surface_deformation

December 6, 2023

[ ]:
```
%load_ext autoreload
%autoreload 2

import matplotlib.pyplot as plt
import numpy as np
import networkx as nx

from mpl_toolkits.mplot3d import Axes3D

import matplotlib
matplotlib.rc('figure', figsize=(10, 5))
plt.style.use('ggplot')
```

0.1 Calcul de déformations : modèle de Moggi

$$\mathbf{u} = \Delta p(1-\nu)\frac{r_s^3}{G}\frac{\mathbf{x}-\mathbf{x}_s}{||\mathbf{x}-\mathbf{x}_s||^3}$$

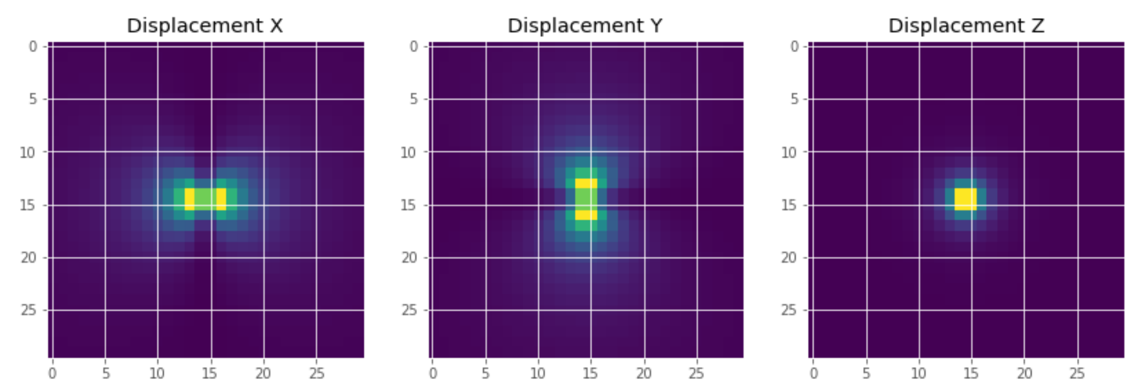
$\mathbf{u}$ le déplacement observé en $\mathbf{x}$ pour une source sphérique de centre $\mathbf{x}_s$ et de rayon r_s affichant une surpression Δp. ν le coefficient de Poisson et G le module de cisaillement du milieu encaissant.

[ ]:
```
# Définition des constantes du milieu
G = 1. # module de cisaillement
nu = 0.25 # coefficient de Poisson

# Domaine d'étude
L = 10.
H = 1.
N = 30 # discrétisation
X = np.linspace(0, L, N)
Y = np.linspace(0, L, N)
XX,YY = np.meshgrid(X,Y)

# Définition de la chambre
a = 0.01 # rayon
xs,ys,zs = [L/2,L/2,H/2] # position
dp = 1. # surpression
```

```
# Calcul de la déformation en surface
s = dp*(1-nu)*a**3/G

iR = 1/((XX-xs)**2+(YY-ys)**2+zs**2)**(3/2)

ux = s*np.abs((XX-xs))*iR
uy = s*np.abs((YY-ys))*iR
uz = s*np.abs((-zs))*iR
```

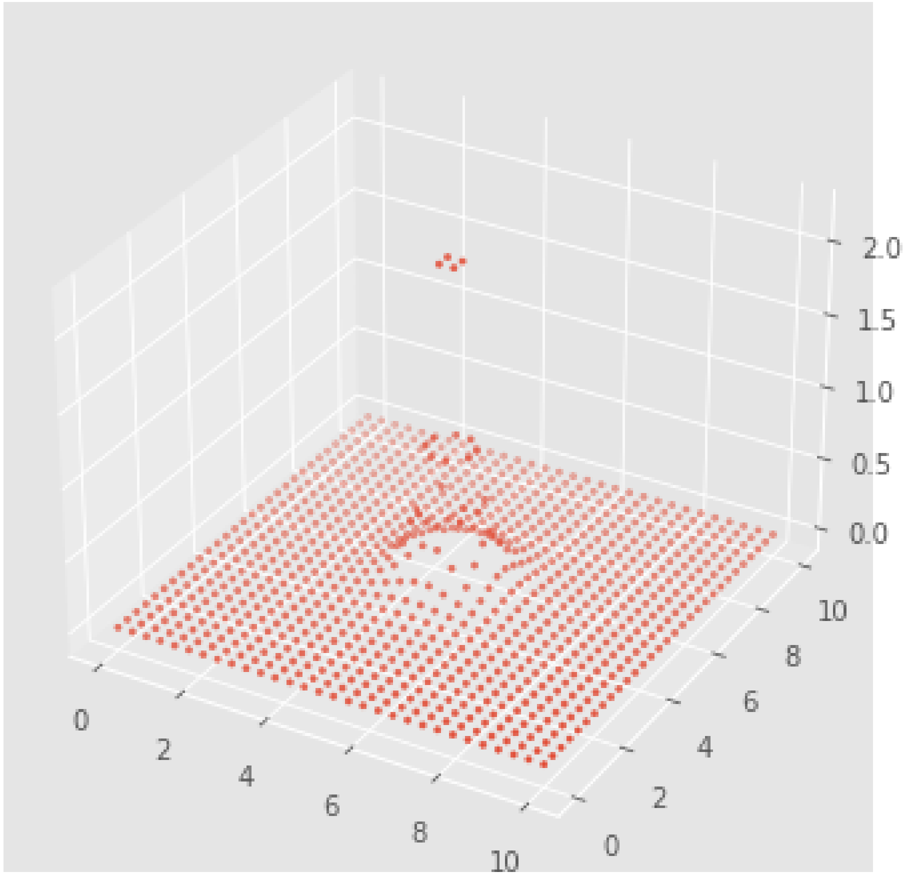
[ ]:

```
# Plot de la déformation
fig,axes = plt.subplots(1, 3, figsize=(14,6))
im = axes[0].imshow(ux)
axes[0].set_title("Displacement X")
im = axes[1].imshow(uy)
axes[1].set_title("Displacement Y")
im = axes[2].imshow(uz)
axes[2].set_title("Displacement Z")

plt.show()
```



[ ]:

```
# Réprésentation dans un maillage en 3D
fig = plt.figure(figsize=(6, 6))

ax = fig.add_subplot(projection='3d')
ax.scatter(XX+ux, YY+uy, H+uz, marker=".")

plt.show()
```



0.2 Tirage aléatoire de chambres dans une zone magmatique cylindrique

[ ]:
```
# Tirage aléatoire des chambres

# On tire des chambres sphériques contenues dans un cylindre de hauteur H de␣
↪profondeur Z et de rayon R
H_magma_body = 1e3
Z_magma_body = 5e3
R_magma_body = 1e3

# Rayon des chambres
# # Hyp 1 : tq suivant une loi exponentielle d'esperance R
# R_mean_magma_chamber = 50.
# Hyp 2 : tq suivant une loi normale d'esperance R et d'écart type SIGMA
R_mean_magma_chamber = 100.
SIGMA_magma_chamber = 50.
```

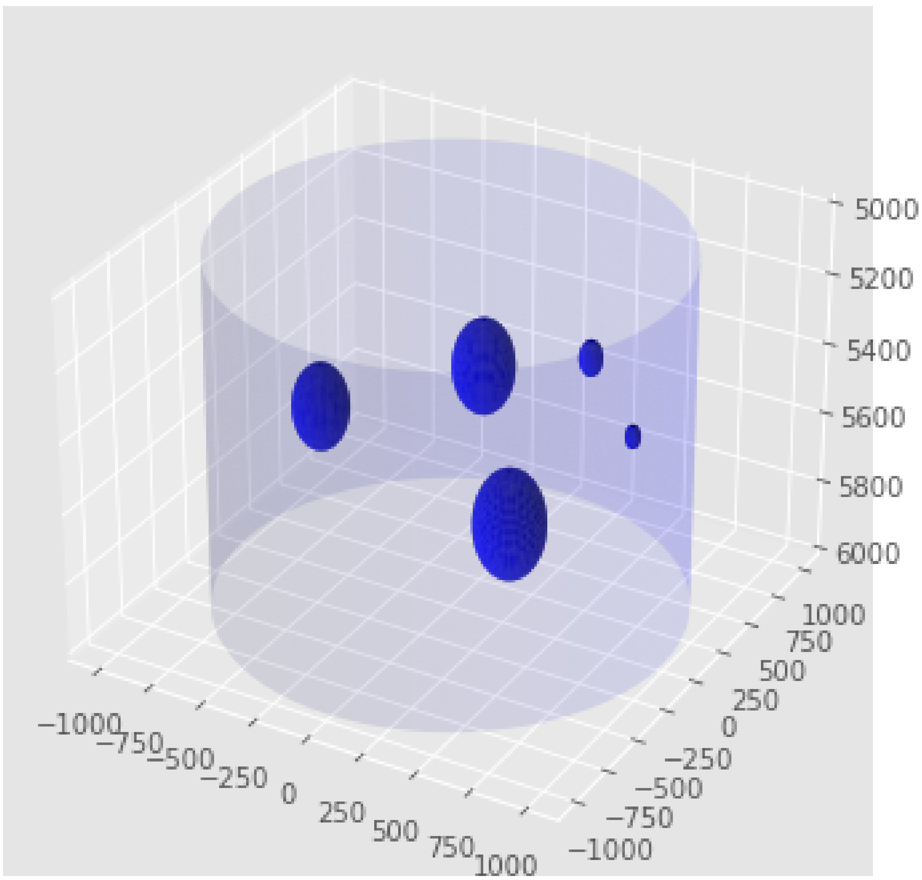
```python
# On tire des chambres jusqu'à que le volume de chambre soit X fois le volume du cylindre
X_fraction_volume = 0.01
volume_cible = X_fraction_volume*np.pi*R_magma_body**2*H_magma_body

class MagmaChamber :
    def __init__(self,x,y,z,r):
        self.x = x
        self.y = y
        self.z = z
        self.radius = r
    def __str__(self):
        return f"{self.radius:.1f}km at ({self.x:.1f},{self.y:.1f},{self.z:.1f})"
    def distance(self, mc_b):
        return np.sqrt((self.x-mc_b.x)**2+(self.y-mc_b.y)**2+(self.z-mc_b.z)**2)

def check_for_collision(mc1,mc2,mc3):
    """Check if edge (mc1,mc2) is crossing mc3"""

    # Paramètres de la ligne mc1-mc2
    line_vector = np.array([mc2.x - mc1.x, mc2.y - mc1.y, mc2.z - mc1.z])
    line_point = np.array([mc1.x, mc1.y, mc1.z])

    # Paramètres de la sphère
    sphere_center = np.array([mc3.x, mc3.y, mc3.z])

    # Calcul du vecteur entre le centre de la sphère et le point sur la ligne le plus proche
    nearest_point = line_point + np.dot(line_vector, sphere_center - line_point) / np.dot(line_vector, line_vector) * line_vector
    distance = np.linalg.norm(nearest_point - sphere_center)

    # Vérification de l'intersection
    if distance <= mc3.radius:
        return True
    else:
        return False


magma_chambers = []
total_mc_vol = 0.
while total_mc_vol < volume_cible :

    # tirage du centre de la chambre
    z = np.random.uniform(Z_magma_body, Z_magma_body+H_magma_body)
    r_pos = np.random.uniform(0., R_magma_body)
```

```
    theta = np.random.uniform(0, 2*np.pi)

    x = r_pos*np.cos(theta)
    y = r_pos*np.sin(theta)

    # tirage du rayon de la chambre
    # r = np.random.exponential(scale=R_mean_magma_chamber)
    r = np.random.normal(R_mean_magma_chamber, SIGMA_magma_chamber)

    # check that within the cylinder
    if r+r_pos > R_magma_body:
        continue

    if ((z-Z_magma_body < r) or (Z_magma_body+H_magma_body-z < r)):
        continue

    # check that does not intersect with other chambers
    for chamber in magma_chambers:
        if np.sqrt((chamber.x-x)**2+(chamber.y-y)**2+(chamber.z-z)**2) <␣
↪chamber.radius + r:
            continue

    # checks passed
    magma_chambers.append(MagmaChamber(x,y,z,r))
    total_mc_vol += 4/3*np.pi*r**3

print(len(magma_chambers))
```

5

[ ]:
```
def plot_sphere(ax, center, radius, text):
    phi, theta = np.mgrid[0.0:2.0*np.pi:100j, 0.0:np.pi:50j]
    x = center[0] + radius * np.sin(theta) * np.cos(phi)
    y = center[1] + radius * np.sin(theta) * np.sin(phi)
    z = center[2] + radius * np.cos(theta)
    ax.plot_surface(x, y, z, color='b',  shade=True, alpha=0.7)
    # ax.text(x, y, z, text, color='red', fontsize=8, ha='center', va='center')

def plot_cylinder(ax, center, radius, height):
    phi = np.linspace(0, 2*np.pi, 100)
    z = np.linspace(0, height, 50)
    Z, Phi = np.meshgrid(z, phi)
    X = center[0] + radius * np.cos(Phi)
    Y = center[1] + radius * np.sin(Phi)
    ax.plot_surface(X, Y, Z + center[2], color='b', alpha=0.1)
```

```
fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(111, projection='3d')

for i,mc in enumerate(magma_chambers):
    plot_sphere(ax, [mc.x, mc.y,mc.z], mc.radius, i+1)

plot_cylinder(ax, [0,0,Z_magma_body], R_magma_body, H_magma_body)

# ax.set_box_aspect([np.ptp(coord) for coord in zip(*[s[0] for s in spheres])])

ax.invert_zaxis()
plt.show()
```



[ ]:
```
# Création d'un graph

mc_graph = nx.Graph()

# 0. ajout de la chambre source
mc_graph.add_node(0, radius=-1, center_coord=np.array([-1,-1,-1]))
```

```
# 1. ajout des noeuds
for i,mc in enumerate(magma_chambers):
    mc_graph.add_node(i+1, radius=mc.radius, center_coord=np.array([mc.x,mc.
 ↪y,mc.z]))

# 2. ajout des arrêtes
for i,mc1 in enumerate(magma_chambers):
    for j,mc2 in enumerate(magma_chambers):
        if i == j: continue
        valid = True
        for k,mc3 in enumerate(magma_chambers):
            if (i==k) or (j==k): continue
            if check_for_collision(mc1,mc2,mc3):
                valid = False
                break
        if valid : mc_graph.add_edge(i+1, j+1, distance=mc1.distance(mc2))
```

[ ]:
```
def draw_graph(G, node_size_comp, edge_width_comp, node_size_factor=400,␣
 ↪edge_width_factor=4):

    seed = 13648
    pos = nx.spring_layout(G, seed=seed)

    node_sizes = [G.nodes[node][node_size_comp]*node_size_factor for node in G.
 ↪nodes]
    node_sizes[0] = 5*node_size_factor
    edge_sizes = [G.edges[edge][edge_width_comp]*edge_width_factor for edge in␣
 ↪G.edges]

    fig,ax = plt.subplots(figsize=(8,8))

    nx.draw_networkx_nodes(G, pos, node_size=node_sizes, node_color="indigo")
    nx.draw_networkx_edges(
        G,
        pos,
        node_size=node_sizes,
        width=edge_sizes,
    )

    nx.draw_networkx_labels(G, pos, font_color="w")

    ax.set_axis_off()
    ax.set_title(f"Node size = {node_size_comp} ; egde width =␣
 ↪{edge_width_comp}")
    fig.patch.set_facecolor('white')
    return fig,ax
```

```
# fig,ax = draw_graph(mc_graph, "radius", "distance", node_size_factor=50,␣
↪edge_width_factor=1e-3)
# plt.show()
```

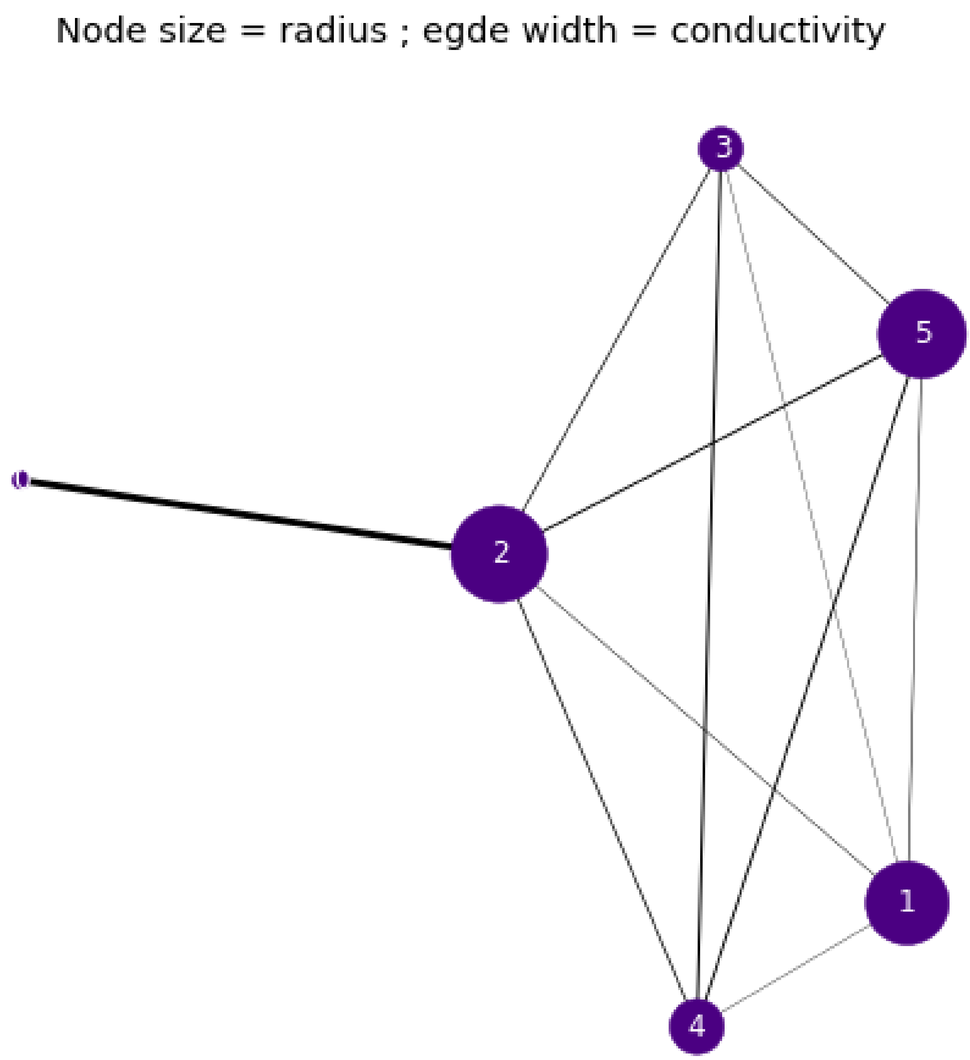
[ ]:
```
# Calcul des compressibilités (absolues : V*compressibilité) et des␣
↪conductivitées

G = 10e9 # 10GPa : granite
R_conduct = 1. # rayon des conduits = 50m
mu_magma = 100 # 100 Pa.s : magma basaltique 1500K

for node in mc_graph.nodes:

    if mc_graph.nodes[node]["radius"] == -1 : continue # source

    mc_graph.nodes[node]["compressibility"] = 4/3*np.pi*mc_graph.
↪nodes[node]["radius"]**3 * 3/(4*G)

    for node2 in mc_graph.nodes:
        if mc_graph.nodes[node2]["radius"] == -1 : continue
        try :
            mc_graph[node][node2]["conductivity"] = np.pi*R_conduct**4/
↪(8*mu_magma*mc_graph[node][node2]["distance"])
        except:
            pass # nodes not connected

# Connexion de la chambre la plus profonde à la source
depths = np.array([center[2] for center in nx.get_node_attributes(mc_graph,␣
↪"center_coord").values()])
radius = np.array(list(nx.get_node_attributes(mc_graph, "radius").values()))
depths_max = depths + radius
i_max_depth = depths_max.argmax()

distance_to_source = Z_magma_body+H_magma_body-depths_max.max()
mc_graph.add_edge(i_max_depth, 0, conductivity=np.pi*R_conduct**4/
↪(8*mu_magma*distance_to_source))
```

[ ]:
```
# Répartition des distances
distances = list(nx.get_edge_attributes(mc_graph, "distance").values())
conductivity = np.array(list(nx.get_edge_attributes(mc_graph, "conductivity").
↪values()))
compressibility = np.array(list(nx.get_node_attributes(mc_graph,␣
↪"compressibility").values()))

print(f"Mean conductivity = {conductivity.mean():.2e}")
```

```
print(f"Mean compressibility = {compressibility.mean():.2e}")
print(f"Characteritic time = {compressibility.mean()/conductivity.mean():.2e}")
```

```
Mean conductivity = 8.30e-06
Mean compressibility = 4.81e-04
Characteritic time = 5.79e+01
```

[ ]:
```
fig,ax = draw_graph(mc_graph, "radius", "conductivity", node_size_factor=10,␣
 ↪edge_width_factor=1e5)
plt.show()
```

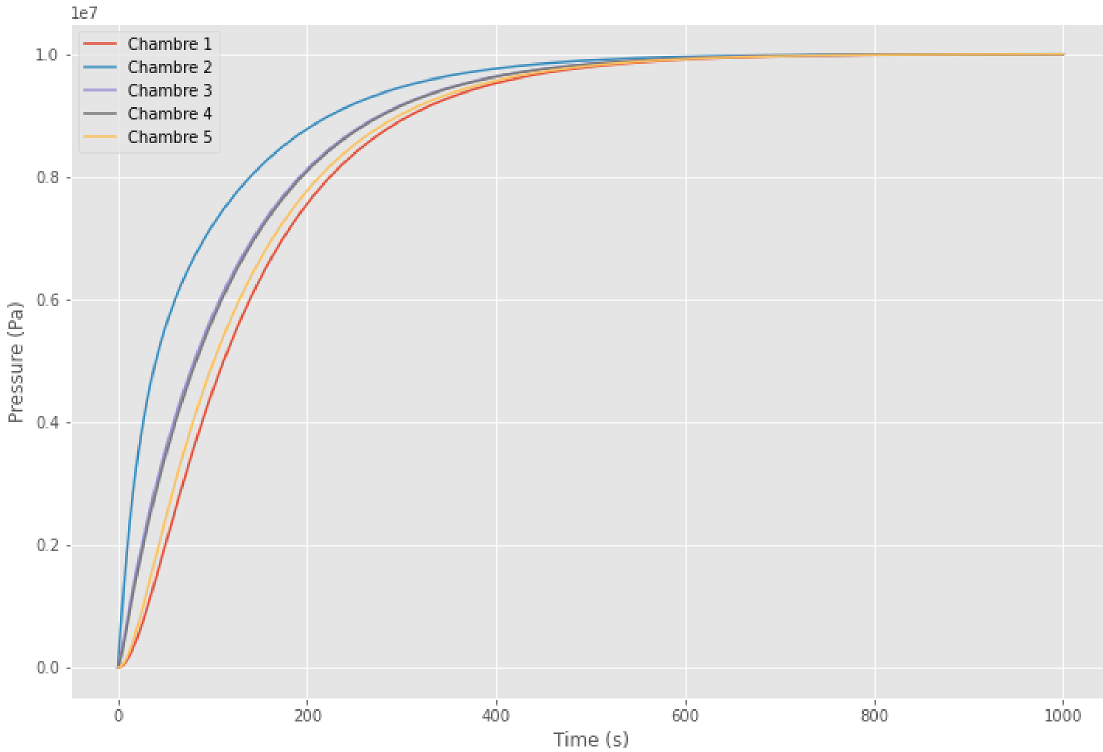


1 Calcul de la réaction du système à une surpression

[ ]:
```
from pressure_timeseries import compute_pressure_time_serie

DELTA_P = 1e7 # 10 MPa
def source(t, t_s=0.): return DELTA_P if t >= t_s else 0

tmax = 1e3 # 1 an
p0 = np.zeros(len(mc_graph)-1)

p = compute_pressure_time_serie(mc_graph, source, tmax, p0)
```

[ ]:
```
# Tracé des pressions dans les chambres au cours du temps
t_space = np.linspace(0, tmax, p.shape[0])
# plt.plot(t_space, [source(t_) for t_ in t_space], label="Source", color="k",␣
 ↪ls=":")

fig,ax = plt.subplots(figsize=(12,8))

for i in range(len(mc_graph)-1):
    ax.plot(t_space, p[:,i], label=f"Chambre {i+1}")

ax.set_xlabel("Time (s)")
ax.set_ylabel("Pressure (Pa)")
plt.legend()
plt.plot()
```

[ ]: []

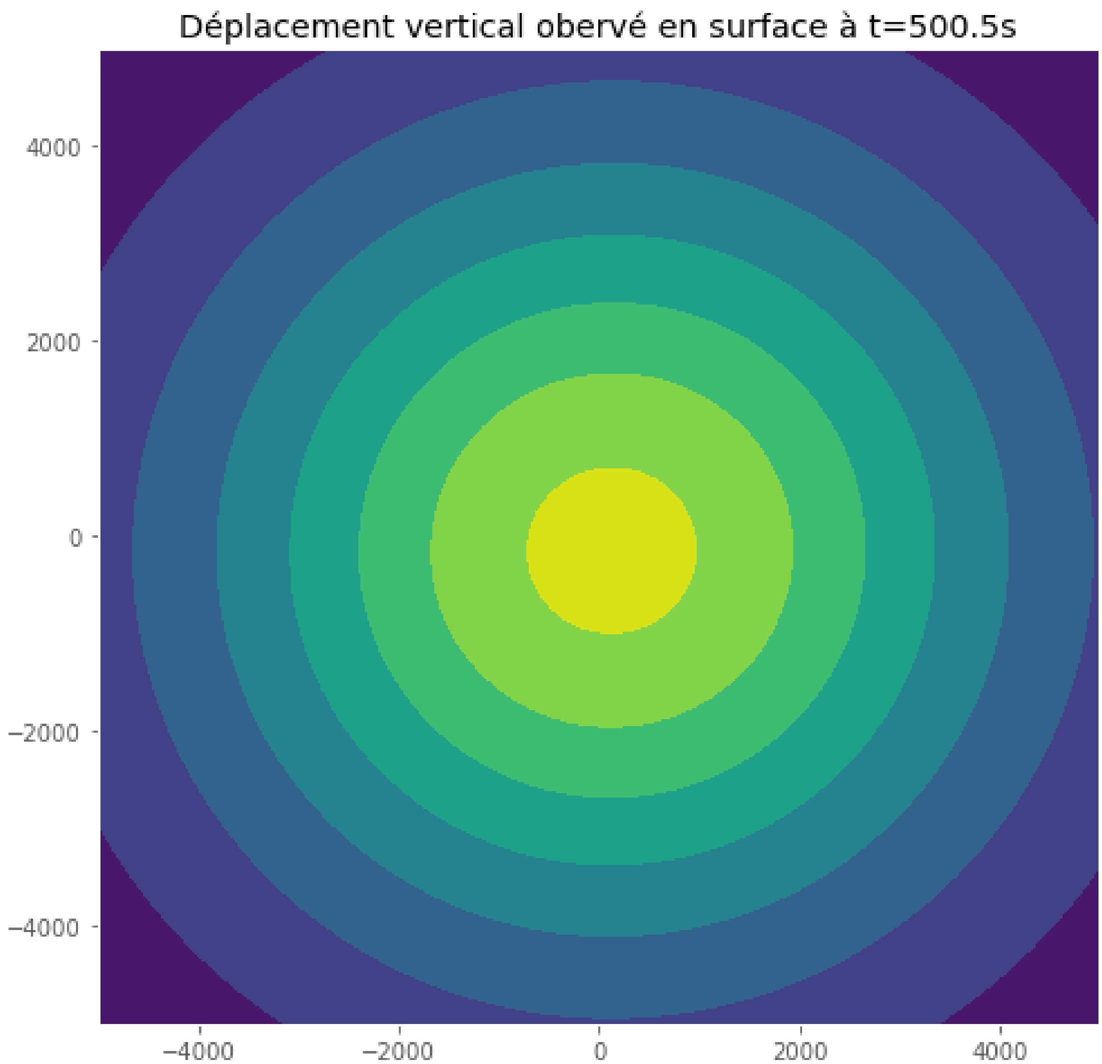


2 Calcul des déformations en surface causées par les surpressions dans les chambres

[ ]:
```
# Calcul des déformations causées par un ensemble de chambres
# On calcul la déformation engendrée en surface pour chaque chambre et on somme

L = 10e3 # 10km sur 10km
N_space = 100
X = np.linspace(-L/2, L/2, N_space)
Y = np.linspace(-L/2, L/2, N_space)

XX,YY = np.meshgrid(X,Y)

# Calcul de la déformation en surface

it = 500

N_it,N_mc = p.shape

ux_mc,uy_mc,uz_mc = np.zeros((N_mc,N_space,N_space,N_it)),np.
 ↪zeros((N_mc,N_space,N_space,N_it)),np.zeros((N_mc,N_space,N_space,N_it))
```

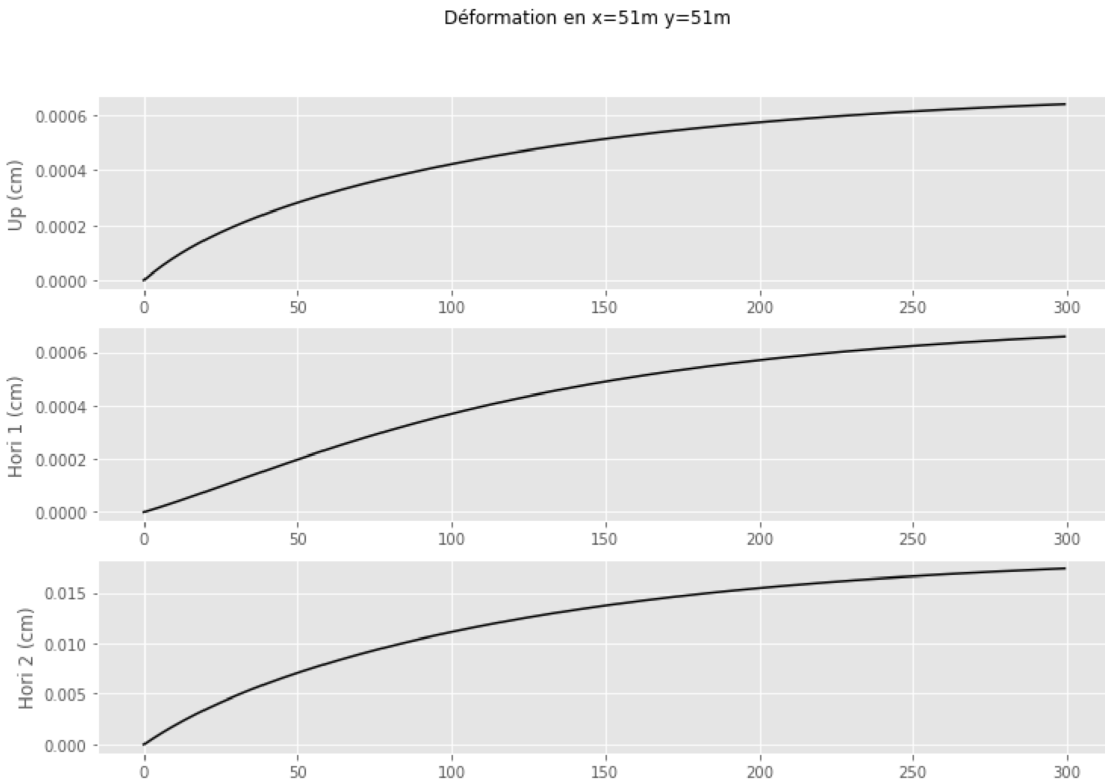
```python
for i_mc,mc in enumerate(magma_chambers):

    for it in range(N_it):

        s = p[it,i_mc]*(1-nu)*mc.radius**3/G

        iR = 1/((XX-mc.x)**2+(YY-mc.y)**2+mc.z**2)**(3/2)

        ux_mc[i_mc,:,:,it] = s*np.abs((XX-mc.x))*iR
        uy_mc[i_mc,:,:,it] = s*np.abs((YY-mc.y))*iR
        uz_mc[i_mc,:,:,it] = s*np.abs((-mc.z))*iR

# Somme des déplacements engendrés par les différentes chambres

ux,uy,uz = ux_mc.sum(axis=0),uy_mc.sum(axis=0),uz_mc.sum(axis=0)
```

[ ]:
```python
# Plot snapshot déplacement
it = 500
fig,ax = plt.subplots(figsize=(8,8))
plt.title(f"Déplacement vertical obervé en surface à t={t_space[it]:.1f}s")
plt.contourf(XX,YY,uz[:,:,it])
# plt.imshow(uz[:,:,it])
plt.show()
```



[ ]:
```
# Plot d'une timeserie de déplacement

x,y = N_space//2,N_space//2

Nt_max = 300

fig, ax = plt.subplots(3, 1, figsize=(12,8)) #figsize=plt.figaspect(1), dpi=200.
 ↪)

for i,(cpnt,name) in enumerate(zip([ux,uy,uz],["Up (cm)","Hori 1 (cm)","Hori 2␣
 ↪(cm)"])):
    ax[i].plot(t_space[:Nt_max], cpnt[x,y,:Nt_max]*100 ,c="k")
    ax[i].set_ylabel(name)
```

```
fig.suptitle(f"Déformation en x={X[x]:.0f}m y={Y[y]:.0f}m")
plt.show()
```



Conclusion : impossible de faire la différence depuis les déplacements en surface entre une chambre unique et un ensemble de chambres

2.1 Système de seuils d'activation sur les conductivités

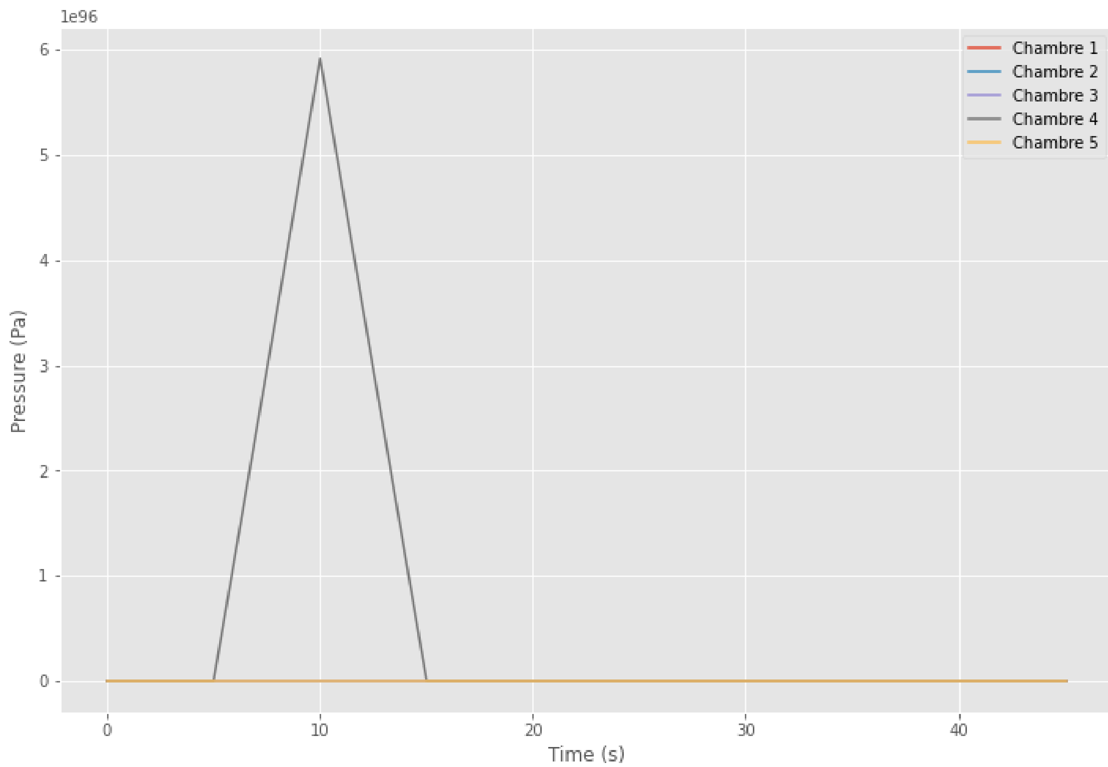
[ ]:
```
# Système de seuil d'activation des conductivités

from pressure_timeseries import compute_pressure_time_serie_p_dep

DELTA_P = 1e7 # 10 MPa
def source(t, t_s=0.): return DELTA_P if t >= t_s else 0

tmax = 5e3 # 1 an
p0 = np.zeros(len(mc_graph)-1)

p2 = compute_pressure_time_serie_p_dep(mc_graph, source, tmax, p0, p_ts=1e5)
```

```
/home/sbrisson/anaconda3/lib/python3.9/site-
packages/scipy/integrate/_odepack_py.py:248: ODEintWarning: Excess work done on
```

```
this call (perhaps wrong Dfun type). Run with full_output = 1 to get
quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
```

[ ]:
```python
t_space = np.linspace(0, tmax, p.shape[0])
# plt.plot(t_space, [source(t_) for t_ in t_space], label="Source", color="k",␣
↪ls=":")

N_tmax = 10

fig,ax = plt.subplots(figsize=(12,8))

for i in range(len(mc_graph)-1):
    ax.plot(t_space[:N_tmax], p2[:N_tmax,i], label=f"Chambre {i+1}")

ax.set_xlabel("Time (s)")
ax.set_ylabel("Pressure (Pa)")
plt.legend()
plt.plot()
```

[ ]: []



[ ]:
```python
p2[:10,:]
```

[ ]:
```
array([[ 0.00000000e+000,  0.00000000e+000,  0.00000000e+000,
         0.00000000e+000,  0.00000000e+000],
       [ 1.20191601e+004,  7.15065047e+005,  2.47471035e+005,
         1.47471034e+005,  1.72763473e+004],
       [ 0.00000000e+000,  0.00000000e+000,  6.90033534e-310,
         5.91524946e+096,  6.90034219e-310],
       [-1.00652733e-111,  0.00000000e+000,  0.00000000e+000,
         0.00000000e+000,  0.00000000e+000],
       [ 0.00000000e+000,  0.00000000e+000,  0.00000000e+000,
         0.00000000e+000,  0.00000000e+000],
       [ 0.00000000e+000,  0.00000000e+000,  0.00000000e+000,
         6.90040115e-310, -3.37810118e-259],
       [ 6.90034025e-310,  2.97835558e-191,  0.00000000e+000,
         0.00000000e+000,  0.00000000e+000],
       [ 0.00000000e+000,  6.90040190e-310,  2.37517129e-288,
         0.00000000e+000,  0.00000000e+000],
       [ 6.90040001e-310,  1.75612040e-222,  0.00000000e+000,
         0.00000000e+000,  6.90037290e-310],
       [ 4.76999345e-146,  6.90034226e-310, -4.02082306e-228,
         0.00000000e+000,  0.00000000e+000]])
```

[ ]: