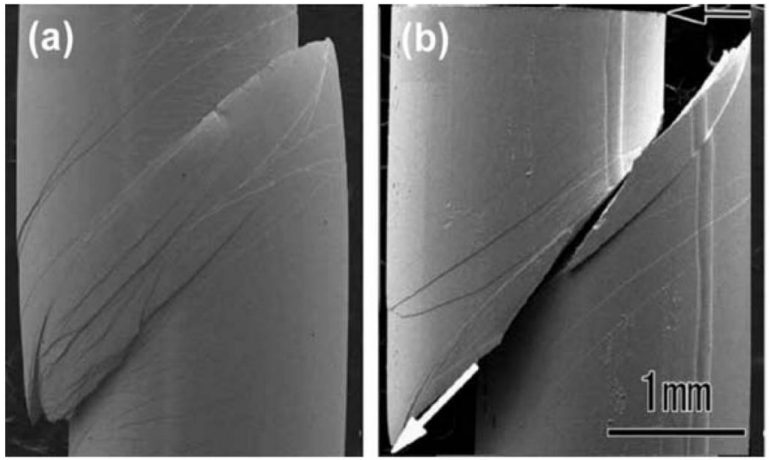


1. Bulk Metallic Glasses

The **elastic interaction** of STs leads to the formation of **shear bands**

Eventually, one macroscopic shear band produces the **failure** of the sample



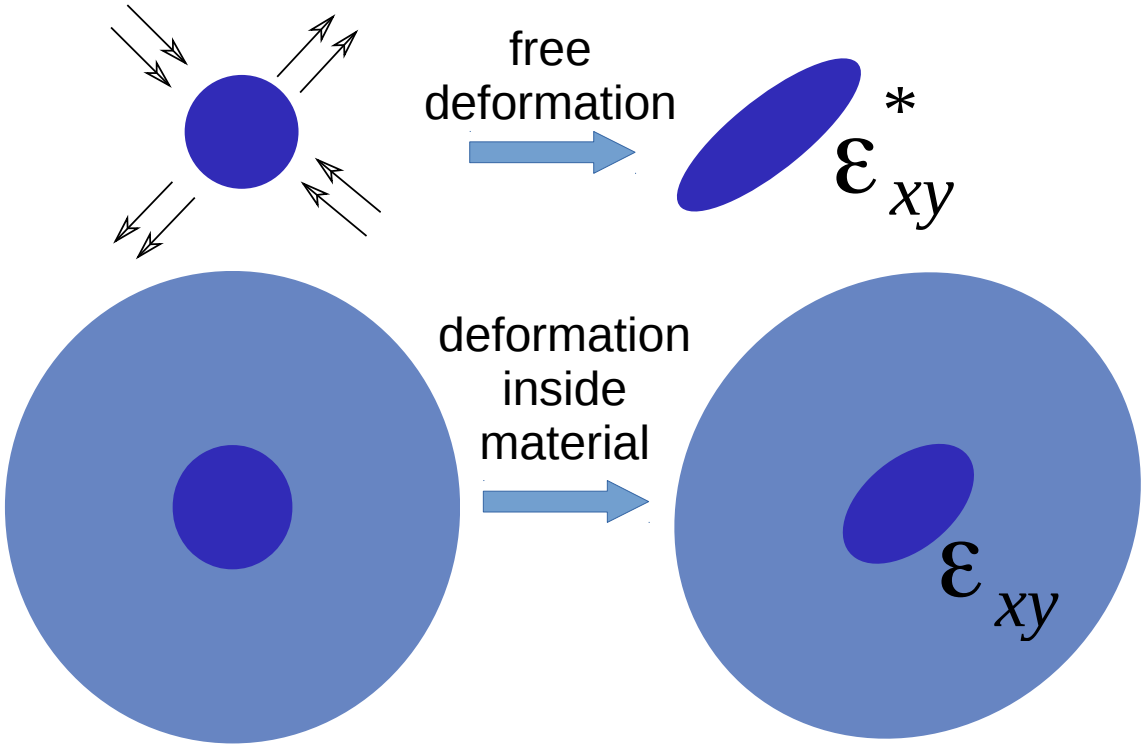


- elastic properties **similar** to conventional metals
- considerably **higher yield** stress
- strong **shear localization**
- extreme **brittleness**
- evidence of **plasticity at microscale**

2.1. Representation of plastic events with Green's functions

2.1. Representation of plastic events

A ST can be represented as an **inclusion** undergoing a shear deformation



The deformation of a “free” inclusion is different from its deformation inside the material. This is due to the reaction forces of the material, which creates a **stress field around the inclusion**

2.1. Representation of plastic events

How to find the **stress field of a inclusion**? Solving the equations of elasticity

$$\nabla \cdot \sigma + \boldsymbol{f} = 0$$

Elasticity equations

$$\boldsymbol{f} = \delta(\boldsymbol{r})$$

Infinitesimally small deformation
(Dirac's delta function)

The solution is called Green's function G

For an **inclusion undergoing a shear deformation** the solution is

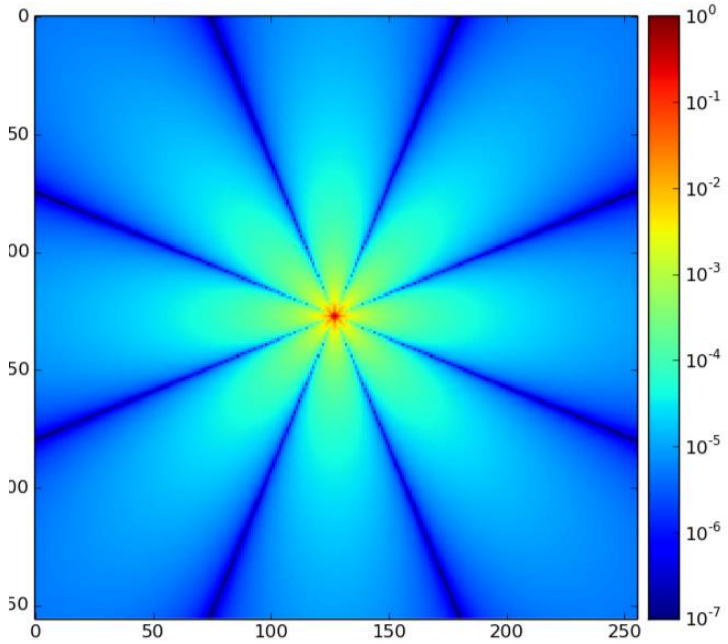
$$G(r,\theta) = \sigma_{xy}(r,\theta) \propto \frac{\cos(4\theta)}{r^2}$$

The **stress associated to a plastic event** in a BMG can be represented with that function

Assumptions to obtain this solution:

* Isotropic and homogeneous material
* **Infinite system size**
* Only shear component of stress

2.1. Representation of plastic events



Shear stress of an inclusion
in a infinite size system

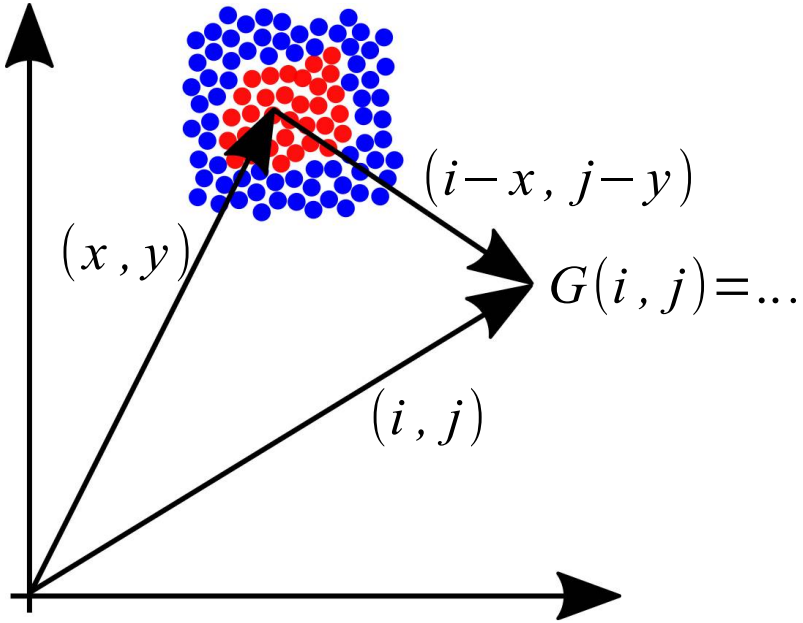
Task #1

Plot the Green's function in 2D using logarithmic scale

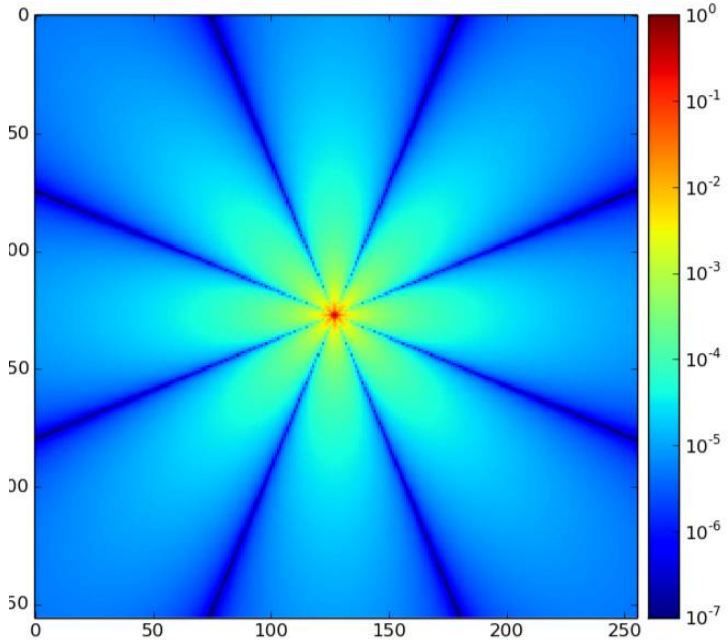
$$G(r,\theta)=\frac{\cos(4\theta)}{r^2}$$

Hint

For a fixed *(x,y)*:
iterate over the system sites *(i,j)* and calculate G using the relative position to *(x,y)*



2.1. Representation of plastic events



Shear stress of an inclusion in a infinite size system

Task #1

Plot the Green's function in 2D using logarithmic scale

$$G(r,\theta)=\frac{\cos(4\theta)}{r^2}$$

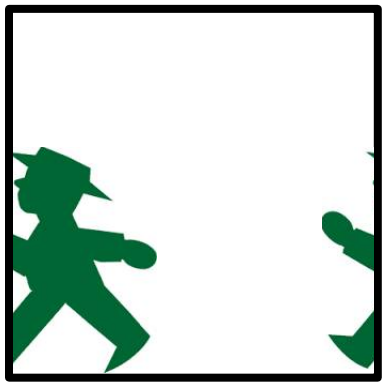
```python
from matplotlib.colors import LogNorm
import numpy as np
import matplotlib.pyplot as plt

# square matrix length
L = 128

# define the position of the inclusion in the middle of the system
x = int(L/2)
y = int(L/2)

# allocate the matrix to hold the Green's function
G = np.zeros([L,L])

# iterate over all positions of the matrix
for i in range(0,L):
    for j in range(0,L):

        # calcualte the angle
        # if the is a problem with the arctan function, then angle use angle 0
        try: z = np.arctan(float(j-y)/float(i-x))
        except: z = 0.

        # calculate distance between position and inclusion location
        r = np.sqrt((i-x)**2 + (j-y)**2)

        # if distance is 0, do this to avoid dividing by 0
        if r>1e-12: G[i,j] += np.cos(4.*z)*np.power(r,-2.)
        else: G[i,j] += 0.

# create a figure
fig1 = plt.figure(figsize=(7,7),facecolor='white')
# add plot to the figure
ax1 = fig1.add_subplot(111)

# plot the matrix with log scale
im = ax1.imshow(np.abs(G), norm=LogNorm(vmin=1e-7, vmax=G.max()))

# add color bar
plt.colorbar(im)
```

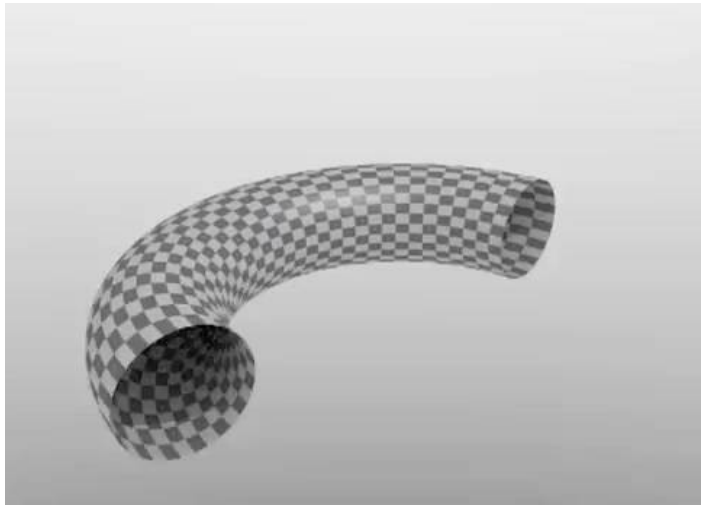
However, we can't simulate an **infinite system,** so we can't use that Green's function directly in our simulation

We need the Green's function for a finite system, but it **doesn't exist analytically**

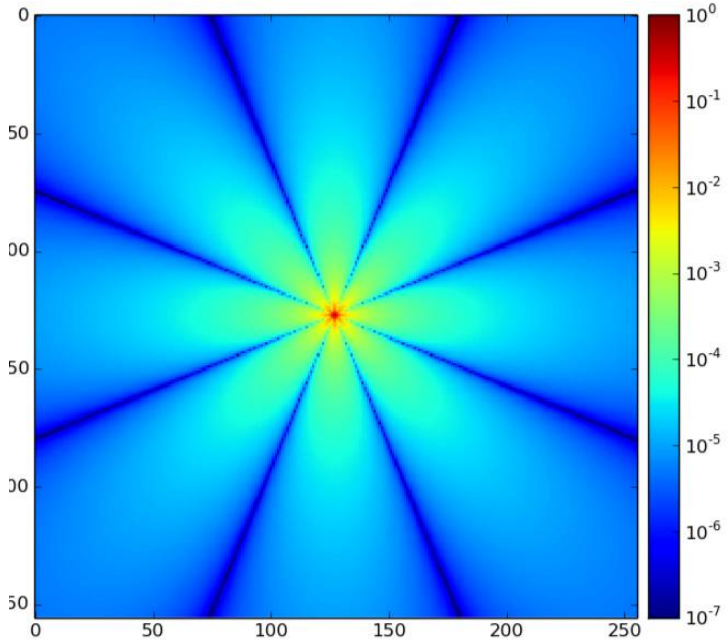
We can still simulate a **periodic system**, i.e., without surfaces but finite size

In a periodic system the surfaces are connected to each other, so the system repeats after reaching the boundary

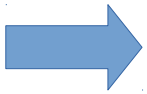


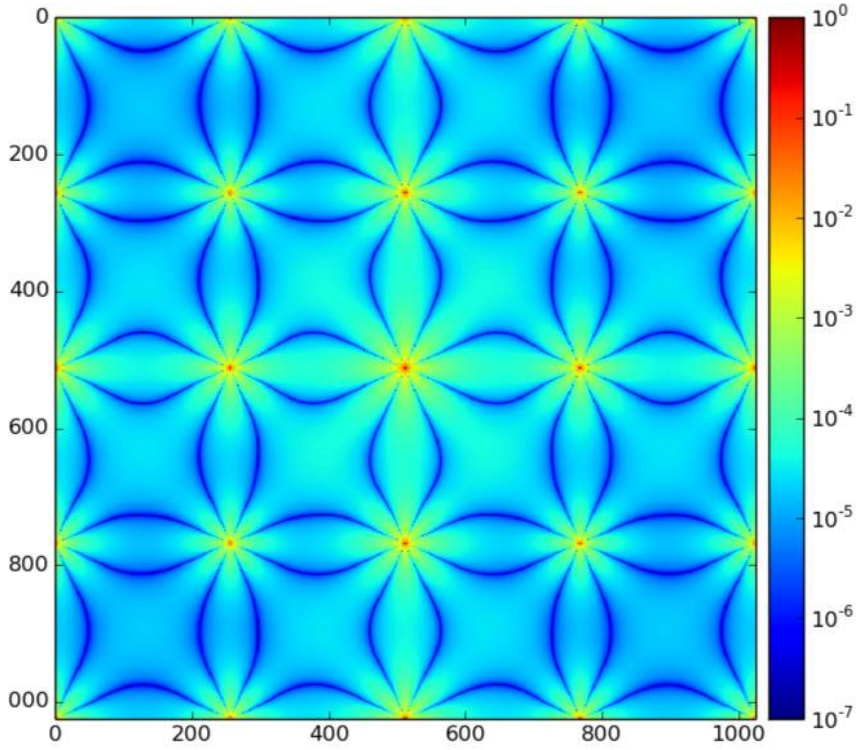


2.1. Representation of plastic events

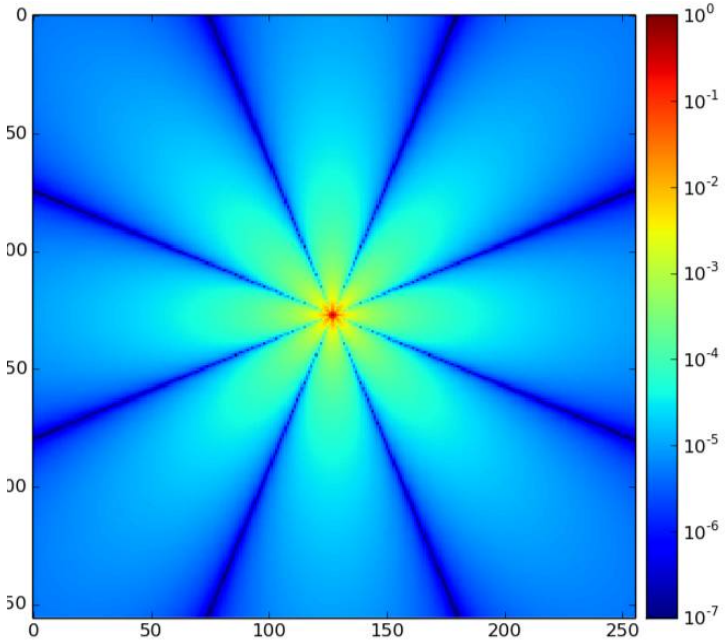


Shear stress of an inclusion
in a infinite size system

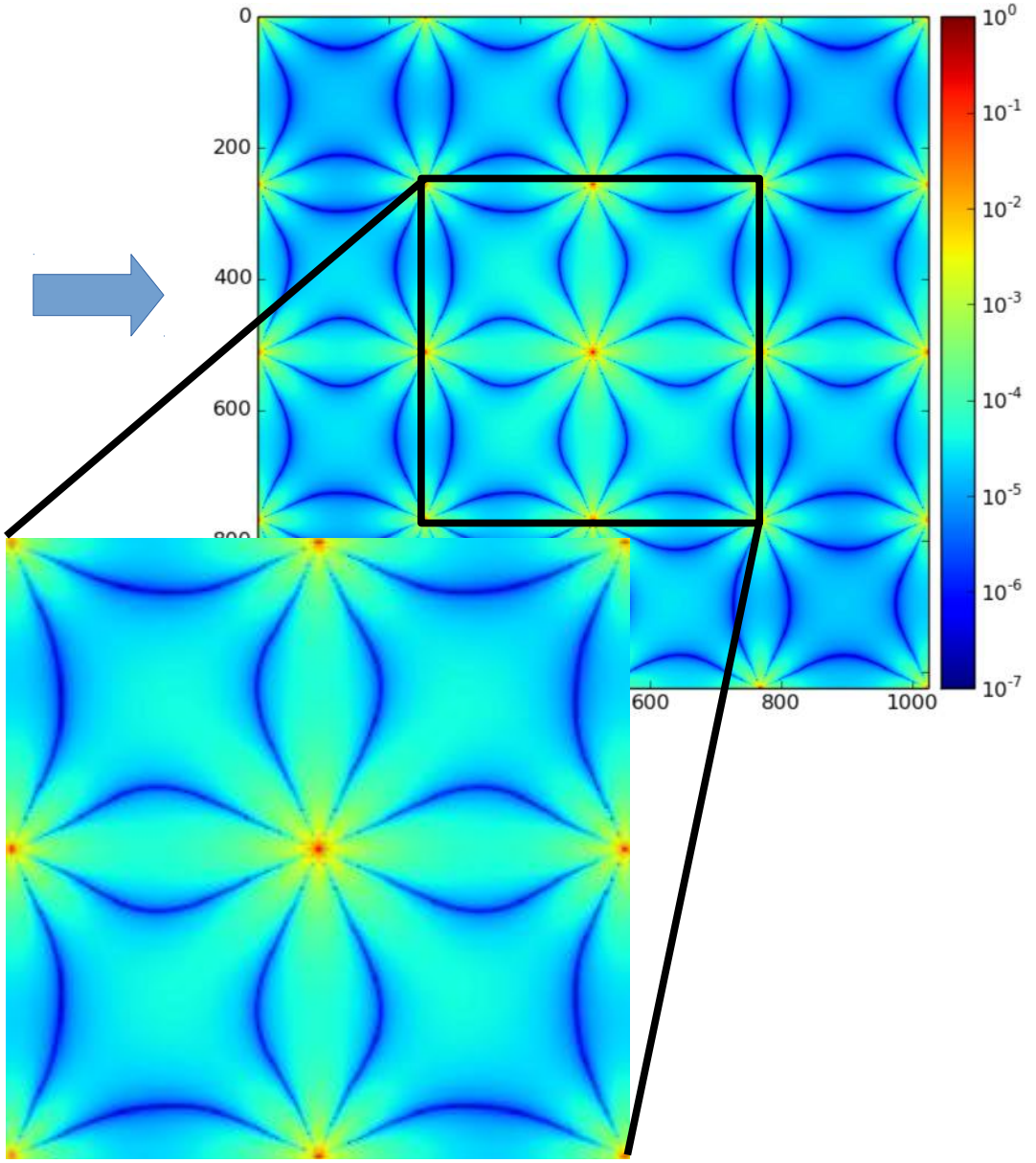


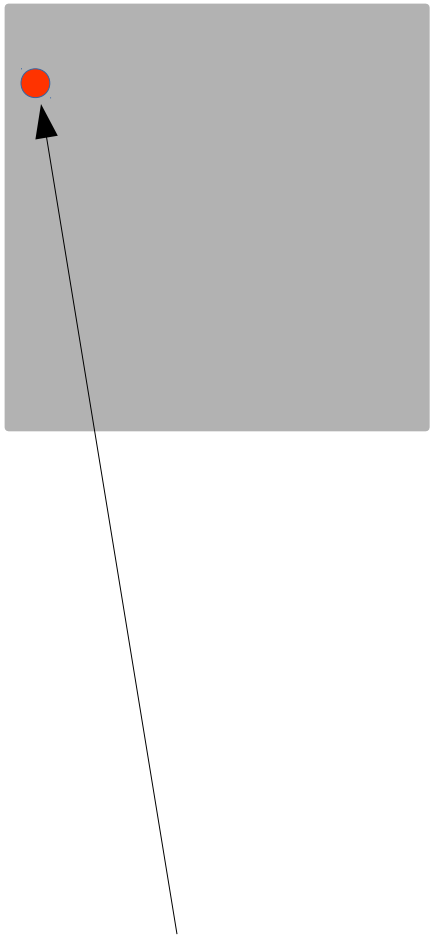


2.1. Representation of plastic events

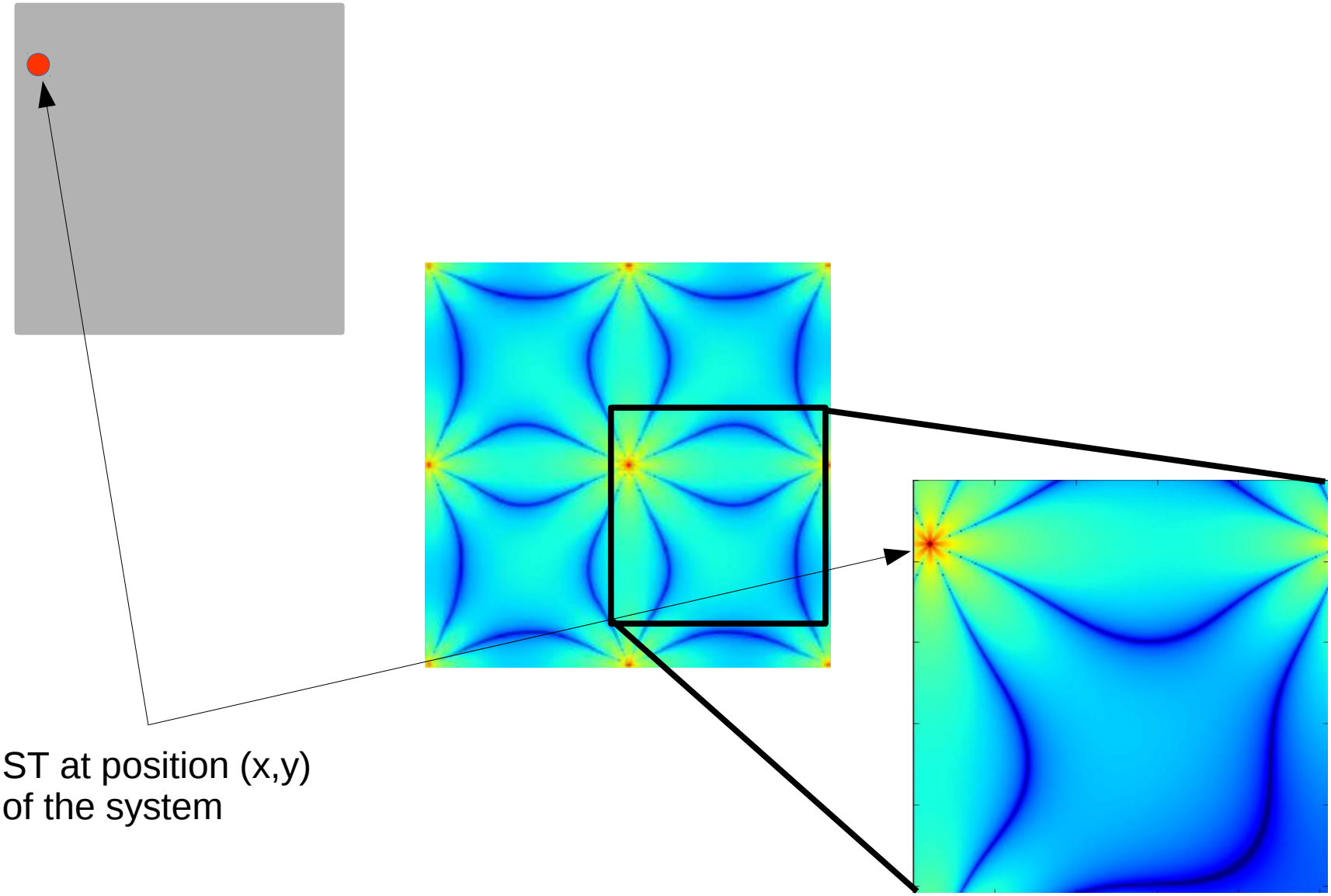


Shear stress of an inclusion
in a infinite size system

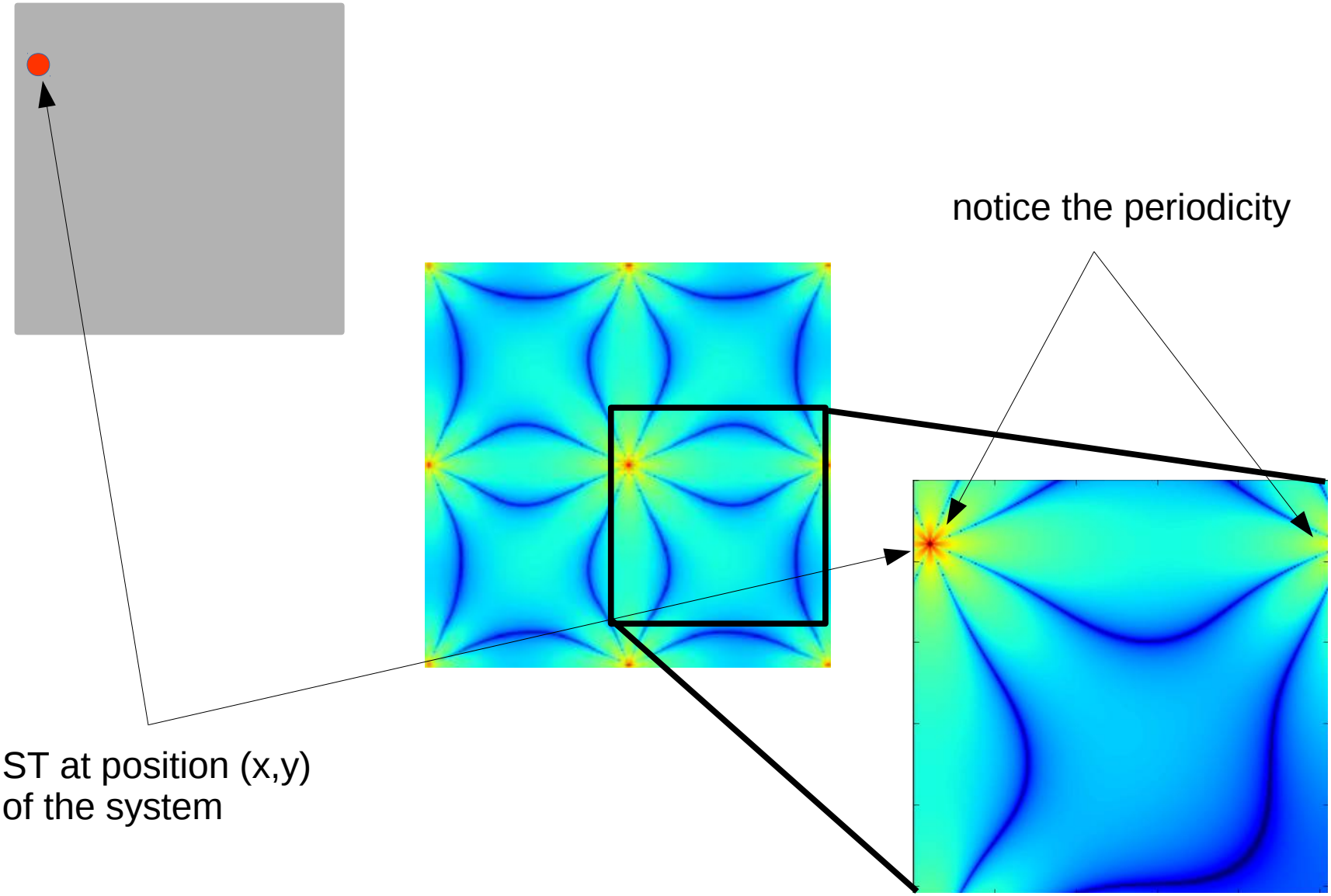


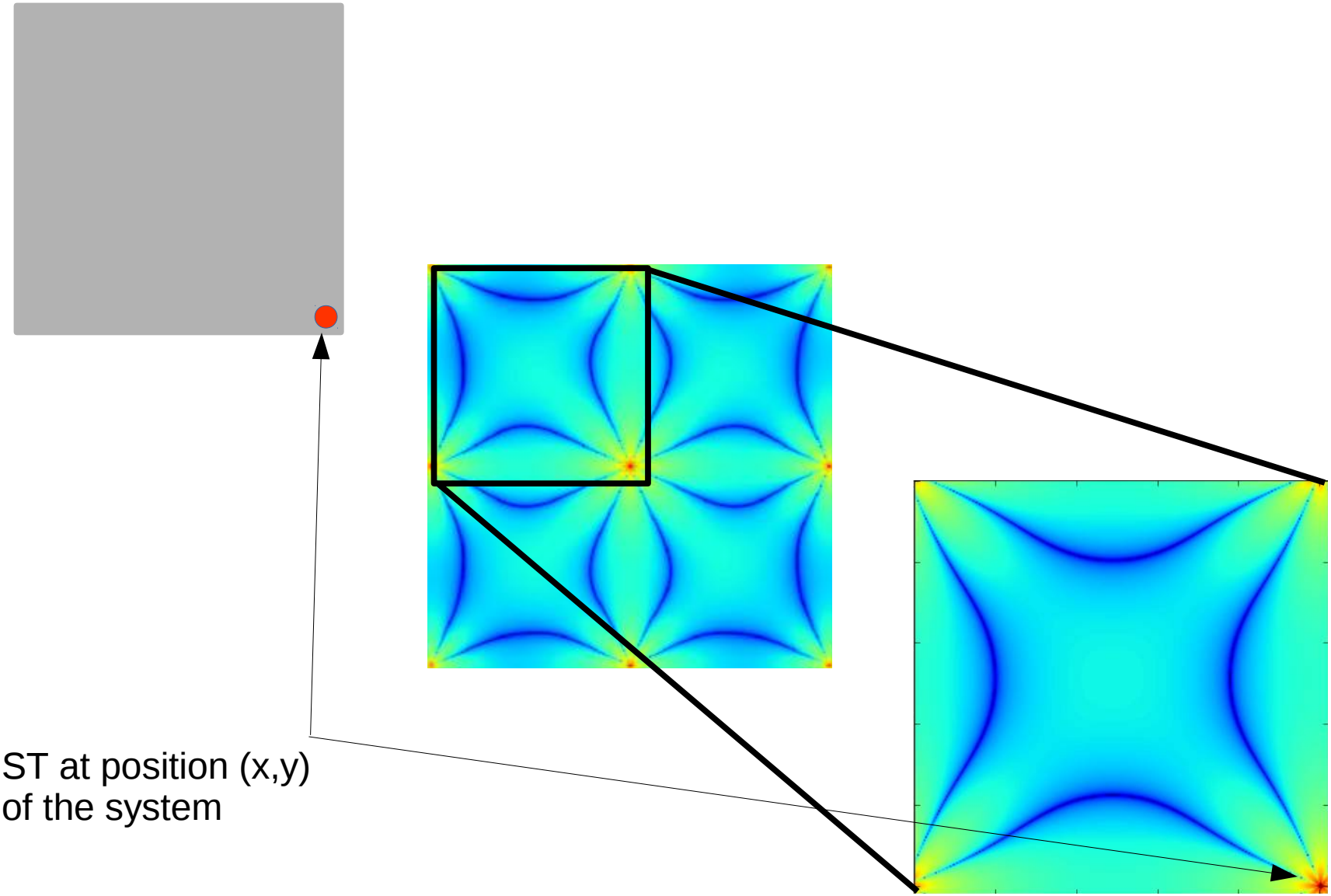


ST at position (x,y)
of the system



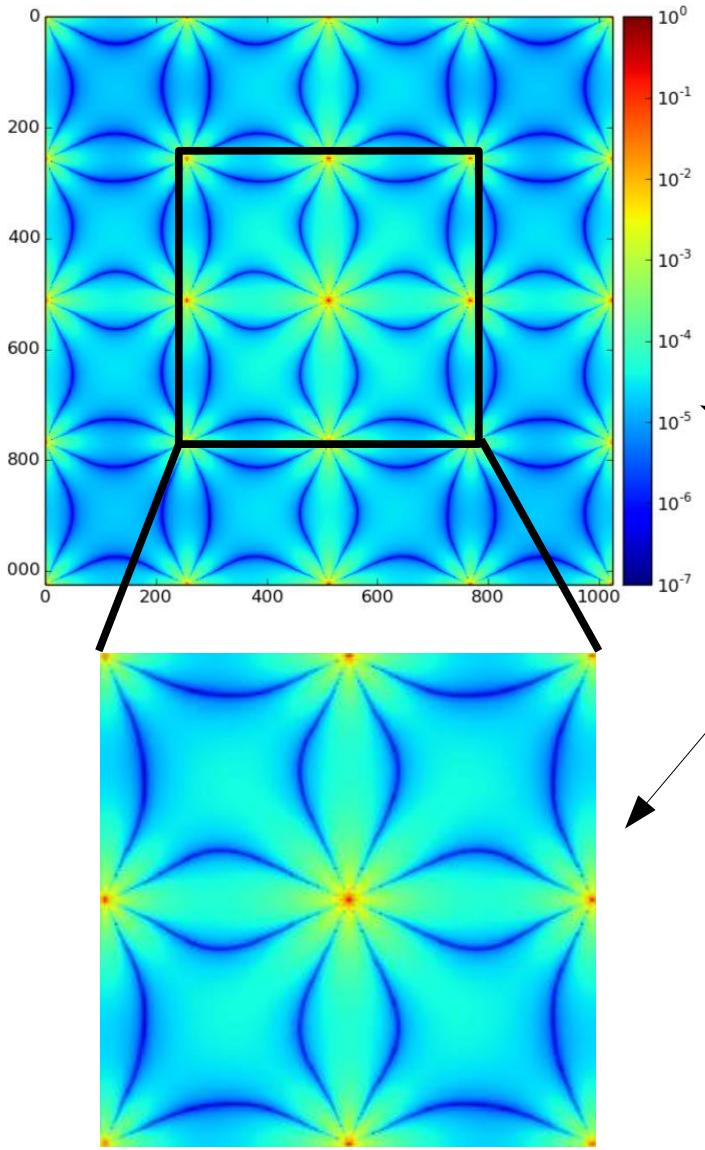
ST at position (x,y)
of the system


notice the periodicity
ST at position (x,y)
of the system



ST at position (x,y)
of the system

2.1. Representation of plastic events



Task #2

Make the reduced array of periodic inclusions

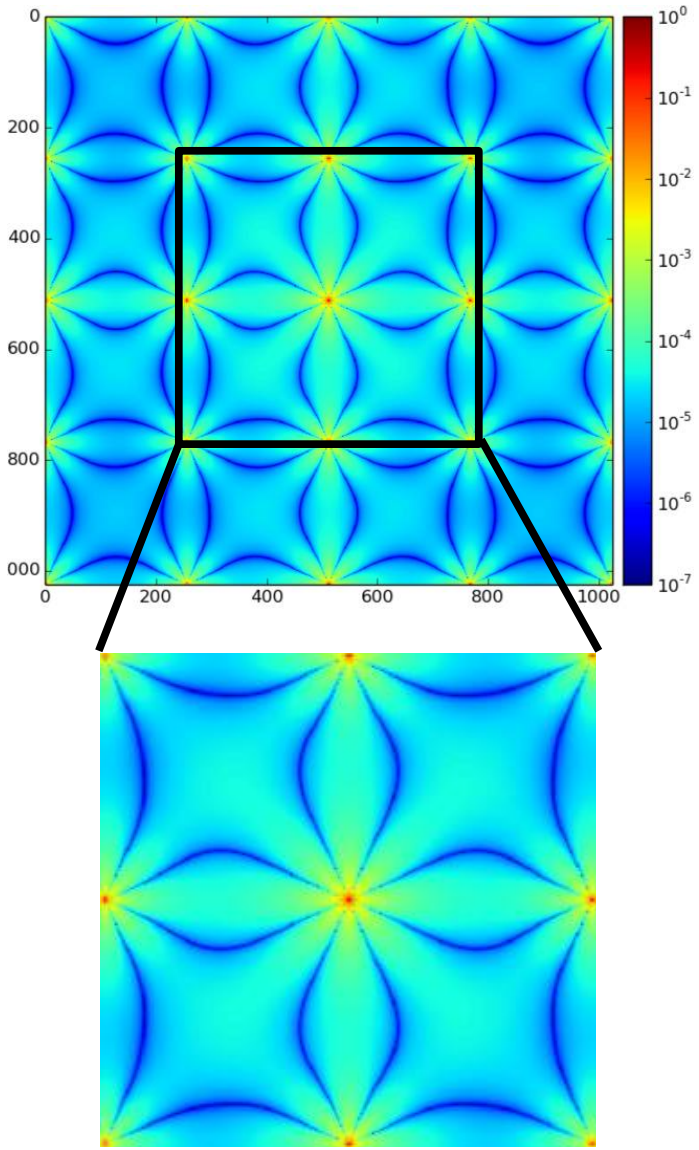
Hint

This is only for visualization

Do this directly instead, but also using

Inclusions beyond the matrix limits

2.1. Representation of plastic events



Task #2

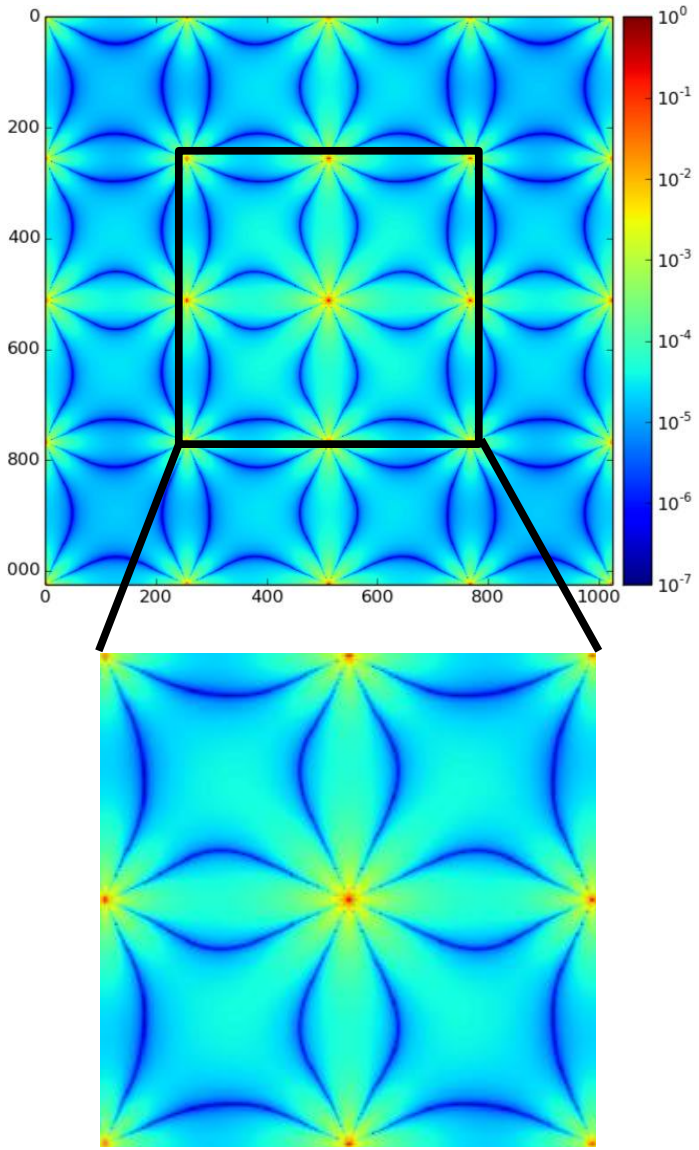
Make the reduced array of periodic inclusions

Hint

Put the code in a function that takes the inclusion position as argument. Then call the function for the position of the Inclusions you want adding up the results

```python
from matplotlib.colors import LogNorm
import numpy as np
import matplotlib.pyplot as plt

def make_single_inclusion(x,y,L):

    # allocate the matrix to hold the Green's function
    G = np.zeros([L,L])

    # iterate over all positions of the matrix
    for i in range(0,L):
        for j in range(0,L):

            # calcualte the angle
            # if there is a problem with the arctan function, then use
            try: z = np.arctan(float(j-y)/float(i-x))
            except: z = 0.

            # calculate distance between position and inclusion locati
            r = np.sqrt((i-x)**2 + (j-y)**2)

            # if distance is 0, do this to avoid dividing by 0
            if r>1e-12: G[i,j] += np.cos(4.*z)*np.power(r,-2.)
            else: G[i,j] += 0.

    return G


# square matrix length
L = 128

# define the position of the inclusion in the middle of the system
x = int(L/2)
y = int(L/2)

###################################
G = make_single_inclusion(x,y,L)
###################################

# create a figure
fig1 = plt.figure(figsize=(7,7),facecolor='white')
# add plot to the figure
ax1 = fig1.add_subplot(111)

# plot the matrix with log scale
im = ax1.imshow(np.abs(G), norm=LogNorm(vmin=1e-7, vmax=G.max()))

# add color bar
plt.colorbar(im)
```

2.1. Representation of plastic events



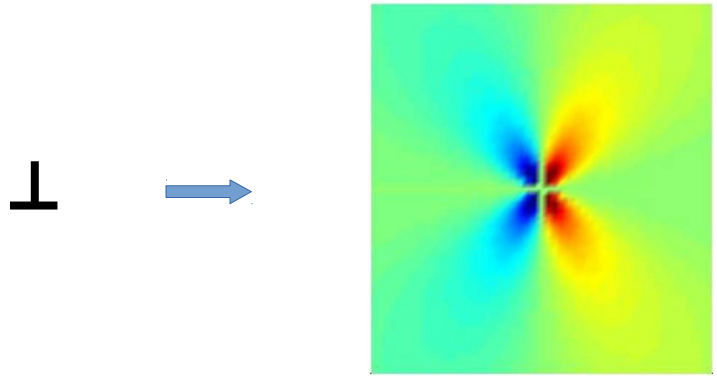
Task #2

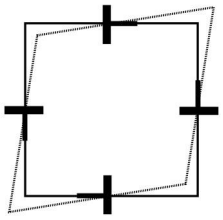
Make the reduced array of periodic inclusions

```python
from matplotlib.colors import LogNorm
import numpy as np
import matplotlib.pyplot as plt


def make_single_inclusion(x,y,L):

    # allocate the matrix to hold the Green's function
    G = np.zeros([L,L])

    # iterate over all positions of the matrix
    for i in range(0,L):
        for j in range(0,L):

            # calcualte the angle
            # if there is a problem with the arctan function, then use angle 0
            try: z = np.arctan(float(j-y)/float(i-x))
            except: z = 0.

            # calculate distance between position and inclusion location
            r = np.sqrt((i-x)**2 + (j-y)**2)

            # if distance is 0, do this to avoid dividing by 0
            if r>1e-12: G[i,j] += np.cos(4.*z)*np.power(r,-2.)
            else: G[i,j] += 0.

    return G


L = 128

G = np.zeros([L,L])

# define x and y positions of all inclusion images
X = [-64,0,64,127,127+64]
Y = [-64,0,64,127,127+64]

# create array of images putting an inclusion at each position
for x in X:
    for y in Y:
        G += make_single_inclusion(x,y,L)

# put an inclusion at x_inc and y_inc and extract the stress from the array
# of images
x_inc = 20
y_inc = 50
g = G[(L/2-x_inc):(L-x_inc), (L/2-y_inc):(L-y_inc)]

# create a figure
fig1 = plt.figure(figsize=(7,7),facecolor='white')
# add plot to the figure
ax1 = fig1.add_subplot(111)

# plot the matrix with log scale
im = ax1.imshow(np.abs(G), norm=LogNorm(vmin=1e-7, vmax=G.max()))

# add color bar
plt.colorbar(im)
```

As seen before, the Green's function gives **problems with singularities**, which are even bigger in the periodic array.

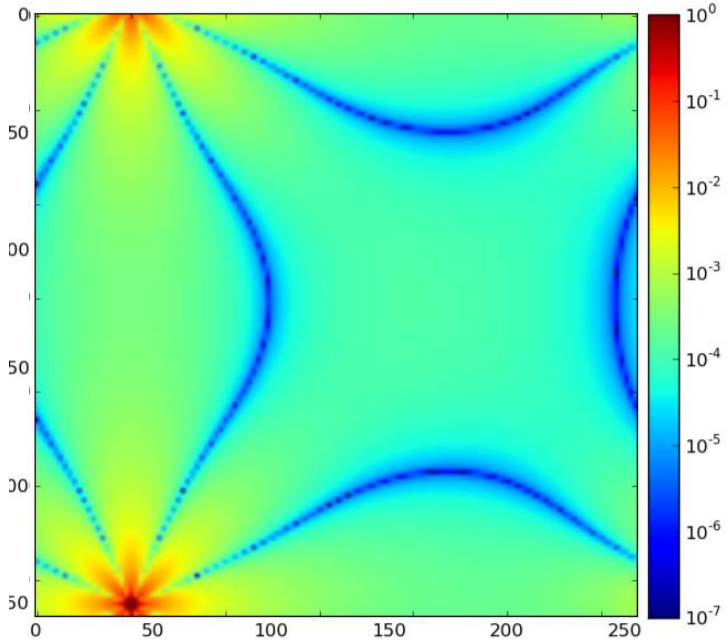
A way of solving this by r**eplacing the Green's function by the field of 4 edge dislocations**. It looks the same away from the inclusion, but without singularity in the center.





$$G(r,\theta)=\frac{\cos(4\theta)}{r^2}$$

2.1. Representation of plastic events



Shear stress in periodic system

Task #3

Learn how to use the script “greens_function.py” to get the stress of an inclusion located at any Place in a periodic system

```
1 import greens_function
2 from matplotlib.colors import LogNorm
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 # system size
7 L = 64
8
9 # number of periodic images
10 n = 2
11
12 # get the periodic array of Green's functions
13 K = greens_function.get_periodic_kernel(L,n)
14
15 # get the stress field of a inclusion centered at (x,y)
16 x = 10
17 y = 20
18 k = greens_function.get_inclusion_stress(K,10,20)
19
20 # the sum of the (internal) stress must be 0 (3rd Newton Law)
21 print 'Sum of stress = ', k.sum()
22
23 # plot it
24 fig1 = plt.figure(figsize=(7,7),facecolor='white')
25 ax1 = fig1.add_subplot(111)
26 im = ax1.imshow(np.abs(k), norm=LogNorm(vmin=1e-7, vmax=K.max()))
27 plt.colorbar(im)
```

2.2. Model of plasticity

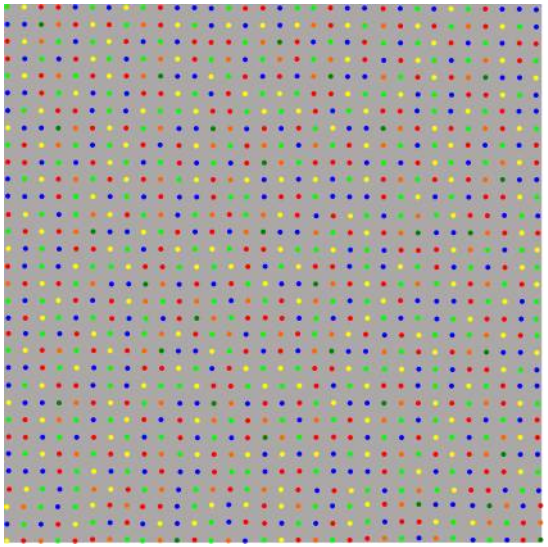
2.2. Model of plasticity

System setup

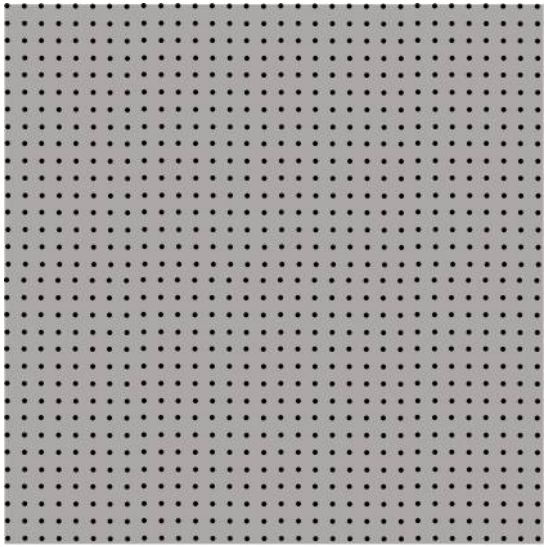
- **Periodic** boundaries
- **Homogeneous** external stress (pure shear)
- Material discretized in 3 fields: **shear stress, plastic strain & energy barrier**

Model

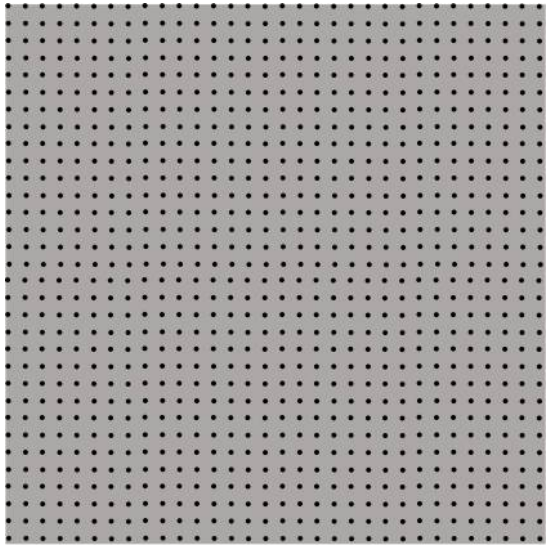
- **Random energy barrier** to each site to represent amorphous structure
- Where a ST happens, a **unit is added** to the plastic strain matrix
- **Shear stress is updated** with the stress of the ST (inclusion)



Energy barrier



Shear stress



Plastic strain

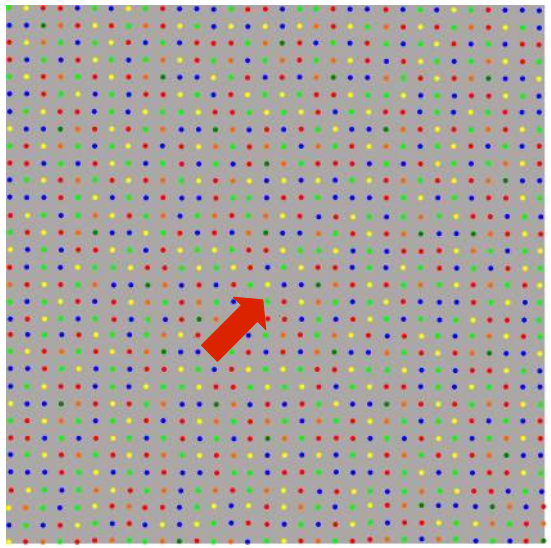
2.2. Model of plasticity

System setup

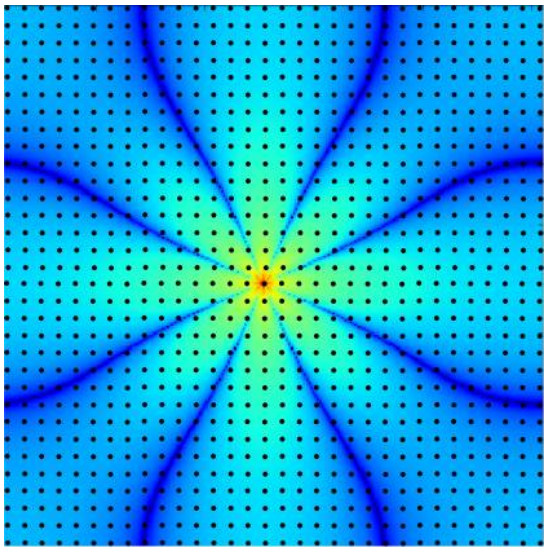
- **Periodic** boundaries
- **Homogeneous** external stress (pure shear)
- Material discretized in 3 fields: **shear stress, plastic strain & energy barrier**

Model

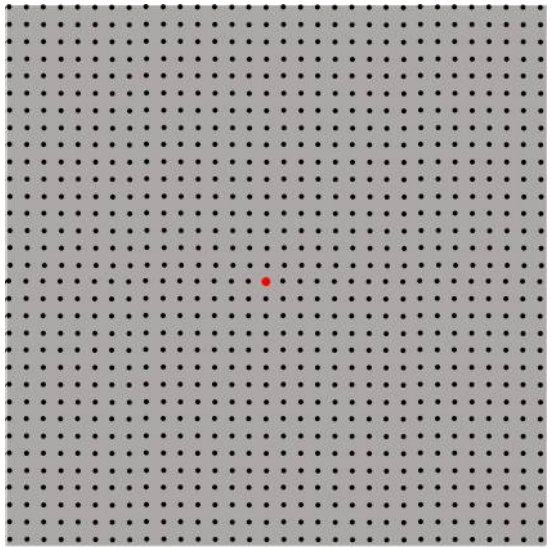
- **Random energy barrier** to each site to represent amorphous structure
- Where a ST happens, a **unit is added** to the plastic strain matrix
- **Shear stress is updated** with the stress of the ST (inclusion)



Energy barrier



Shear stress



Plastic strain

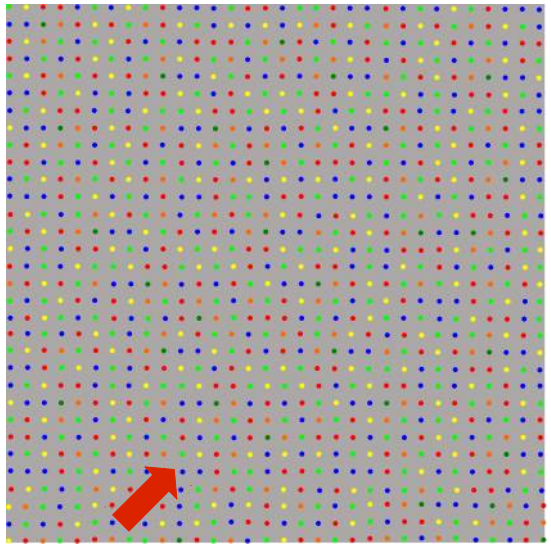
2.2. Model of plasticity

System setup

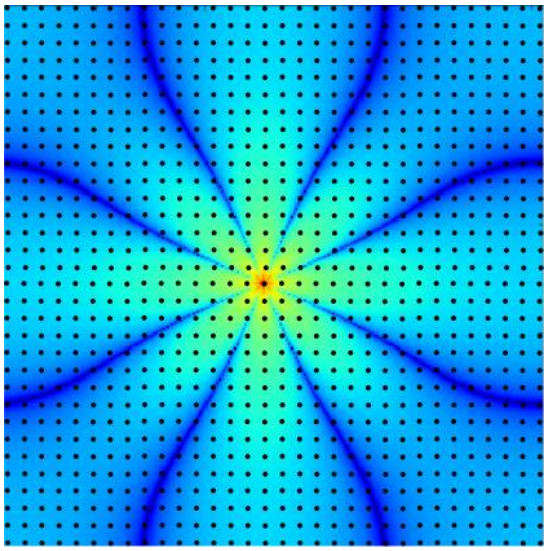
- **Periodic** boundaries
- **Homogeneous** external stress (pure shear)
- Material discretized in 3 fields: **shear stress, plastic strain & energy barrier**

Model

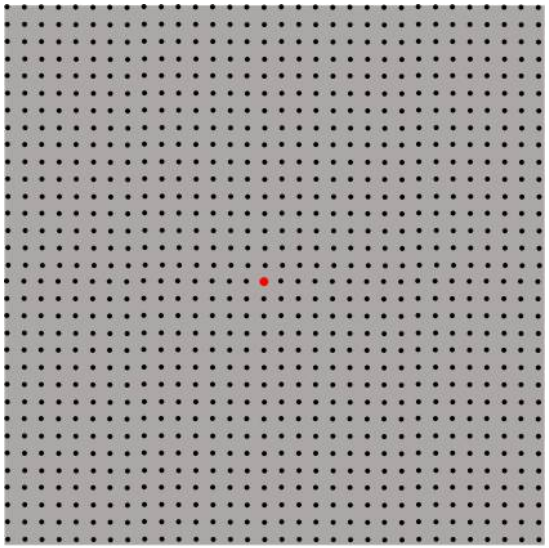
- **Random energy barrier** to each site to represent amorphous structure
- Where a ST happens, a **unit is added** to the plastic strain matrix
- **Shear stress is updated** with the stress of the ST (inclusion)



Energy barrier



Shear stress



Plastic strain

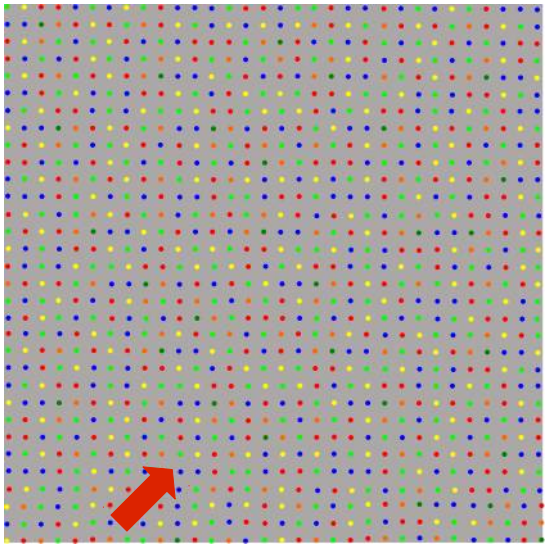
2.2. Model of plasticity

System setup

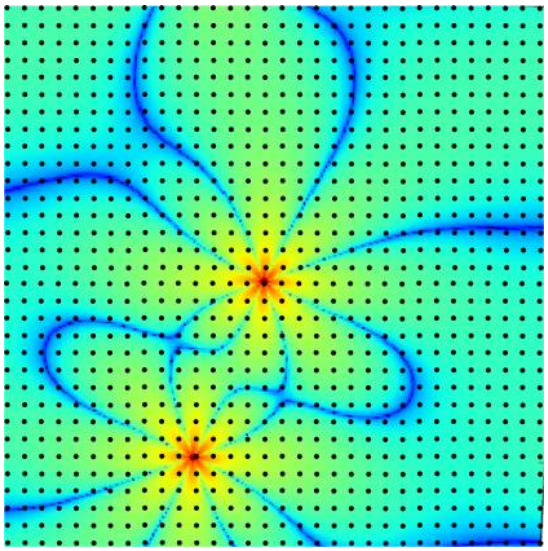
- **Periodic** boundaries
- **Homogeneous** external stress (pure shear)
- Material discretized in 3 fields: **shear stress, plastic strain & energy barrier**

Model

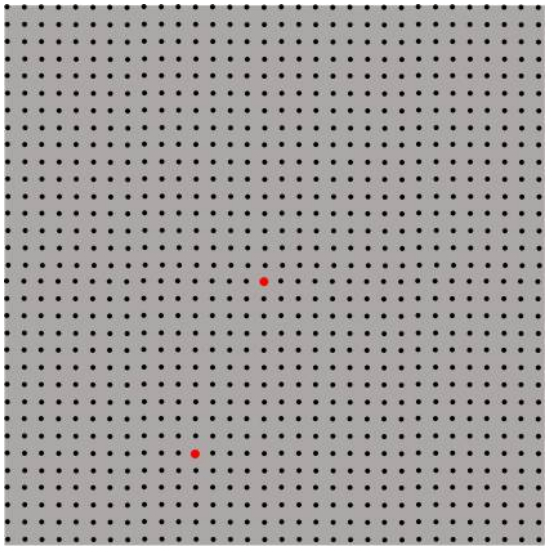
- **Random energy barrier** to each site to represent amorphous structure
- Where a ST happens, a **unit is added** to the plastic strain matrix
- **Shear stress is updated** with the stress of the ST (inclusion)



Energy barrier



Shear stress



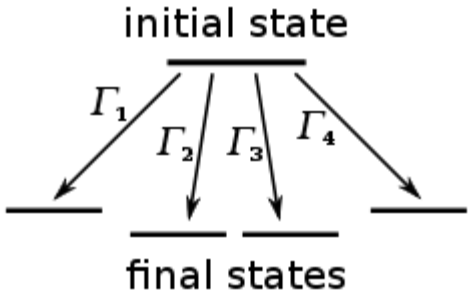
Plastic strain

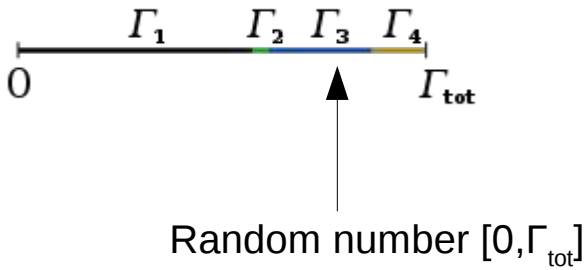
2.3. Evolution of the system with the Kinetic Montecarlo method

We know how to represent the system and STs, but how do we know where and when a ST happens?

Kinetic Montecarlo method

- Simulates the time evolution of a system (decides what happens next)
- Needs to know all possibilities to decide what happens next
- The decision is based on probability
- States are defined in terms of energy. We don't need to know about atomistic details

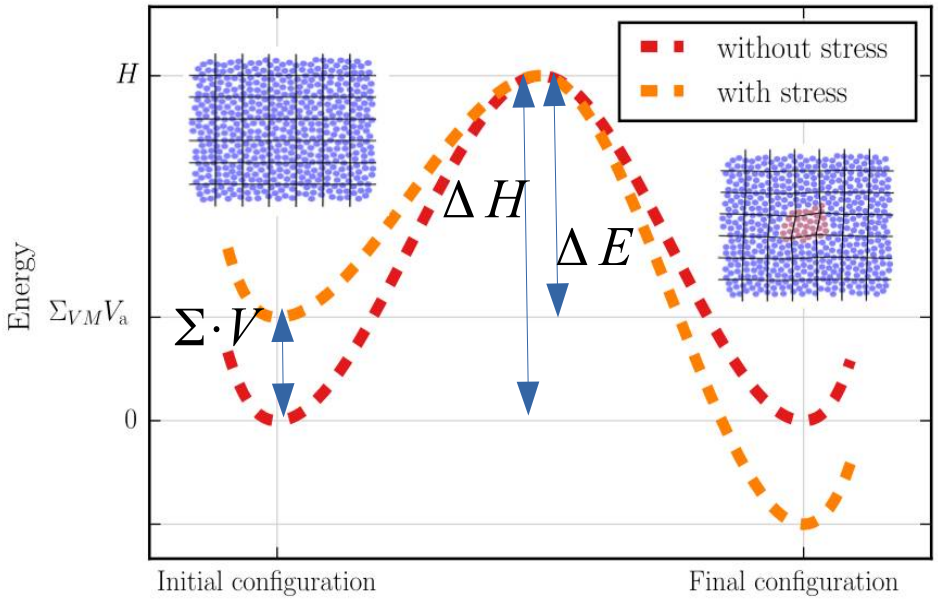




In the following is convenient to think in the system as a 1D vector. KMC doesn't care about 2D coordinates, only needs vectors with values to make a decision

Kinetic Montecarlo method

1) Calculate the **rates**. We need to know the energy that each position would require to deform.



$$\Delta E_i = \Delta H_i - \Sigma_i V$$

extra energy needed to jump the barrier

energy barrier at 0 stress

stress

Rates are related with the probability of an event to happen. The lower the barrier, the more likely it is to jump it due to thermal fluctuations of energy kT

$$r_i = \nu \exp\left(-\frac{\Delta E_i}{kT}\right) = \nu \exp\left(-\frac{\Delta H_i - \Sigma_i V}{kT}\right)$$

2.3. Evolution of the system

We can rewritte the rates in a more useful way for simulation, identifying simulation units and truly independent parameters

$$r_i = \nu \exp\left(-\frac{\Delta E_i}{kT}\right) = \nu \exp\left(-\frac{\Delta H_i - \Sigma_i V}{kT}\right) = \dots$$

We assume the barriers to be uniformly distributed

$$\Delta H = f \cdot 2\,\overline{\Delta H} \qquad f \text{ uniform random [0,1)}$$

$$\dots = \nu \exp\left(-\frac{2\,\Delta \overline{H}\, f_i - \Sigma_i V}{kT}\right) = \nu \exp\left(-P\,\frac{f_i - \hat{\Sigma}_i}{kT}\right)$$

Approx. 1 eV

$$P = \frac{2\,\Delta \overline{H}}{kT} \approx \frac{2.3 \cdot 10^4\, kelvin}{T}$$

Approx. 1 nm³

$$\hat{\Sigma}_i = \Sigma_i \frac{V}{2\,\Delta \overline{H}} \approx \Sigma_i / 320\, MPa$$

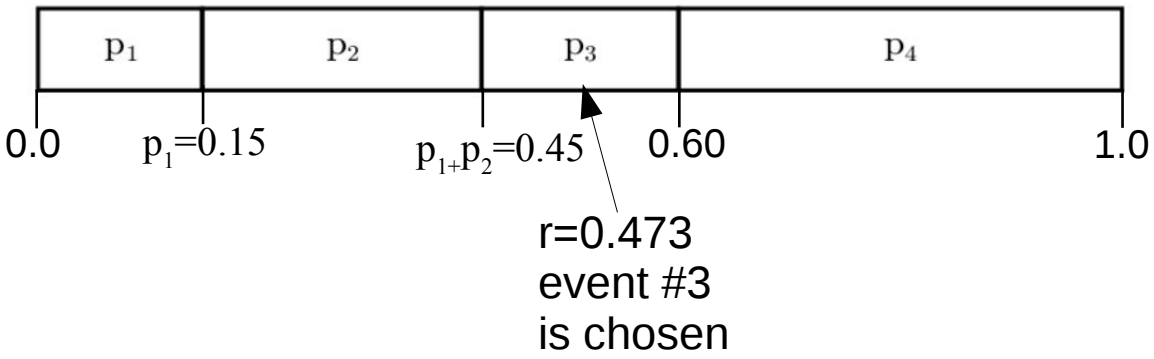
2.3. Evolution of the system

2) Calculate the **probability** of each possible event using the rates

$$R = \sum_i r_i \quad \Rightarrow \quad P_i = \frac{r_i}{R}$$

3) **Choose an event** using the probabilities

Make a list of accumulated probabilities



Given a random uniform number between [0,1), find where it lies the list

Notice: the higher the probability, the more likely it is that the random number chooses the event associated with that probability

We only choose where the event happens. Other things like amount of plastic deformation of the event are constant for us.

2.3. Evolution of the system

4) Calculate the **time that has passed since the previous event** till the chose one

$$\Delta t = -\frac{\log(rand)}{R}$$

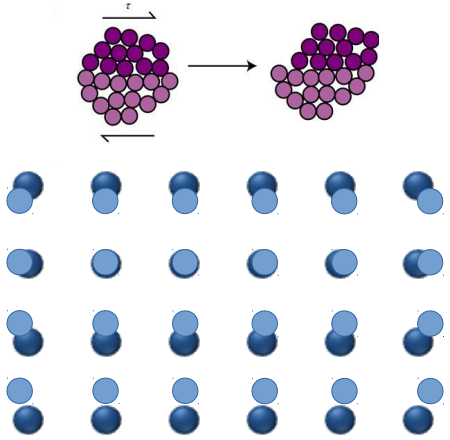
$$r_i = \nu \exp\left(-\frac{\Delta E_i}{kT}\right)$$

This constant is defining the time units. It has units of frequency, i.e., inverse of time. We can use the diffusion attempt frequency for it (the typical frequency of atomic thermal movements).

$$\Delta t = \Delta \hat{t} / \nu(T)$$

real time

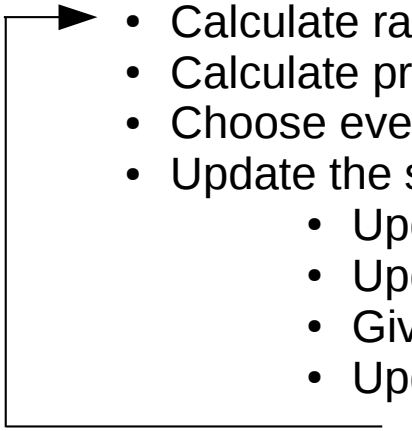
time in the simulation



$$\nu(T) = \nu_0 \exp\left(-\frac{\Delta H}{kT}\right) = \nu_0 \exp(-P)$$

2.3. Evolution of the system

- Create stress vector
- Create strain vector
- Create energy barrier vector
- Add constant external stress to the stress vector



- Calculate rates
- Calculate probabilities
- Choose event (i.e., the site where it happens)
- Update the system:
    - Update stress by placing an inclusion in the chosen site
    - Update the strain matrix by adding a strain unit to the chosen site
    - Give a new random energy barrier to the site
    - Update the time

Repeat till max. strain or
time is reached

- Plot results
    - Strain vs. time curve
    - Strain pattern (picture)

```
L = 32
C = 0.05
max_plastic_strain = 10.
external_stress = 0.5
P = 50.
```

Calculate Green's function and store it

```
G = greens_function.get_periodic_kernel(L,2)
```

Allocate in memory vectors and data to represent the system

```
N = L*L
stress = np.zeros(N)
plastic_strain = np.zeros(N)
energy_barrier  = np.random.uniform(0.,1.0,N)
time = 0.
av_plastic_strain = 0.
```

Create some list to save the data that we want to plot at the end

```
time_data = list( )
av_plastic_strain_data = list( )
```

Simulate a single plastic event

```
element = ...
plastic_strain[element] += 1
stress += C*greens_function.get_inclusion_stress_1D(G,element)
energy_barrier[element] = np.random.uniform(0.,1.0)
```

Plot some of these quantities to test that it's working so far

```
plt.imshow(stress.resahe(L,L), cmap='afmhot')
plt.imshow(plastic_strain.resahe(L,L), cmap='afmhot')
```

Simulate a single plastic event

```
element = ...
plastic_strain[element] += 1
stress += C*greens_function.get_inclusion_stress_1D(G,element)
energy_barrier[element] = np.random.uniform(0.,1.0)
```

Plot some of these quantities to test that it's working so far

```
plt.imshow(stress.resahe(L,L), cmap='afmhot')
plt.imshow(plastic_strain.resahe(L,L), cmap='afmhot')
```

When it's working, then let's choose the element with KMC instead of manually

Apply a constant external stress (creep test)

```
stress += external_stress
```

Calculate the list of accumulated probabilities

```
rates = np.exp(-P*(energy_barrier-stress))
probability = rates/sum(rates)
acumulated_probabilitiy = np.cumsum(probability)
```

Choose and event / element from that list

```
r = np.random.uniform(0.,1.)
element = 0
 while acumulated_probabilitiy[element] < r:
       Element += 1

 time += -np.log(r)/sum(rates)
```

```
print element
```

Print the element to see if its working, it element=0 or element=N (the last one) it's probably wrong

Now put everything inside a loop, but keep outside the things that shouldn't be repeated, like allocating vectors or plotting the result

```
stress += external_stress

for _ in range(1000):

    rates = np.exp(-P*(energy_barrier-stress))
    probability = rates/sum(rates)
    acumulated_probabilitiy = np.cumsum(probability)

    r = np.random.uniform(0.,1.)
    element = 0
    while acumulated_probabilitiy[element] < r:
        Element += 1

    time += -np.log(r)/sum(rates)

    plastic_strain[element] += 1
    stress += C*greens_function.get_inclusion_stress_1D(G,element)
    energy_barrier[element] = np.random.uniform(0.,1.0)

plt.imshow(plastic_strain.resahe(L,L), cmap='afmhot')
```

Now save the data in the lists that we created a the beginning, to plot the strain vs time curve

```
while max_average_strain < av_plastic_strain:

    rates = np.exp(-P*(energy_barrier-stress))
    probability = rates/sum(rates)
    acumulated_probabilitiy = np.cumsum(probability)

    r = np.random.uniform(0.,1.)
    element = 0
    while acumulated_probabilitiy[element] < r:
        Element += 1

    time += -np.log(r)/sum(rates)

    plastic_strain[element] += 1
    stress += C*greens_function.get_inclusion_stress_1D(G,element)
    energy_barrier[element] = np.random.uniform(0.,1.0)

    av_plastic_strain = np.average(plastic_strain)
    time_data.append(time)
    av_plastic_strain_data.append(av_plastic_strain)
```

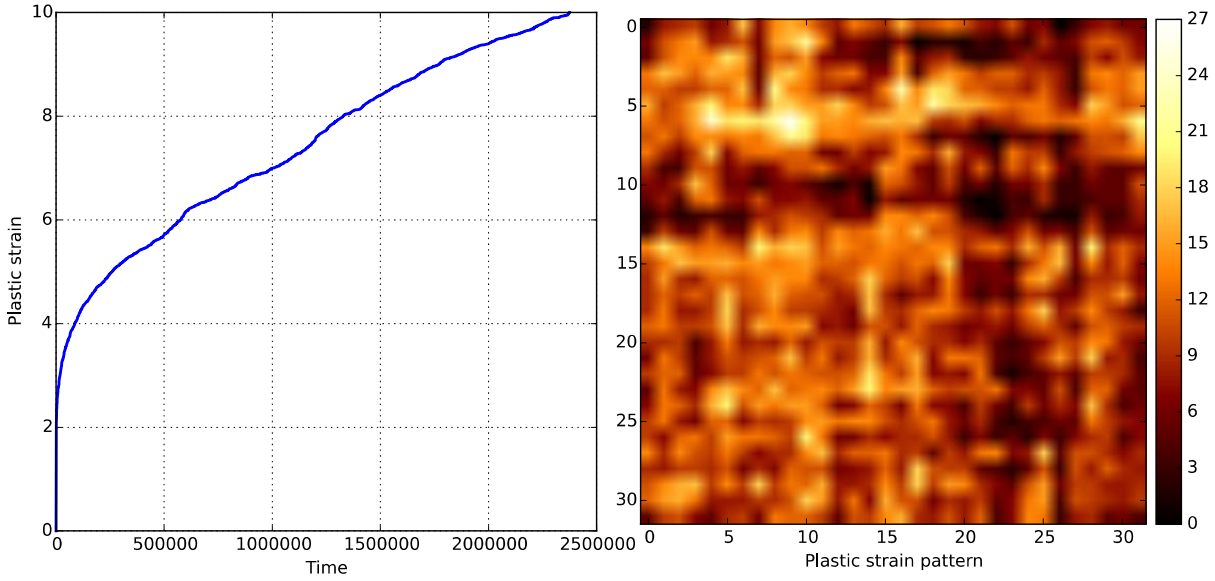
You can use this function for plotting nicely. You can find in the python files in StudOn

```
def plot_KMC(time,strain,strain_pattern):

    fig, axs = plt.subplots(nrows=1, ncols=2,figsize=(12, 6), facecolor='white')
    plt.tight_layout(pad=2., w_pad=0.05)

    axs[0].plot(time,strain,'-', linewidth=2)
    axs[0].grid(True)
    axs[0].set_xlabel('Time')
    axs[0].set_ylabel('Plastic strain')

    im = axs[1].imshow(strain_pattern,cmap='afmhot')
    axs[1].set_xlabel('Plastic strain pattern')
    divider = make_axes_locatable(axs[1])
    cbar_ax1 = divider.append_axes("right", size="5%", pad=0.1)
    fig.colorbar(im, cax=cbar_ax1)

    plt.show()
```

To use it:

```
plot_KMC(time_data,av_plastic_strain_data,plastic_strain.reshape(L,L))
```

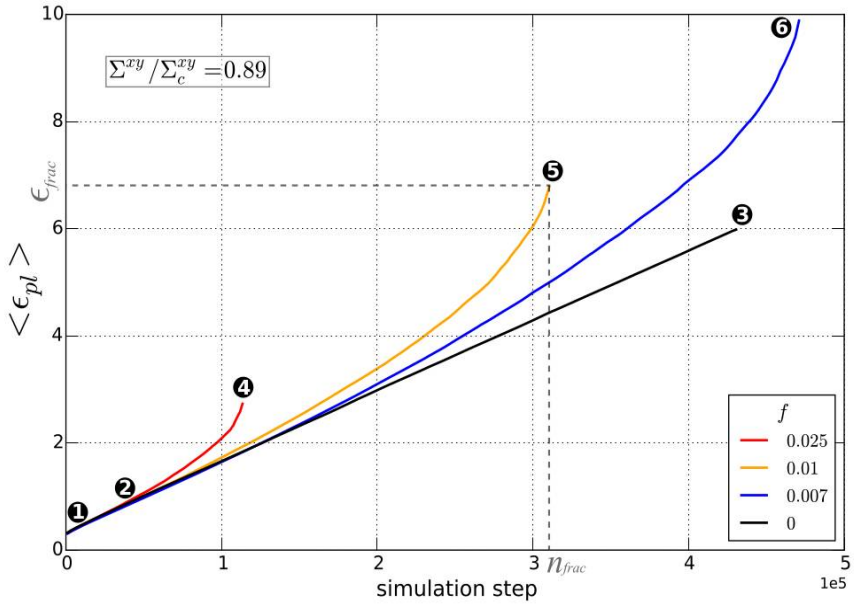
You should get something like this:

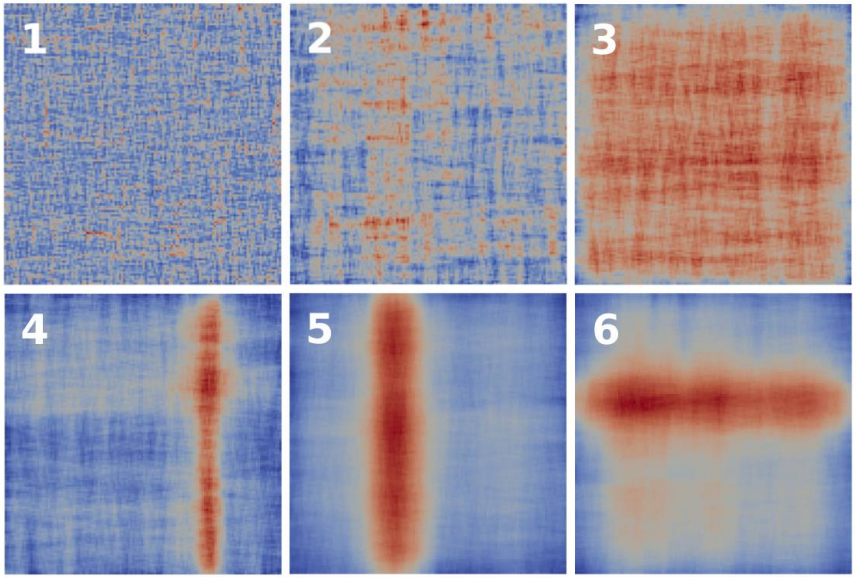


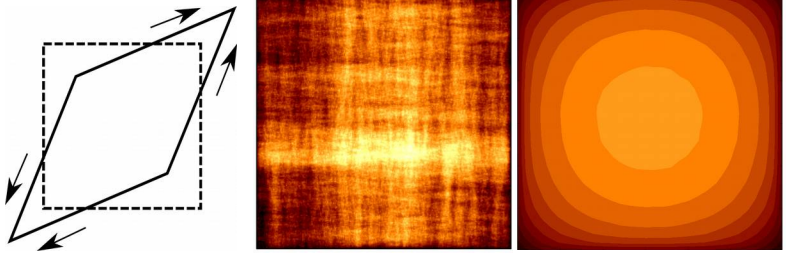
Now, if this is working, we can add **strain softening**:

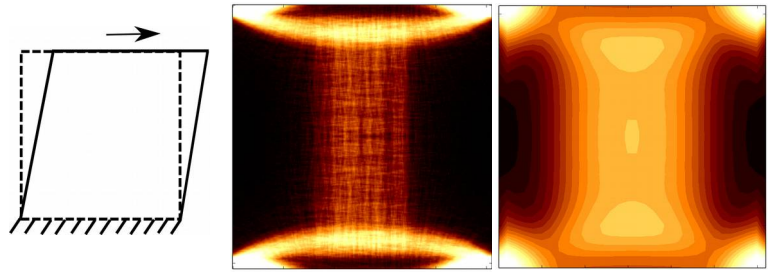
```
energy_barrier[element] = np.random.uniform(0.,1.0)
```

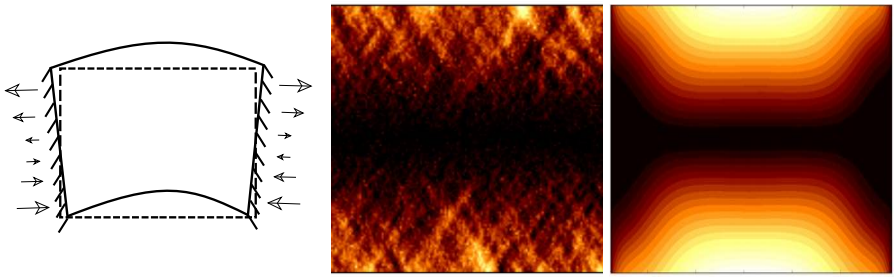
```
energy_barrier[element] = np.random.uniform(0.,1.0)*np.exp( -plastic_strain[element]/5. )
```

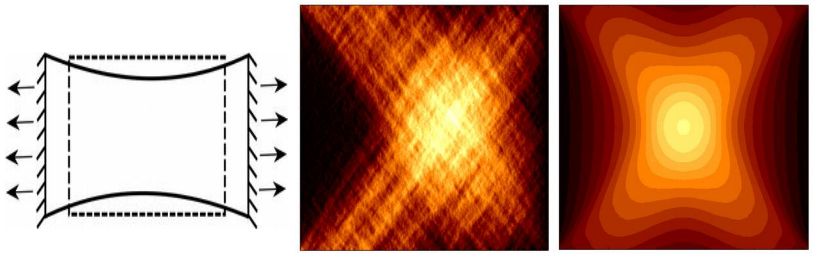

$\Sigma^{xy}/\Sigma_c^{xy} = 0.89$
ϵ_{frac}
$<\epsilon_{pl}>$
n_{frac}
simulation step
1e5
f
0.025
0.01
0.007
0
1
2
3
4
5
6

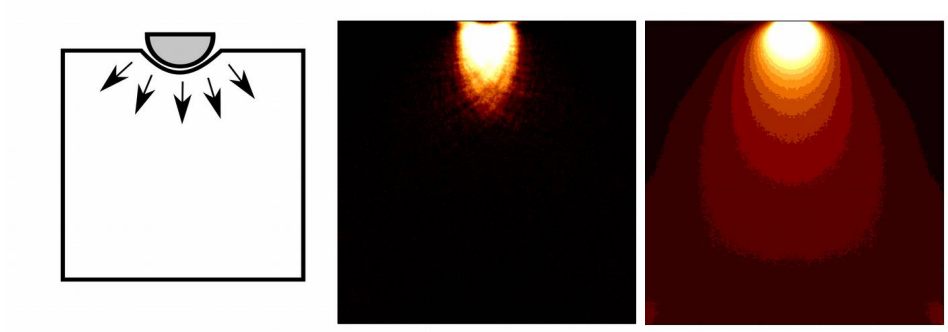

1
2
3
4
5
6

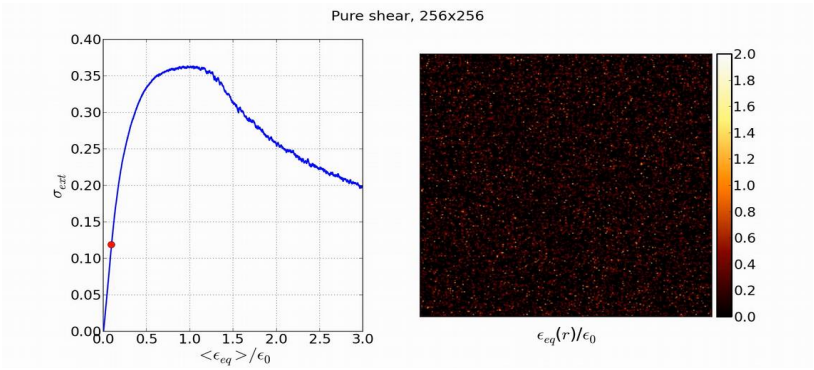


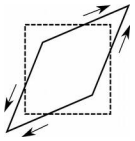


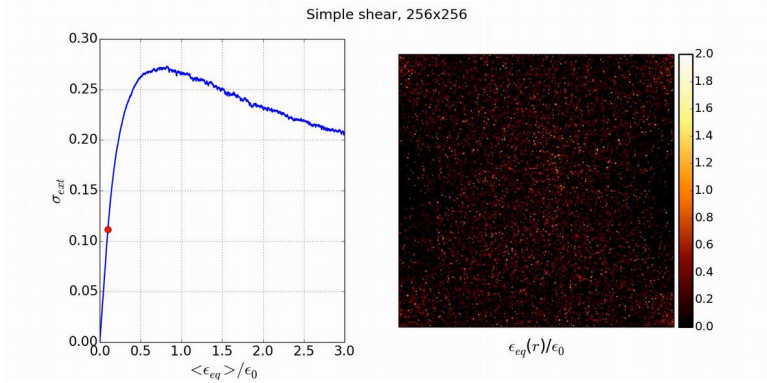


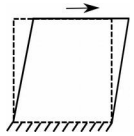


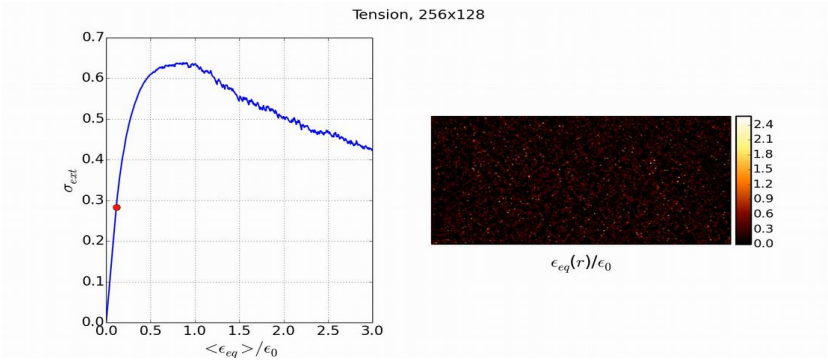


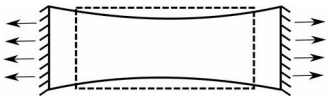

Pure shear, 256x256
σ_{ext}
$<\epsilon_{eq}>/\epsilon_0$
$\epsilon_{eq}(r)/\epsilon_0$

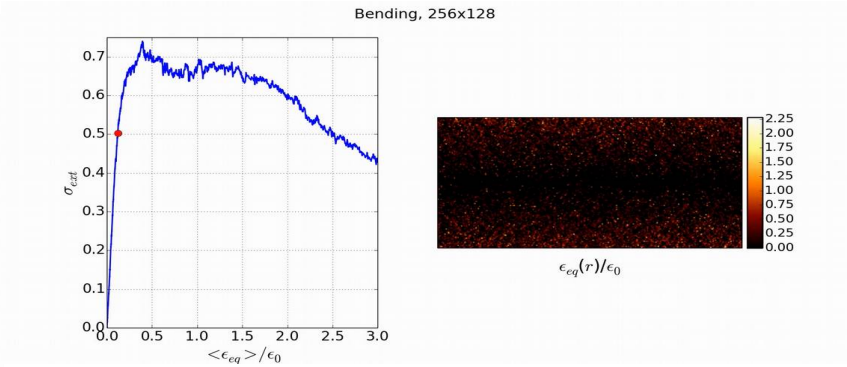


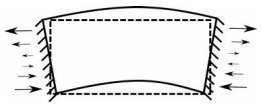

Simple shear, 256x256
σ_{ext}
$<\epsilon_{eq}>/\epsilon_0$
$\epsilon_{eq}(r)/\epsilon_0$

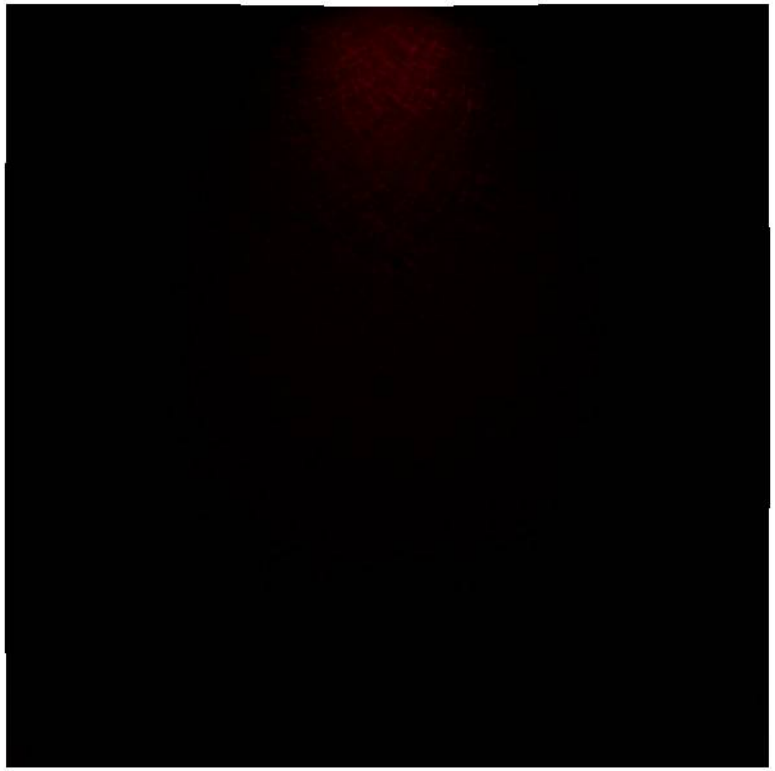



Tension, 256x128
σ_{ext}
$<\epsilon_{eq}>/\epsilon_0$
$\epsilon_{eq}(r)/\epsilon_0$

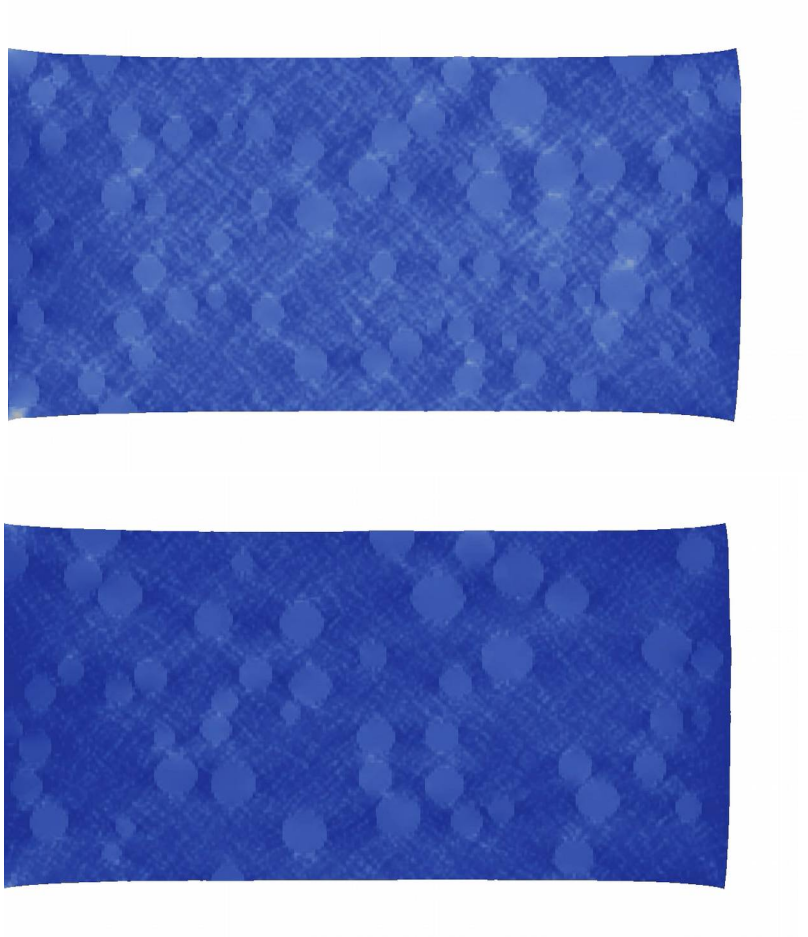


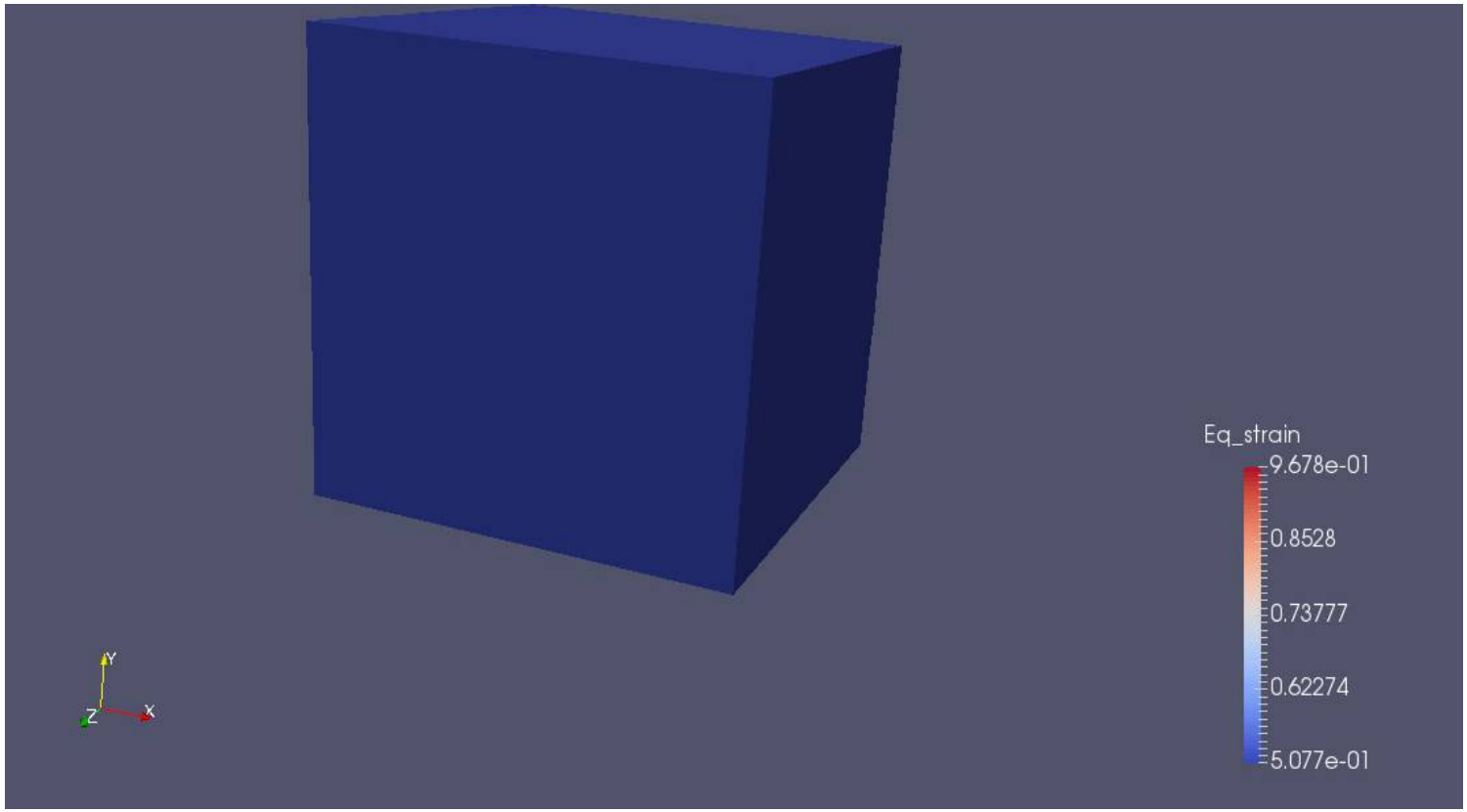

Bending, 256x128
σ_{ext}
$<\epsilon_{eq}>/\epsilon_0$
$\epsilon_{eq}(r)/\epsilon_0$





Study of the effects of nanocrystal inclusions on the strain localization and failure




Eq_strain
9.678e-01
0.8528
0.73777
0.62274
5.077e-01
Y
Z
X