```
from creeepy.parameters import *
from creeepy.law import *


f = Fixed('config')
s = Stochastic('config')

f.GenerateVariable()
s.GenerateSeed()
s.CharacteriseSeed()

creep = Creep(T = 1473, A = 1.5e9, n = 1, p = 3, Q = 375e3, R = 8.3145, r = 0, fH2O = 1, alpha = 40, phi = 0.001)
creep.SetCalibration('Hirth_2003', 'diffusion', 'dry')
creep.SetParams(f, s)
creep.SetVariables(f.s, s.d, 'mean', 'd')
creep.d = creep.SR['mean']

creep.Hirth2003()
creep.SaveSR('meanStrain')             # creep for the mean grain size at each step of stress
creep.StochasticCreep(s.d, s.nbDraw)  # mean creep of the grain sizes for each step of stress
creep.d = s.stoMean
creep.Hirth2003()
creep.SaveSR('theoStrain')

def Gouriet_2018(stress, val, T, A = 1.699e16, n = 3, mu = 80e9, Q = 460e3, R = 8.314, p = 1.5, q = 2, peierls = 2e9):

    strainRate =  A * ( stress*1e6/mu )**val * np.exp((-Q/(R*T)) * ( 1 - ( stress*1e6/peierls )**p )**q)
    return strainRate

comp = creep.SR.copy()

comp['Disl'] = Gouriet_2018(comp.s, 3, 1473)

for j in range(1, len(comp.index)):
    comp.loc[j, 'esperanceMeanStrain'] = sum(comp.loc[0:j, 'meanStrain'])/(j)
    comp.loc[j, 'esperanceStoStrain'] = sum(comp.loc[0:j, 'stoStrain'])/(j)

import matplotlib.pyplot as plt

_ = plt.hist(s.tab, bins=100)

plt.title("Histogram with 'auto' bins")
plt.savefig('distribGrainSize.png')



import matplotlib.pyplot as plt

plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(10, 8), dpi=300)


plt.xlabel(rf'Log(Strain rate ($s^{-1}$)), T = {creep.T}K')
plt.ylabel('Log(Stress (MPa))' )
plt.xscale('log')
plt.yscale('log')

plt.plot(comp.Disl, comp.s, color = 'black', label = f'dislocation creep, Gouriet et al 2018 (n = 3)')
plt.plot(comp.theoStrain, comp.s, color = 'red', label = rf'diffusion creep, Hirth and Kohltsedt 2003, (d = {s.stoMean}$\mu m$)')

plt.legend()
plt.savefig('2creep.png')

plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(10, 8), dpi=300)

plt.plot(comp.theoStrain, comp.s, color = 'red', label = rf'd = {s.stoMean}$\mu m$')
plt.scatter(comp.stoStrain, comp.s, color = 'red', marker = 'o', alpha = 0.05)
plt.scatter(comp.meanStrain, comp.s, color = 'red', marker = '+', alpha = 0.05)
plt.plot(comp.esperanceStoStrain, comp.s, color = 'red', ls = '--', label = 'esperance of the mean o
```

```
f strains for each d, for each stress step')
plt.plot(comp.esperanceMeanStrain, comp.s, color = 'red', ls = ':', label = 'esperance of the mean d for each stress step')
plt.xlabel(rf'Log(Strain rate ($s^{-1}$)), T = {creep.T}K')
plt.ylabel('Log(Stress (MPa))' )
plt.legend()
plt.xscale('log')
plt.yscale('log')
plt.savefig('diffSto.png')


esp = pd.DataFrame()

stress = comp.s
T = creep.T

esp['totalCreep'] = comp.Disl + comp.theoStrain
esp['totalCreepDiffSto'] = comp.Disl + comp.stoStrain
esp['percent'] = comp.Disl/(comp.Disl + comp.stoStrain)
esp['stress'] = stress


for j in range(1, len(esp.index)):
    esp.loc[j, 'esperanceTotalStoDiff'] = sum(esp.loc[0:j, 'totalCreepDiffSto'])/(j)

plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(10, 8), dpi=300)

plt.plot(esp.totalCreep, stress, color = 'gray', label = f'total creep, T = {T}')
plt.scatter(esp['totalCreepDiffSto'], stress, marker = '+', alpha = 0.1, color = 'gray')
plt.plot(esp.esperanceTotalStoDiff, stress, color = 'gray', ls = '--', label = 'esperance')

point = esp[esp['percent']>0.25]
plt.scatter(esp.loc[point.index[0], 'totalCreepDiffSto'], esp.loc[point.index[0], 'stress'], color = 'red', marker = '_', s = 1000, alpha = 0.25, label = f'dislocation creep 25%')

point = esp[esp['percent']>0.5]
plt.scatter(esp.loc[point.index[0], 'totalCreepDiffSto'], esp.loc[point.index[0], 'stress'], color = 'red', marker = '_', s = 1000, alpha = 0.5, label = f'dislocation creep 50%')

point = esp[esp['percent']>0.75]
plt.scatter(esp.loc[point.index[0], 'totalCreepDiffSto'], esp.loc[point.index[0], 'stress'], color = 'red', marker = '_', s = 1000, alpha = 0.75, label = f'dislocation creep 75%')

plt.xlabel('Log(Strain rate (s-1))')
plt.ylabel('Log(Stress (MPa))' )
plt.legend()
plt.xscale('log')
plt.yscale('log')
plt.savefig('esperanceCreep.png')


plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(10, 8), dpi=300)
plt.scatter(esp.percent, esp.totalCreep, c = esp.stress, cmap = 'bone')
plt.yscale('log')
cb = plt.colorbar()
cb.set_label(label='stress (MPa)')
plt.ylabel(r'strain rate ($s^{-1}$)')
plt.xlabel(f'percentage of dislocation creep on total strain, T = {T}K')
plt.savefig('esperanceCreeppercent.png')
```

```python
#!/usr/bin/env python
# coding: utf-8

# In[2]:


from creeepy.parameters import *
from creeepy.creep import *
from creeepy.output import *
from creeepy.plot import *


fix = Fixed('config')
sto = Stochastic('config')

fix.GenerateVariable()
sto.GenerateSeed()

fix.SetVariable('s', fix.name)


plt.rcParams["font.family"] = "serif"
plt.rcParams.update({'font.size': 8})
plt.figure(figsize=(5, 3), dpi=300)

_ = plt.hist(sto.grainSize, bins='auto')

plt.text(160,11, f'mean = {sto.stoMean}\nmax = {sto.stoMax}\nmin = {sto.stoMin}\nstd = {sto.stoStd}\ndistribution = normal\nnbStep = {sto.nbStep}\nnbDraw = {sto.nbDraw}')

plt.xlabel(r'Grain size ($\mu m$')
plt.savefig('distribGrainSize.png')


# In[3]:


creep = Creep(T = 1473, phi = 0.001, d = 200)

creep.SetCalibration('Gouriet2018', 'dislocation', 'dry')
creep.SetParams(fix, sto)
creep.RegularCreep(creep.Gouriet2018)

creep.SetCalibration('Hirth2003', 'diffusion', 'dry', clear = ['T'])
creep.T = 1473
creep.d = 200
creep.SetParams(fix, sto)
creep.RegularCreep(creep.Hirth2003)

plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(10, 8), dpi=300)
plt.plot(creep.rS.r_Gouriet2018, creep.rS.s, label = 'dislocation creep, n = 3 (Gouriet et al 2018)', color = 'black')
plt.plot(creep.rS.r_Hirth2003, creep.rS.s, label = r'diffusion creep, d = 200 $\mu m$ (Hirth and Kohlstedt 2003)', color = 'red')
plt.legend()
plt.xlabel(r'Log(Strain rate ($s^{-1}$))')
plt.ylabel('Log(Stress (MPa))' )
plt.xscale('log')
plt.yscale('log')
plt.show()
plt.savefig('creep_grainSize.png')


# In[4]:


creep.SetCalibration('Hirth2003', 'diffusion', 'dry')
sto.SetVariable('d', sto.stoName, clear = True)
creep.SetParams(fix, sto)
creep.StochasticCreep(sto.grainSize, 'd', creep.Hirth2003)
a = ['r_Gouriet2018', 'r_Gouriet2018']
b = ['r_Hirth2003', f'sto_{creep.name}']
creep.TotalCreep(a, b)

```

```

# In[17]:


import random

mylist = []

for i in range(0,100):
    x = random.randint(1,sto.nbStep)
    mylist.append(x)


# In[18]:


plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(12, 10), dpi=300)

tab = creep.S_d_grainSize_1473

plt.plot(tab['t1-2'], tab['s'], color = 'grey', label = r'Total creep, n = 3, d = 200 $\mu m$')
plt.plot(tab['t1-3_esp'], tab['s'], color = 'grey', ls = '--', label = 'Total creep, n = 3, d = grai
nSize table')

for i in mylist:
    plt.scatter(tab.loc[i, 't1-3'], tab.loc[i, 's'], color = 'grey', alpha = 0.1)


plt.legend()
plt.xlabel(r'Log(Strain rate ($s^{-1}$))')
plt.ylabel('Log(Stress (MPa))' )
plt.xscale('log')
plt.yscale('log')
plt.show()
plt.savefig('creep_grainSize-esperance.png')


# In[6]:


T = 1473
plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(10, 8), dpi=300)
plt.scatter(tab['t1-3_percent'], tab['t1-3'], c = tab['s'], cmap = 'bone')
plt.yscale('log')
cb = plt.colorbar()
cb.set_label(label='stress (MPa)')
plt.ylabel(r'strain rate ($s^{-1}$)')
plt.xlabel(f'percentage of dislocation creep on total strain, T = {T}K')
plt.savefig('creep_grainSize-percent.png')


# In[9]:


sto.GenerateCompositeSeed()

plt.rcParams["font.family"] = "serif"
plt.rcParams.update({'font.size': 8})
plt.figure(figsize=(5, 3), dpi=300)

out = Output()
out.Describe('test', fixed = fix, stochastic = sto, creep = creep)
out.SaveAttributes(sto)
out.SaveAttributes(creep)

_ = plt.hist(sto.grainSizeComp, bins='auto')

plt.text(1000,400, f'nbStep = {sto.nbStep}\nnbDraw = {sto.nbDraw}')

plt.xlabel(r'Grain size ($\mu m$')
plt.savefig('distribGrainSizeComp.png')


```

```
creep.SetCalibration('Hirth2003', 'diffusion', 'dry')
sto.SetVariable('d', sto.stoNameComp, clear = True)
creep.SetParams(fix, sto)
creep.StochasticCreep(sto.grainSizeComp, 'd', creep.Hirth2003, prop = 2)
a = ['r_Gouriet2018', 'r_Gouriet2018', 'r_Gouriet2018', 'r_Gouriet2018']
b = ['r_Hirth2003', f'sto_{creep.name}', 'sto_Hirth2003_SubSeed1', 'sto_Hirth2003_SubSeed2']
creep.TotalCreep(a, b, prop = 2)


# In[16]:


plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(12, 10), dpi=300)

tab = creep.S_d_grainSizeComp_1473

plt.plot(tab['t1-2'], tab['s'], color = 'grey', label = r'Total creep, n = 3, d = 200 $\mu m$')

plt.plot(tab['t1-3_esp'], tab['s'], color = 'grey', ls = '--', label = 'Total creep, n = 3, d = grainSizeComp table')
plt.plot(tab['t1-4_esp'], tab['s'], color = 'orange', ls = '--', label = 'Total creep, n = 3, d = subSeed1 table')
plt.plot(tab['t1-5_esp'], tab['s'], color = 'teal', ls = '--', label = 'Total creep, n = 3, d = subSeed2 table')

for i in mylist:
    plt.scatter(tab.loc[i, 't1-3'], tab.loc[i, 's'], color = 'grey', alpha = 0.1)
    plt.scatter(tab.loc[i, 't1-4'], tab.loc[i, 's'], color = 'orange', alpha = 0.1)
    plt.scatter(tab.loc[i, 't1-5'], tab.loc[i, 's'], color = 'teal', alpha = 0.1)


plt.legend()
plt.xlabel(r'Log(Strain rate ($s^{-1}$))')
plt.ylabel('Log(Stress (MPa))' )
plt.xscale('log')
plt.yscale('log')
plt.show()
plt.savefig('creep_grainSizeComp-esperance.png')


# In[11]:


T = 1473
plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(10, 8), dpi=300)
plt.scatter(tab['t1-3_percent'], tab['t1-3'], c = tab['s'], cmap = 'bone')
plt.yscale('log')
cb = plt.colorbar()
cb.set_label(label='stress (MPa)')
plt.ylabel(r'strain rate ($s^{-1}$)')
plt.xlabel(f'percentage of dislocation creep on total strain, T = {T}K')
plt.savefig('creep_grainSizeComp-percent.png')


# In[12]:


T = 1473
plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(10, 8), dpi=300)
plt.scatter(tab['t1-4_percent'], tab['t1-4'], c = tab['s'], cmap = 'bone')
plt.yscale('log')
cb = plt.colorbar()
cb.set_label(label='stress (MPa)')
plt.ylabel(r'strain rate ($s^{-1}$)')
plt.xlabel(f'percentage of dislocation creep on total strain, T = {T}K')
plt.savefig('creep_subSeed1-percent.png')


# In[13]:


T = 1473
```

```
plt.rcParams["font.family"] = "serif"
plt.figure(figsize=(10, 8), dpi=300)
plt.scatter(tab['t1-5_percent'], tab['t1-5'], c = tab['s'], cmap = 'bone')
plt.yscale('log')
cb = plt.colorbar()
cb.set_label(label='stress (MPa)')
plt.ylabel(r'strain rate ($s^{-1}$)')
plt.xlabel(f'percentage of dislocation creep on total strain, T = {T}K')
plt.savefig('creep_subSeed2-percent.png')


# In[ ]:
```

```python
class CreepLaw():
    def __init__(self):

        self.R = 8.3145
        self.e = 2.718281828459045

    def _convert(self, s, var):
        '''
        Convert the variable array to the same as stress
        '''
        self.s = s
        self.T = T
        if type(var) == 'float':
            self.var = np.full((len(s), ), var)
        else:
            self.var = var

    def Hirth2003(self, s, var, T, param):
        '''
        Creep law associated to the review of G. Hirth and D. Kohlstedt, 2003.

        Arguments :
            s      stress                       [MPa]
            var    variable value (d)           [Î¼m]
            T      temperature                  [Kelvin]

        Output :
            S      strain rate                  [s-1]
        '''

        self._convert(s, var)
        self.S = param.A * self.s**param.n * (1/self.var)**param.p  * param.fH2O**param.r * np.exp(param.alpha*param.phi) * np.exp( -param.Q /(self.R * T))
        return self.S

    def Tommasi2021(self, s, var, T, param):
        '''
        Creep law associated to disocation creep.

        Arguments :
            s      stress                       [MPa]
            var    variable value (d)           [Î¼m]
            T      temperature                  [Kelvin]

        Output :
            S      strain rate                  [s-1]
        '''

        self._convert(s, var)
        self.S =  param.A * ( self.s*1e6/param.mu )**param.n * np.exp((-param.Q/(self.R*T)) * ( 1 - ( self.s*1e6/param.s_P )**param.p )**param.q)
        return self.S

    def Demouchy2013(self, s, var, T, param):
        '''
        Creep law semi-empirical determined on dro olivine experiments by S. Demouchy et al, 2013.

        Arguments :
            s      stress                       [MPa]
            var    variable value (d)           [Î¼m]
            T      temperature                  [Kelvin]

        Output :
            S      strain rate                  [s-1]
        '''

        self._convert(s, var)
        self.S =  param.A * ( self.s*1e6/param.mu )**param.n * np.exp((-param.Q/(self.R*T)) * ( 1 - ( self.s*1e6/param.s_P )**param.p )**param.q)
        return self.S

    def Creep(self, first, *args):
        '''
        Sum values of strain rate calculated by creep laws, calculate percentage part of the first argument
```

```

        Argument :


        Output :
            St     strain rate total         [s-1]
            perct  percentage of the 1st     0 < perct < 1
                   array
        '''

        self.St = first
        for a in args:
            self.St += a

        self.perct = (self.St - first) / self.St
        return self.St, self.perct


    def CREEP(self, s, var, T, param, phi, conditions = 'all'):

        if conditions == 'all':
            conditions = p.param.index

        plt.rcParams["font.family"] = "serif"
        plt.figure(figsize=(10, 8), dpi=300)

        self.Tommasi2021(s, var, T, param)
        plt.plot(self.S, self.s, label = 'dislocation Tommasi')

        for cond in conditions:
            param.SetCondition(cond, T, phi)
            self.Hirth2003(s, var, T, param)
            plt.plot(self.S, self.s, label = cond)

        plt.legend()
        plt.xlabel('Log(Strain rate (s-1))')
        plt.ylabel('Log(Stress (MPa))' )
        plt.xscale('log')
        plt.yscale('log')
        plt.show()

#for i in range(1,nStress):
#    stress = st.loc[i, 'stress']
    #print(stress)
#    strainStoch = 0
#    for j in range(1, nSample+1):
#        varStoch = varMinDiff
#        while (varStoch >= varMaxDiff) or (varStoch <= varMinDiff) :
            #print(varStoch)
#            varStoch = varMeanDiff + np.random.standard_normal() * varStdDiff
            #print(varStoch)
#        misfit = creepDiffusion(stress, varStoch, T) - strainStoch
#        strainStoch = strainStoch + misfit / j
#    st.loc[i, 'strainStochDiffusion'] = strainStoch
```

```
from laws import *

#decorateurs


#percent


#iterstrain
```

```python

def plotFit(x, y, dfs = True, allSamples = False):

    if dfs == False :
        dfw = pd.read_csv(’TUR_Olivine.txt’, sep = ’;’)

    if allSamples == True :
        d = dfs[dfs[’indexSets’] == i]

    d = d.sort_values(by = ’sse’, inplace = True)

    dist = getattr(stats, distribution)
    parameters = dist.fit(data)
    loc = parameters[-2]
    scale = parameters[-1]
    arg = parameters[:-2]


    pdf = dist.pdf(x, *arg, loc=loc, scale=scale)

    plt.rcParams["font.family"] = "serif"
    plt.figure(figsize=(8, 5), dpi=300)
    plt.plot(x, y, label="DonnÃ©es", color = ’red’)
    plt.plot(x, pdf, label=f’{distribution}’, linewidth=1)
    plt.legend(loc=’upper right’)
    plt.show()
```

```python
import numpy as np
import pandas as pd

def GenerateVariable(Min, Max, ds, mode = 'range'):

    if mode == 'range':
        var = np.arange(Min, Max, ds)

    if mode == 'linearSpace':
        var = np.linspace(Min, Max, num = ds)

    if mode == 'uniform':
        var = Min + np.random.uniform(0, 1, Nb) * (Max - Min)
        var = np.sort(var)

    return var


def stochasticSeed(Min, Max, Std, NbDraw, NbVal):

    df = pd.DataFrame()

    for i in range(0, NbVal):

        for j in range(0, NbDraw):

            varStoch = 0

            while (varStoch >= Max) or (varStoch <= Min):
                varStoch = Min + np.random.standard_normal() + Std
            df.loc[i, j] = varStoch

    return df


def statisticSeed(df,
                  calculation = ['min', 'max', 'std', 'mean', 'median', 'mad'],
                  quantile = [0.2, 0.4, 0.6, 0.8],
                  option = 1):


    for element in calculation:

        if element == 'min':
            df[element] = df.min(axis = option)

        if element == 'mean':
            df[element] = df.mean(axis = option)

        if element == 'max':
            df[element] = df.max(axis = option)

        if element == 'var':
            df[element] = df.var(axis = option)

        if element == 'std':
            df[element] = df.std(axis = option)

        if element == 'median':
            df[element] = df.median(axis = option)

        if element == 'mad':
            df[element] = df.mad(axis = option)


    for q in quantile:
        df[f'q{q}'] = df.quantile(q, axis = option)
    return df
```

```python
def loadF(filename):

    file = open(filename, 'r')
    fixedDict = {}
    stoDict = {}

    for line in file:
        if not line.startswith('###') and 'sto' not in line and 'nb' not in line:
            k, v = line.strip().split('=')
            try:
                fixedDict[k.strip()] = float(v.strip())
            except ValueError:
                fixedDict[k.strip()] = v.strip()

        if line.startswith('sto') or 'nb' in line:
            k, v = line.strip().split('=')
            try:
                stoDict[k.strip()] = float(v.strip())
            except ValueError:
                stoDict[k.strip()] = v.strip()
    file.close()

    return fixedDict, stoDict
```

```python
import numpy as np
import load

class Fixed():
    def __init__(self, Dict):

        self.attribute = []

        for k, v in Dict.items():
            setattr(self, k, v)

    def GenerateVariable(self, mode = 'uniform', **kwargs):

        if len(kwargs) == 0:
            Min = self.Min
            Max = self.Max
            ds = self.ds
            name = self.name

        else:
            Min = kwargs['Min']
            Max = kwargs['Max']
            ds = kwargs['ds']
            name = kwargs['name']

        if mode == 'range':
            var = np.arange(Min, Max, ds)

        if mode == 'linearSpace':
            var = np.linspace(Min, Max, num = ds)

        if mode == 'uniform':
            ds = int(ds)
            var = Min + np.random.uniform(0, 1, ds) * (Max - Min)
            var = np.sort(var)

        self.mode = mode
        self.attribute.append(name)

        setattr(self, name, var)

        txt = f'{name} :    {Min}  {Max}  {ds}  {mode}'
        setattr(self, f'm{name}', txt)


        print(f'Attribute "{name}" set.')
```

```
import numpy as np
import pandas as pd


class Stochastic():
    def __init__(self, Dict):

        for k, v in Dict.items():
            setattr(self, k, v)

        self.attribute = []

    def stochasticSeed(self, **kwargs):

        if len(kwargs) == 0:
            Min = self.stoMin
            Max = self.stoMax
            Std = self.stoStd
            NbStep = int(self.nbStep)
            NbDraw = int(self.nbDraw)
            name = self.stoName

        else:
            Min = kwargs['Min']
            Max = kwargs['Max']
            Std = kwargs['Std']
            NbStep = int(kwargs['nbStep'])
            NbDraw = int(kwargs['nbDraw'])
            name = kwargs['name']

        self.tab = np.zeros((NbStep, NbDraw))

        for i in range(0, NbStep):

            for j in range(0, NbDraw):

                varStoch = 0

                while (varStoch >= Max) or (varStoch <= Min):
                    varStoch = Min + np.random.standard_normal() + Std
                    self.tab[i, j] = varStoch

        col = [str(x) for x in range(0, NbDraw)]
        self.tab = pd.DataFrame(self.tab, columns = col)

        self.attribute.append(name)

        setattr(self, name, self.tab)

        txt = f'{name} :    {Min}  {Max}  {Std}  normal[{NbStep}, {NbDraw}]'
        setattr(self, f'm{name}', txt)

        print(f'Stochastic seed "{name}" set')


    def statisticSeed(self,
                  calculation = ['min', 'max', 'std', 'mean', 'median', 'mad'],
                  quantile = [0.2, 0.4, 0.6, 0.8],
                  option = 1, **kwargs):

        if len(kwargs) == 0:
            df = self.tab
            name = self.stoName

        else:
            df = kwargs['df']
            name = kwargs['name']

        for element in calculation:

            if element == 'min':
                df[element] = df.min(axis = option)

            if element == 'mean':
                df[element] = df.mean(axis = option)
```

```python

            if element == 'max':
                df[element] = df.max(axis = option)

            if element == 'var':
                df[element] = df.var(axis = option)

            if element == 'std':
                df[element] = df.std(axis = option)

            if element == 'median':
                df[element] = df.median(axis = option)

            if element == 'mad':
                df[element] = df.mad(axis = option)


        for q in quantile:
            df[f'q{q}'] = df.quantile(q, axis = option)


        setattr(self, f'{name}', df)

        print(f'Characterisation of stochastic seed "{name}" made')
```

```
# functions

def load(i, title = 'EGD', plot = False):
    df = pd.read_csv(sets[i], sep = ';')
    name = sets[i]
    # Array
    if title == 'EGD':
        data = df.equivalentRadius*2
    elif title == 'log':
        data = np.log10(df.equivalentRadius*2)
    elif title == 'areaPond':
        data = (df.equivalentRadius*2 * df.area)/0.7

    # Histogramme des donnÃ©es
    bins='auto'
    y, x = np.histogram(data, bins=bins, density=True)
    # Milieu de chaque classe
    x = (x + np.roll(x, -1))[:-1] / 2.0

    if plot == True:

        plt.rcParams["font.family"] = "serif"
        plt.figure(figsize=(8, 5), dpi=300)
        n, bins, patches = plt.hist(data, bins=bins, density=True)
        plt.xlabel(r'$\mu m$')
        plt.title(f'{sets[i]}')
        plt.savefig(f'{name}_{title}.png', transparent = False)
        plt.show()

    return data, x, y, name


def Characterise(data, x, y, name, distNames = stats._distr_params.distcont):


    y, x = np.histogram(data, bins='auto', density=True)
    # Milieu de chaque classe
    x = (x + np.roll(x, -1))[:-1] / 2.0

    for distribution in distNames:

        if distNames == stats._distr_params.distcont:
            distribution = distribution[0]
        print(distribution)

        with warnings.catch_warnings(record=True) as w:
    # Cause all warnings to always be triggered.
            warnings.simplefilter("always")
            w = 0

            if w == 0:
                try:
                    dist = getattr(stats, distribution)
                    parameters = dist.fit(data)
                    loc = parameters[-2]
                    scale = parameters[-1]
                    arg = parameters[:-2]

                    ######## Sum square error
                    pdf = dist.pdf(x, *arg, loc=loc, scale=scale)
                    sse = np.sum( (y - pdf)**2 )

                    ######## Kolmogorov-Smirnov test for goodness of fit
                    p = stats.kstest(data, distribution, args = parameters)[1]

                    param = "_".join([str(_) for _ in parameters])
                except:
                    pass

                with open('OLIVINElog_.txt', 'a') as file:
                    file.write(f'{name};{distribution};{param[0]};{param[1]};{sse};{p}\n')
                    print('done')
            elif w == 1:
                pass
                print('RunTime Warning : bad distribution')
```

```
        del w

        loc = parameters[-2]
        scale = parameters[-1]
        arg = parameters[:-2]


        pdf = dist.pdf(x, *arg, loc=loc, scale=scale)

        plt.rcParams["font.family"] = "serif"
        plt.figure(figsize=(8, 5), dpi=300)
        plt.plot(x, y, label="Data", color = 'red')
        plt.plot(x, pdf, label=f'{distribution}', linewidth=1)
        plt.legend(loc='upper right')
        plt.show()
        plt.savefig(f'{name}_{distribution}.png')

    return x, y, pdf, parameters


def fitter(data, sets, i):
    f = Fitter(data)
    f.fit()
    f.summary()
    plt.savefig(f'{sets[i]}.png', transparent = False)

```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May  5 11:26:06 2021

@author: antoinemaitre
"""

import numpy as np
import matplotlib.pyplot as plt


"""Construction du profil rhÃ©ologique du gabbro du Queyras"""


'Gradient gÃ©othermique'

z = np.arange(0,100,0.1)              #Profondeur (km)
q = 15*10**(-3)                       #Chaleur (J = N.m)
T = (q*(z*10**3))+273                 #TempÃ©rature (K)



"""Comportement fragile - Byerlee"""

'Pression lithostatique'

rho = 3300                            #Masse volumique (kg/m3)

g = 9.81                              #AccÃ©lÃ©ration de pesanteur (m/s2)
Plith = rho*g*(z*10**3)               #Pression lithostatique (Pa)
mu = 0.6                              #Coefficient de friction (/)
pf=0.9                                #Pression fluide (comprise entre 0 et 1)

taub = (mu*Plith*(1-pf))*10**(-6)     #Contrainte dÃ©viatorique (MPa)



"""Comportement ductile - fluage dislocation"""

'Albite'

n = 3                        #Exposant de contrainte (/)
logA1 = 3.4
A1 = 10**logA1               #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
#fH2O= 0.2                   #Pourcentage d'H2O (wt%)
Q = 332*10**3                #Ã\211nergie d'activation (J.mol^(-1))
R = 8.314                    #Constante des gaz parfaits ((kPa.L)/(mol.K))
strainrate = 10**(-14)       #Taux de dÃ©formation (s-1)

taud1 = (strainrate/A1)**(1/n)*np.exp(Q/(R*T*n))    #Contrainte dÃ©viatorique (MPa)


# 'Anorthite 1'

# n = 3                        #Exposant de contrainte (/)
# logA2 = 12.7
# A2 = 10**logA2               #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
# #fH2O= 0.2                   #Pourcentage d'H2O (wt%)
# Q = 648*10**3                #Ã\211nergie d'activation (J.mol^(-1))
# R = 8.314                    #Constante des gaz parfaits ((kPa.L)/(mol.K))
# strainrate = 10**(-14)       #Taux de dÃ©formation (s-1)

# taud2 = (strainrate/A2)**(1/n)*np.exp(Q/(R*T*n))    #Contrainte dÃ©viatorique (MPa)

# 'Anorthite 2'

# n = 3                        #Exposant de contrainte (/)
# logA3 = 2.6
# A3 = 10**logA3               #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
# #fH2O= 0.2                   #Pourcentage d'H2O (wt%)
# Q = 356*10**3                #Ã\211nergie d'activation (J.mol^(-1))
# R = 8.314                    #Constante des gaz parfaits ((kPa.L)/(mol.K))
# strainrate = 10**(-14)       #Taux de dÃ©formation (s-1)

```

```
# taud3 = (strainrate/A3)**(1/n)*np.exp(Q/(R*T*n))    #Contrainte dÃ©viatorique (MPa)


'Clinopyroxène (diospide wet)'

n = 5.5                      #Exposant de contrainte (/)
logA4 = 0.8
A4 = 10**logA4               #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
#fH2O= 0.2                   #Pourcentage d'H2O (wt%)
Q = 534*10**3                #Ã\211nergie d'activation (J.mol^(-1))
R = 8.314                    #Constante des gaz parfaits ((kPa.L)/(mol.K))
strainrate = 10**(-14)       #Taux de dÃ©formation (s-1)

taud4 = (strainrate/A4)**(1/n)*np.exp(Q/(R*T*n))    #Contrainte dÃ©viatorique (MPa)


'Amphibole'

#n = 3
#logA5 =
#A5 = 10**logA5
#fH2O= 0.2
#Q =
#R = 8.314
#strainrate = 10**(-14)      #Taux de dÃ©formation (s-1)

#taud5 = (strainrate/A5)**(1/n)*np.exp(Q/(R*T*n))    #Contrainte dÃ©viatorique (MPa)


"""ReprÃ©sentation"""


'Profil rhÃ©ologique (Loi de fluage - Profondeur)'

plt.figure(1)

#Loi de Byerlee :

#plt.plot(taub,-z,'black', label='Loi de Byerlee', linewidth = 2)

#Loi de fluage : Albite - Arnothite 1 - Anorthite 2 - ClinopyroxÃ¨ne :

#plt.plot(taud1,-z,'-g',label='Albite wet', linewidth = 2)
#plt.plot(taud2,-z,'--b',label='Anorthite 1',linewidth = 1)
#plt.plot(taud3,-z,'--g',label ='Anorthite 2',linewidth = 1)
#plt.plot(taud4,-z,'-r',label='ClinopyroxÃ¨ne',linewidth = 2)
#plt.plot(T-273, -z,'b')

plt.legend(loc = 'lower right')
plt.grid()
plt.xlim(0,200)
plt.ylim (-100,0)
plt.xlabel('Contrainte dÃ©viatorique (MPa)') ; plt.ylabel('Profondeur (km)')
plt.title("Profil rhÃ©ologique")


'GÃ©otherme'

plt.figure(2)

plt.plot(T-273, -z,'r')

plt.xlabel('TempÃ©rature (Â°C)') ; plt.ylabel('Profondeur (km)')
plt.title("GÃ©otherme")
plt.grid()
plt.legend()


'Loi de fluage'

plt.figure(3)

plt.plot(T-273,taud1,'-y',label='Albite wet')
#plt.plot(T-273,taud2,'--b',label='Anorthite 1')
#plt.plot(T-273,taud3,'--g',label ='Anorthite 2')
```

```
plt.plot(T-273,taud4,'-r',label='ClinopyroxÃ¨ne')

plt.ylim(0,800)
plt.grid()
plt.legend(loc = 'upper right')
plt.xlabel('TempÃ©rature (Â°C)') ; plt.ylabel('Contrainte dÃ©viatorique (MPa)')
plt.title("Lois de fluage")
```

```
import numpy as np
import pandas as pd
import os

class Creep():
    def __init__(self, **kwargs):

        self.__dict__.update(kwargs)

        self.attribute = []                                                    # attribute created
        self.configurations = []

        self.R = 8.3145
        self.e = 2.718281828459045


    def SetCalibration(self, calib, process, condition, clear = []):      # read the wanted dataframe and the line associated to the right deformation process and the right condition
        df = pd.read_csv(f'/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/creeepy/calibration/{calib}.txt', sep = ';')
        df = df[df['process'] == process]
        df = df[df['condition'] == condition]
        df.reset_index(drop = True, inplace = True)

        self.name = calib
        self.calibration = df
        self.process = process
        self.condition = condition
        self.listParam = self.calibration.columns

        if len(clear) >= 1:                                                     # clean parameters between two loadings
            for e in range(2, len(self.listParam)):
                attr = self.listParam[e]
                if hasattr(self, attr) and attr not in clear:
                    delattr(self, attr)


    def SetParams(self, fixed, stochastic):
        self.txt = f'Paramters used in the law {self.name}, for {self.condition} {self.process} : \n'

        for e in self.listParam:
            if 'process' not in e and 'condition' not in e:
                if hasattr(fixed, e):
                    setattr(self, e, getattr(fixed, e))
                    self.txt = self.txt + f'{e} = see tables      fixed \n'

                elif hasattr(stochastic, e):
                    setattr(self, e, getattr(stochastic, e))
                    self.txt = self.txt + f'{e} = see tables      stochastic \n'

                elif hasattr(self, e):
                    self.txt = self.txt + f'{e} = {getattr(self, e)}      set manually \n'

                else:
                    var = float(self.calibration.loc[0, e])
                    setattr(self, e, var)
                    self.txt = self.txt + f'{e} = {getattr(self, e)}      from calibration \n'

        self.varSto = stochastic.variables
        self.varFix = fixed.variables

        self.txt = self.txt + f'\nFixed array : {", ".join(self.varFix)}'
        self.txt = self.txt + f'\nStochastic array : {", ".join(self.varSto)}\n\n'

        self.configurations.append(self.txt)


    def SetVariables(self, variable, values):
        setattr(self, variable, values)


    def StochasticCreep(self, tab, variable, func, prop = 0):

```

```
        self.stoS = pd.DataFrame()

        for col in range(0, tab.shape[1]):

            var = tab.iloc[:,col]
            self.SetVariables(variable, var)
            func()
            self.stoS[tab.columns[col]] = self.S

        columns = self.stoS.columns
        self.stoS['meanSeed'] = self.stoS.mean(axis = 1)

        if prop >= 0:
            for e in range(1, prop+1):
                col = b = [x for x in columns if x.startswith(f'{e}_')]
                self.stoS[f'meanSubSeed{e}'] = self.stoS.loc[:, col].mean(axis = 1)


    def RegularCreep(self, func):
        if hasattr(self, 'rS') == False:
            self.rS = pd.DataFrame()
            self.rS['s'] = self.s
        func()
        self.rS[f'r_{self.name}'] = self.S


    def SetPercentage(self, df, col1, total):
        df[f'{total}_percent'] = df[col1]/df[total]


    def Esperance(self, tab, col):

        for j in range(1, len(tab.index)):
            tab.loc[j, f'{col}_esp'] = sum(tab.loc[0:j, col])/(j)
        return tab

    def TotalCreep(self, a, b, prop = 0):

        df = self.rS.copy()
        df[f'sto_{self.name}'] = self.stoS['meanSeed']

        if prop >= 0:
            for e in range(1, prop+1):
                df[f'sto_{self.name}_SubSeed{e}'] = self.stoS[f'meanSubSeed{e}']


        columns = list(df.columns)

        for i in range(0, len(a)):
            ind1 = columns.index(a[i])
            ind2 = columns.index(b[i])
            df[f't{ind1}-{ind2}'] = df.iloc[:,ind1] + df.iloc[:,ind2]

            self.SetPercentage(df, columns[ind1], f't{ind1}-{ind2}')
            df = self.Esperance(df, f't{ind1}-{ind2}')

        var = '-'.join(self.varSto)
        name = f'S_{var}_{self.T}'
        self.attribute.append(name)
        setattr(self, name, df)




################################################################################################
#

    def Hirth2003(self):
        '''
        Creep law associated to the review of G. Hirth and D. Kohlstedt, 2003.

        Arguments :
            s       stress                      [MPa]
            var     variable value (d)          [Î¼m]
            T       temperature                 [Kelvin]
```

```

        Output :
            S      strain rate               [s-1]
        '''

        self.S = self.A * self.s**self.n * (1/self.d)**self.p  * self.fH2O**self.r * np.exp(self.alpha*self.phi) * np.exp( -self.Q /(self.R * self.T))


    def Gouriet2018(self):
        '''
        Creep law associated to disocation creep.

        Arguments :
            s      stress                    [MPa]
            var    variable value (d)        [Î¼m]
            T      temperature               [Kelvin]

        Output :
            S      strain rate               [s-1]
        '''

        self.S =  self.A * ( self.s*1e6/self.mu )**self.n * np.exp((-self.Q/(self.R*self.T)) * ( 1 - ( self.s*1e6/self.s_P )**self.p )**self.q)
```

```python
import numpy as np
import pandas as pd

class Data():
    def __init__(self, filename):

        file = open(filename, 'r')                                        # open config file

        self.fixedDict = {}                                               # create dictionaries for both fixed and stochastic parameters
        self.stoDict = {}

        self.variables = []

        for line in file:
            if not line.startswith('###') and 'sto' not in line and 'nb' not in line:      # lines wich begin by 'sto' or 'nb' are affected to the stochastic dictionariy
                k, v = line.strip().split('=')
                try:
                    self.fixedDict[k.strip()] = float(v.strip())          # convert numbers to float for further calculations...
                except ValueError:
                    self.fixedDict[k.strip()] = v.strip()

            if line.startswith('sto') or 'nb' in line:
                k, v = line.strip().split('=')
                try:
                    self.stoDict[k.strip()] = float(v.strip())
                except ValueError:
                    self.stoDict[k.strip()] = v.strip()
        file.close()

    def SetVariable(self, var, newVar, clear = False):
        if clear == True:
            self.variables = []
        # dupplicate an attribute under another name
        setattr(self, var, getattr(self, newVar))
        self.variables.append(f'{var}_{newVar}')




class Fixed(Data):
    def __init__(self, filename = 'config'):
        super().__init__(filename)

        self.attribute = []                                               # keep a trace of the name of the variables generated for output

        for k, v in self.fixedDict.items():                               # update the class attribute with dictionary keys
            setattr(self, k, v)

    def GenerateVariable(self, mode = 'uniform', **kwargs):               # generate variables by three different manners ('uniform' or 'linearSpace' better, to make the number of stress values correspond to the number of points wanted)

        if len(kwargs) == 0:                                              # take the attributes of the class
            Min = self.Min
            Max = self.Max
            ds = self.ds
            name = self.name

        else:                                                             # take the kwargs (possibility to create other variables manually if all the wanted kwargs are declared)
            Min = kwargs['Min']
            Max = kwargs['Max']
            ds = kwargs['ds']
            name = kwargs['name']

        if mode == 'range':
            var = np.arange(Min, Max, ds)

```

```
        if mode == 'linearSpace':
            var = np.linspace(Min, Max, num = ds)

        if mode == 'uniform':
            ds = int(ds)
            var = Min + np.random.uniform(0, 1, ds) * (Max - Min)
            var = np.sort(var)

        self.mode = mode
        self.attribute.append(name)

        setattr(self, name, var)

        txt = f'{name} :    {Min}  {Max}  {ds}  {mode} \n'

        tname = f'gen_{name}'                                                          # create the txt for output
        setattr(self, tname, txt)

        print(f'Attribute "{name}" set.')


class Stochastic(Data):
    def __init__(self, filename = 'config'):
        super().__init__(filename)

        for k, v in self.stoDict.items():
            setattr(self, k, v)

        self.attribute = []

    def GenerateSeed(self, **kwargs):                                                  # same principle as Fixed()

        if len(kwargs) == 0:
            Min = self.stoMin
            Max = self.stoMax
            Std = self.stoStd
            NbStep = int(self.nbStep)
            NbDraw = int(self.nbDraw)
            name = self.stoName

        else:
            Min = kwargs['Min']
            Max = kwargs['Max']
            Std = kwargs['Std']
            NbStep = int(kwargs['nbStep'])
            NbDraw = int(kwargs['nbDraw'])
            name = kwargs['name']

        self.tab = np.zeros((NbStep, NbDraw))

        for i in range(0, NbStep):

            for j in range(0, NbDraw):

                varStoch = 0

                while (varStoch >= Max) or (varStoch <= Min):                          # affect the random value to the numpy 2D array
                    varStoch = Min + np.random.standard_normal() * Std
                    self.tab[i, j] = varStoch

        col = [str(x) for x in range(0, NbDraw)]
        self.tab = pd.DataFrame(self.tab, columns = col)                               # convert info dataframe

        self.attribute.append(name)

        setattr(self, name, self.tab)

        txt = f'{name} :    {Min}  {Max}  {Std}  normal[{NbStep}, {NbDraw}] \n'

        tname = f'gen_{name}'
        setattr(self, tname, txt)
```

```python

        print(f'Stochastic seed "{name}" set')


    def GenerateCompositeSeed(self):                                                               # experimental, no possibility to generate manually

        tab = pd.DataFrame()
        p = ''

        for e in range(1, int(self.stoDistrib) +1):                                               # for the good number of wanted distribution

            prop = f'stoProp{e}'
            snbdraw = self.nbDraw * getattr(self, prop)                                           # proportionally take the good number of random value to draw from nbDraw and the percentage of the distirbution

            smin = str(f'stoMin{e}')                                                              # update the distribu numbers
            smax = str(f'stoMax{e}')
            sstd = str(f'stoStd{e}')
            sname = str(f'stoName{e}')

            self.GenerateSeed(Min = getattr(self, smin),                                          # generate a sub seed manually
                              Max = getattr(self, smax),
                              Std = getattr(self, sstd),
                              nbDraw = snbdraw,
                              nbStep = self.nbStep,
                              name = getattr(self, sname))

            self.tab.columns = [f'{e}_{col}' for col in self.tab.columns]            # update the columns number with the number of the distribution :  e = number of the distribution, col = number of the value
            tab = pd.concat([tab, self.tab], axis = 1)                                             # concat to the upper seed

            p = p + f'             {getattr(self, sname)}, {getattr(self, prop)}\n'    # create a sub txt with the name and the proportion of the sub seeds


        self.attribute.append(self.stoNameComp)                                                    # add to the created attribute the upper seed name

        setattr(self, self.stoNameComp, tab)

        txt = f'{self.stoNameComp} : \n' + p                                                       # create a txt wich gather the upper seed name, the sub seeds name and their proportion for output

        tname = f'comp_{self.stoNameComp}'
        setattr(self, tname, txt)

        print(f'   --->  added to the stochastic seed {self.stoNameComp} \n')

    def CharacteriseSeed(self,                                                                     # can characterise the values of the values by lines...., possibility to juste make few calculation and not all with kwargs...
                  calculation = ['min', 'max', 'std', 'mean', 'median', 'mad'],
                  quantile = [0.2, 0.4, 0.6, 0.8],
                  option = 1, **kwargs):

        if len(kwargs) == 0:
            df = self.tab
            name = self.stoName

        else:
            df = kwargs['df']
            name = kwargs['name']

        for element in calculation:

            if element == 'min':
                df[element] = df.min(axis = option)

            if element == 'mean':
```

```
                df[element] = df.mean(axis = option)

            if element == 'max':
                df[element] = df.max(axis = option)

            if element == 'var':
                df[element] = df.var(axis = option)

            if element == 'std':
                df[element] = df.std(axis = option)

            if element == 'median':
                df[element] = df.median(axis = option)

            if element == 'mad':
                df[element] = df.mad(axis = option)


        for q in quantile:
            df[f'q{q}'] = df.quantile(q, axis = option)


        setattr(self, f'{name}', df)

        txt = f'Characterised by : \n parameters {calculation}, quantiles {quantile} on {option}'
        tname = f'cha_{name}'
        setattr(self, tname, txt)                                                          # keep calculations names in output

        print(f'Characterisation of stochastic seed "{name}" made')
```

```
import random
import pandas as pd
import matplotlib.pyplot as plt


class Plot():
    def __init__(self):

        self.plot = 0

    def ParamPlot(self, width = 10, height = 8, dpi = 300):
        plt.rcParams["font.family"] = "serif"
        plt.figure(figsize=(10, 8), dpi=300)


    def PlotsS(self, tab, a, color, label, esp = 0, nb = 100, Max = 1000):

        self.ParamPlot()

        columns = list(tab.columns)
        ind1 = columns.index(a[0])
        ind2 = columns.index(a[1])

        col = f't{ind1}-{ind2}'

        plt.plot(tab[col], tab['s'], color = color, label = label)

        if esp == 'esp':
            plt.plot(tab['s'], tab[f'{col}_esp'], color = color, ls = '--', label = label)
        elif esp == 'points':
            plt.scatter(tab.iloc[i, 's'], tab.iloc[i, f'{col}'], color = color, marker = 'o', alpha = 0.5, label = label)

        elif esp == 'esppoint':
            plt.plot(tab['s'], tab[f'{col}_esp'], color = color, ls = '--', label = label)
            plt.scatter(tab.iloc[0:nb, 's'], tab.iloc[i, f'{col}'], color = color, marker = 'o', alpha = 0.5, label = label)

        plt.legend()
        plt.xlabel(r'Log(Strain rate (s^{-1}))')
        plt.ylabel('Log(Stress (MPa))' )
        plt.xscale('log')
        plt.yscale('log')

    def Save(self, title):
        plt.saveFig(title)
```

```python
import matplotlib.pyplot as plt
from datetime import datetime


class Output():
    def __init__(self):

        self.date = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    def Describe(self, title, **kwargs):

        name = f'EXP_{title}'

        file = open(f'{title}.txt', 'w')

        file.write(f'{title}, {self.date} \n \n')


        for e in kwargs:

            if e == 'fixed':
                fixed = kwargs[e]
                file.write(f'\nFIXED PARAMETERS   : \n              Min        Max        ds        Mode \n')
                for e in fixed.attribute:
                    gen = f'gen_{e}'
                    cha = f'cha_{e}'
                    if hasattr(fixed, gen):
                        file.write(getattr(fixed, gen))
                    if hasattr(fixed, cha):
                        file.write(getattr(fixed, cha))

            if e == 'stochastic':
                stochastic = kwargs[e]
                file.write(f'\nSTOCHASTIC PARAMETERS   : \n              Min        Max        Std   distribution[NbStep, NbDraw] \n')
                for e in stochastic.attribute:
                    gen = f'gen_{e}'
                    cha = f'cha_{e}'
                    comp = f'comp_{e}'
                    if hasattr(stochastic, gen):
                        file.write(getattr(stochastic, gen))
                    if hasattr(stochastic, cha):
                        file.write(getattr(stochastic, cha))
                    if hasattr(stochastic, comp):
                        file.write(getattr(stochastic, comp))

            if e == 'creep':
                creep = kwargs[e]
                file.write(f'\n \nCREEP LAW   : \n')
                config = getattr(creep, 'configurations')
                for element in config:
                    file.write(f'{element} \n \n ')

    def SaveAttributes(self, obj, attribute = 'all'):

        if attribute == 'all':
            attribute = getattr(obj, 'attribute')

        for element in attribute:
            df = getattr(obj, element)
            df.to_csv(f'{element}.txt', sep = ';')
```

```
# functions

def load(i, title = 'EGD', plot = False):
    df = pd.read_csv(sets[i], sep = ';')
    name = sets[i]
    # Array
    if title == 'EGD':
        data = df.equivalentRadius*2
    elif title == 'log':
        data = np.log10(df.equivalentRadius*2)
    elif title == 'areaPond':
        data = (df.equivalentRadius*2 * df.area)/0.7

    # Histogramme des donnÃ©es
    bins='auto'
    y, x = np.histogram(data, bins=bins, density=True)
    # Milieu de chaque classe
    x = (x + np.roll(x, -1))[:-1] / 2.0

    if plot == True:

        plt.rcParams["font.family"] = "serif"
        plt.figure(figsize=(8, 5), dpi=300)
        n, bins, patches = plt.hist(data, bins=bins, density=True)
        plt.xlabel(r'$\mu m$')
        plt.title(f'{sets[i]}')
        plt.savefig(f'{name}_{title}.png', transparent = False)
        plt.show()

    return data, x, y, name


def Characterise(data, x, y, name, distNames = stats._distr_params.distcont):


    y, x = np.histogram(data, bins='auto', density=True)
    # Milieu de chaque classe
    x = (x + np.roll(x, -1))[:-1] / 2.0

    for distribution in distNames:

        if distNames == stats._distr_params.distcont:
            distribution = distribution[0]
        print(distribution)

        with warnings.catch_warnings(record=True) as w:
    # Cause all warnings to always be triggered.
            warnings.simplefilter("always")
            w = 0

            if w == 0:
                try:
                    dist = getattr(stats, distribution)
                    parameters = dist.fit(data)
                    loc = parameters[-2]
                    scale = parameters[-1]
                    arg = parameters[:-2]

                    ######## Sum square error
                    pdf = dist.pdf(x, *arg, loc=loc, scale=scale)
                    sse = np.sum( (y - pdf)**2 )

                    ######## Kolmogorov-Smirnov test for goodness of fit
                    p = stats.kstest(data, distribution, args = parameters)[1]

                    param = "_".join([str(_) for _ in parameters])
                except:
                    pass

                with open('OLIVINElog_.txt', 'a') as file:
                    file.write(f'{name};{distribution};{param[0]};{param[1]};{sse};{p}\n')
                    print('done')
            elif w == 1:
                pass
                print('RunTime Warning : bad distribution')
```

```
        del w

        loc = parameters[-2]
        scale = parameters[-1]
        arg = parameters[:-2]


        pdf = dist.pdf(x, *arg, loc=loc, scale=scale)

        plt.rcParams["font.family"] = "serif"
        plt.figure(figsize=(8, 5), dpi=300)
        plt.plot(x, y, label="Data", color = 'red')
        plt.plot(x, pdf, label=f'{distribution}', linewidth=1)
        plt.legend(loc='upper right')
        plt.show()
        plt.savefig(f'{name}_{distribution}.png')

    return x, y, pdf, parameters


def fitter(data, sets, i):
    f = Fitter(data)
    f.fit()
    f.summary()
    plt.savefig(f'{sets[i]}.png', transparent = False)

```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May  5 11:26:06 2021

@author: antoinemaitre
"""

import numpy as np
import matplotlib.pyplot as plt


"""Construction du profil rhÃ©ologique du gabbro du Queyras"""


'Gradient gÃ©othermique'

z = np.arange(0,100,0.1)              #Profondeur (km)
q = 15*10**(-3)                       #Chaleur (J = N.m)
T = (q*(z*10**3))+273                 #TempÃ©rature (K)



"""Comportement fragile - Byerlee"""

'Pression lithostatique'

rho = 3300                            #Masse volumique (kg/m3)

g = 9.81                              #AccÃ©lÃ©ration de pesanteur (m/s2)
Plith = rho*g*(z*10**3)               #Pression lithostatique (Pa)
mu = 0.6                              #Coefficient de friction (/)
pf=0.9                                #Pression fluide (comprise entre 0 et 1)

taub = (mu*Plith*(1-pf))*10**(-6)     #Contrainte dÃ©viatorique (MPa)



"""Comportement ductile - fluage dislocation"""

'Albite'

n = 3                        #Exposant de contrainte (/)
logA1 = 3.4
A1 = 10**logA1               #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
#fH2O= 0.2                   #Pourcentage d'H2O (wt%)
Q = 332*10**3                #Ã\211nergie d'activation (J.mol^(-1))
R = 8.314                    #Constante des gaz parfaits ((kPa.L)/(mol.K))
strainrate = 10**(-14)       #Taux de dÃ©formation (s-1)

taud1 = (strainrate/A1)**(1/n)*np.exp(Q/(R*T*n))    #Contrainte dÃ©viatorique (MPa)


# 'Anorthite 1'

# n = 3                        #Exposant de contrainte (/)
# logA2 = 12.7
# A2 = 10**logA2               #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
# #fH2O= 0.2                   #Pourcentage d'H2O (wt%)
# Q = 648*10**3                #Ã\211nergie d'activation (J.mol^(-1))
# R = 8.314                    #Constante des gaz parfaits ((kPa.L)/(mol.K))
# strainrate = 10**(-14)       #Taux de dÃ©formation (s-1)

# taud2 = (strainrate/A2)**(1/n)*np.exp(Q/(R*T*n))    #Contrainte dÃ©viatorique (MPa)

# 'Anorthite 2'

# n = 3                        #Exposant de contrainte (/)
# logA3 = 2.6
# A3 = 10**logA3               #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
# #fH2O= 0.2                   #Pourcentage d'H2O (wt%)
# Q = 356*10**3                #Ã\211nergie d'activation (J.mol^(-1))
# R = 8.314                    #Constante des gaz parfaits ((kPa.L)/(mol.K))
# strainrate = 10**(-14)       #Taux de dÃ©formation (s-1)

```

```
# taud3 = (strainrate/A3)**(1/n)*np.exp(Q/(R*T*n))    #Contrainte dÃ©viatorique (MPa)


'ClinopyroxÃ¨ne (diospide wet)'

n = 5.5                       #Exposant de contrainte (/)
logA4 = 0.8
A4 = 10**logA4                #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
#fH2O= 0.2                    #Pourcentage d'H2O (wt%)
Q = 534*10**3                 #Ã\211nergie d'activation (J.mol^(-1))
R = 8.314                     #Constante des gaz parfaits ((kPa.L)/(mol.K))
strainrate = 10**(-14)        #Taux de dÃ©formation (s-1)

taud4 = (strainrate/A4)**(1/n)*np.exp(Q/(R*T*n))    #Contrainte dÃ©viatorique (MPa)


'Amphibole'

#n = 3
#logA5 =
#A5 = 10**logA5
#fH2O= 0.2
#Q =
#R = 8.314
#strainrate = 10**(-14)       #Taux de dÃ©formation (s-1)

#taud5 = (strainrate/A5)**(1/n)*np.exp(Q/(R*T*n))    #Contrainte dÃ©viatorique (MPa)


"""ReprÃ©sentation"""


'Profil rhÃ©ologique (Loi de fluage - Profondeur)'

plt.figure(1)

#Loi de Byerlee :

#plt.plot(taub,-z,'black', label='Loi de Byerlee', linewidth = 2)

#Loi de fluage : Albite - Arnothite 1 - Anorthite 2 - ClinopyroxÃ¨ne :

#plt.plot(taud1,-z,'-g',label='Albite wet', linewidth = 2)
#plt.plot(taud2,-z,'--b',label='Anorthite 1',linewidth = 1)
#plt.plot(taud3,-z,'--g',label ='Anorthite 2',linewidth = 1)
#plt.plot(taud4,-z,'-r',label='ClinopyroxÃ¨ne',linewidth = 2)
#plt.plot(T-273, -z,'b')

plt.legend(loc = 'lower right')
plt.grid()
plt.xlim(0,200)
plt.ylim (-100,0)
plt.xlabel('Contrainte dÃ©viatorique (MPa)') ; plt.ylabel('Profondeur (km)')
plt.title("Profil rhÃ©ologique")


'GÃ©otherme'

plt.figure(2)

plt.plot(T-273, -z,'r')

plt.xlabel('TempÃ©rature (Â°C)') ; plt.ylabel('Profondeur (km)')
plt.title("GÃ©otherme")
plt.grid()
plt.legend()


'Loi de fluage'

plt.figure(3)

plt.plot(T-273,taud1,'-y',label='Albite wet')
#plt.plot(T-273,taud2,'--b',label='Anorthite 1')
#plt.plot(T-273,taud3,'--g',label ='Anorthite 2')
```

```
plt.plot(T-273,taud4,'-r',label='ClinopyroxÃ¨ne')

plt.ylim(0,800)
plt.grid()
plt.legend(loc = 'upper right')
plt.xlabel('TempÃ©rature (Â°C)') ; plt.ylabel('Contrainte dÃ©viatorique (MPa)')
plt.title("Lois de fluage")
```

```
###############################################################################
#              Makefile for Python scripts for Adeli                          #
#-----------------------------------------------------------------------------#

###############################################################################
# Define exec name and obj directory:
#------------------------------------------------------------------------------

PS2P := creeepy2p.ps
PDF2P := creeepy2p.pdf

PS := creeepy.ps
PDF := creeepy.pdf

###############################################################################
# Source and object files:
#------------------------------------------------------------------------------

SRCS := `find ~/RhEoVOLUTION/CODES/SCRIPTS/creeepy -type f -name "*.py"`

###############################################################################
# Make a pdf :

SRCS += makefile

write:
	enscript -2r --line-numbers --highlight=fortran --toc --fancy-header=header $(SRCS) -o $(PS2P)
	gs -sDEVICE=pdfwrite -o $(PDF2P) $(PS2P)

print:
	enscript -MA4 --line-numbers --highlight=fortran --toc --fancy-header=headerB5 -fCourier8.5 $(SRCS) -o $(PS)
	gs -sDEVICE=pdfwrite -o $(PDF) $(PS)
	#gs -q -sDEVICE=pdfwrite -dBATCH -dNOPAUSE -sOutputFile=print.pdf \
	-dDEVICEWIDTHPOINTS=595 -dDEVICEHEIGHTPOINTS=842 -dFIXEDMEDIA \ -c "<< /CurrPageNum 1 def /Install { /CurrPageNum CurrPageNum 1 add def CurrPageNum 2 mod 1 eq {} {96 0 translate} ifelse } bind  >> setpagedevice " -f "out.pdf"
	#rm out.pdf


###############################################################################
# Delete objects:
#------------------------------------------------------------------------------

clean:
	/bin/rm $(DIROBJ)/*.o

cleanmod:
	/bin/rm $(DIROBJ)/*.mod

all:
	@echo $(OBJS)
```