

```

1: from creeepy.parameters import *
2: from creeepy.law import *
3:
4:
5: f = Fixed('config')
6: s = Stochastic('config')
7:
8: f.GenerateVariable()
9: s.GenerateSeed()
10: s.CharacteriseSeed()
11:
12: creep = Creep(T = 1473, A = 1.5e9, n = 1, p = 3, Q = 375e3, R = 8.3145, r = 0, fH20 = 1, alpha = 40,
phi = 0.001)
13: creep.SetCalibration('Hirth_2003', 'diffusion', 'dry')
14: creep.SetParams(f, s)
15: creep.SetVariables(f.s, s.d, 'mean', 'd')
16: creep.d = creep.SR['mean']
17:
18: creep.Hirth2003()
19: creep.SaveSR('meanStrain') # creep for the mean grain size at each step of stress
20: creep.StochasticCreep(s.d, s.nbDraw) # mean creep of the grain sizes for each step of stress
21: creep.d = s.stoMean
22: creep.Hirth2003()
23: creep.SaveSR('theoStrain')
24:
25: 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):
26:
27:     strainRate = A * ( stress*1e6/mu )**val * np.exp((-Q/(R*T)) * ( 1 - ( stress*1e6/peierls )**p )
**q)
28:     return strainRate
29:
30: comp = creep.SR.copy()
31:
32: comp['Disl'] = Gouriet_2018(comp.s, 3, 1473)
33:
34: for j in range(1, len(comp.index)):
35:     comp.loc[j, 'esperanceMeanStrain'] = sum(comp.loc[0:j, 'meanStrain'])/(j)
36:     comp.loc[j, 'esperanceStoStrain'] = sum(comp.loc[0:j, 'stoStrain'])/(j)
37:
38: import matplotlib.pyplot as plt
39:
40: _ = plt.hist(s.tab, bins=100)
41:
42: plt.title("Histogram with 'auto' bins")
43: plt.savefig('distribGrainSize.png')
44:
45:
46:
47: import matplotlib.pyplot as plt
48:
49: plt.rcParams["font.family"] = "serif"
50: plt.figure(figsize=(10, 8), dpi=300)
51:
52:
53: plt.xlabel(rf'Log(Strain rate (${s}^{-1}$))', T = {creep.T}K')
54: plt.ylabel('Log(Stress (MPa))' )
55: plt.xscale('log')
56: plt.yscale('log')
57:
58: plt.plot(comp.Disl, comp.s, color = 'black', label = f'dislocation creep, Gouriet et al 2018 (n = 3)
')
59: plt.plot(comp.theoStrain, comp.s, color = 'red', label = rf'diffusion creep, Hirth and Kohltsedt 200
3, (d = {s.stoMean}$\mu$ m$)$')
60:
61: plt.legend()
62: plt.savefig('2creep.png')
63:
64: plt.rcParams["font.family"] = "serif"
65: plt.figure(figsize=(10, 8), dpi=300)
66:
67: plt.plot(comp.theoStrain, comp.s, color = 'red', label = rf'd = {s.stoMean}$\mu$ m$')
68: plt.scatter(comp.stoStrain, comp.s, color = 'red', marker = 'o', alpha = 0.05)
69: plt.scatter(comp.meanStrain, comp.s, color = 'red', marker = '+', alpha = 0.05)
70: plt.plot(comp.esperanceStoStrain, comp.s, color = 'red', ls = '--', label = 'esperance of the mean o

```

```

f strains for each d, for each stress step')
71: plt.plot(comp.esperanceMeanStrain, comp.s, color = 'red', ls = ':', label = 'esperance of the mean d
for each stress step')
72: plt.xlabel(rf'Log(Strain rate ( $s^{-1}$ )), T = {creep.T}K')
73: plt.ylabel('Log(Stress (MPa))' )
74: plt.legend()
75: plt.xscale('log')
76: plt.yscale('log')
77: plt.savefig('diffSto.png')
78:
79:
80: esp = pd.DataFrame()
81:
82: stress = comp.s
83: T = creep.T
84:
85: esp['totalCreep'] = comp.Disl + comp.theoStrain
86: esp['totalCreepDiffSto'] = comp.Disl + comp.stoStrain
87: esp['percent'] = comp.Disl/(comp.Disl + comp.stoStrain)
88: esp['stress'] = stress
89:
90:
91: for j in range(1, len(esp.index)):
92:     esp.loc[j, 'esperanceTotalStoDiff'] = sum(esp.loc[0:j, 'totalCreepDiffSto'])/(j)
93:
94: plt.rcParams["font.family"] = "serif"
95: plt.figure(figsize=(10, 8), dpi=300)
96:
97: plt.plot(esp.totalCreep, stress, color = 'gray', label = f'total creep, T = {T}')
98: plt.scatter(esp['totalCreepDiffSto'], stress, marker = '+', alpha = 0.1, color = 'gray')
99: plt.plot(esp.esperanceTotalStoDiff, stress, color = 'gray', ls = '--', label = 'esperance')
100:
101: point = esp[esp['percent']>0.25]
102: 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%')
103:
104: point = esp[esp['percent']>0.5]
105: 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%')
106:
107: point = esp[esp['percent']>0.75]
108: 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%')
109:
110: plt.xlabel('Log(Strain rate ( $s^{-1}$ ))')
111: plt.ylabel('Log(Stress (MPa))' )
112: plt.legend()
113: plt.xscale('log')
114: plt.yscale('log')
115: plt.savefig('esperanceCreep.png')
116:
117:
118: plt.rcParams["font.family"] = "serif"
119: plt.figure(figsize=(10, 8), dpi=300)
120: plt.scatter(esp.percent, esp.totalCreep, c = esp.stress, cmap = 'bone')
121: plt.yscale('log')
122: cb = plt.colorbar()
123: cb.set_label(label='stress (MPa)')
124: plt.ylabel(r'strain rate ( $s^{-1}$ ))')
125: plt.xlabel(f'percentage of dislocation creep on total strain, T = {T}K')
126: plt.savefig('esperanceCreeppercent.png')

```

```

1: #!/usr/bin/env python
2: # coding: utf-8
3:
4: # In[2]:
5:
6:
7: from creeepy.parameters import *
8: from creeepy.creep import *
9: from creeepy.output import *
10: from creeepy.plot import *
11:
12:
13: fix = Fixed('config')
14: sto = Stochastic('config')
15:
16: fix.GenerateVariable()
17: sto.GenerateSeed()
18:
19: fix.SetVariable('s', fix.name)
20:
21:
22: plt.rcParams["font.family"] = "serif"
23: plt.rcParams.update({'font.size': 8})
24: plt.figure(figsize=(5, 3), dpi=300)
25:
26: _ = plt.hist(sto.grainSize, bins='auto')
27:
28: plt.text(160,11, f'mean = {sto.stoMean}\nmax = {sto.stoMax}\nmin = {sto.stoMin}\nstd = {sto.stoStd}\n\
distribution = normal\nnbStep = {sto.nnbStep}\nnbDraw = {sto.nnbDraw}')
29:
30: plt.xlabel(r'Grain size ( $\mu$  m)')
31: plt.savefig('distribGrainSize.png')
32:
33:
34: # In[3]:
35:
36:
37: creep = Creep(T = 1473, phi = 0.001, d = 200)
38:
39: creep.SetCalibration('Gouriet2018', 'dislocation', 'dry')
40: creep.SetParams(fix, sto)
41: creep.RegularCreep(creep.Gouriet2018)
42:
43: creep.SetCalibration('Hirth2003', 'diffusion', 'dry', clear = ['T'])
44: creep.T = 1473
45: creep.d = 200
46: creep.SetParams(fix, sto)
47: creep.RegularCreep(creep.Hirth2003)
48:
49: plt.rcParams["font.family"] = "serif"
50: plt.figure(figsize=(10, 8), dpi=300)
51: plt.plot(creep.rS.r_Gouriet2018, creep.rS.s, label = 'dislocation creep, n = 3 (Gouriet et al 2018)',
, color = 'black')
52: plt.plot(creep.rS.r_Hirth2003, creep.rS.s, label = r'diffusion creep, d = 200  $\mu$  m (Hirth and Koh
lstedt 2003)', color = 'red')
53: plt.legend()
54: plt.xlabel(r'Log(Strain rate ( $s^{-1}$ ))')
55: plt.ylabel('Log(Stress (MPa))')
56: plt.xscale('log')
57: plt.yscale('log')
58: plt.show()
59: plt.savefig('creep_grainSize.png')
60:
61:
62: # In[4]:
63:
64:
65: creep.SetCalibration('Hirth2003', 'diffusion', 'dry')
66: sto.SetVariable('d', sto.stoName, clear = True)
67: creep.SetParams(fix, sto)
68: creep.StochasticCreep(sto.grainSize, 'd', creep.Hirth2003)
69: a = ['r_Gouriet2018', 'r_Gouriet2018']
70: b = ['r_Hirth2003', f'sto_{creep.name}']
71: creep.TotalCreep(a, b)
72:

```

```

73:
74: # In[17]:
75:
76:
77: import random
78:
79: mylist = []
80:
81: for i in range(0,100):
82:     x = random.randint(1,sto.nbStep)
83:     mylist.append(x)
84:
85:
86: # In[18]:
87:
88:
89: plt.rcParams["font.family"] = "serif"
90: plt.figure(figsize=(12, 10), dpi=300)
91:
92: tab = creep.S_d_grainSize_1473
93:
94: plt.plot(tab['t1-2'], tab['s'], color = 'grey', label = r'Total creep, n = 3, d = 200 $\mu$ m$')
95: plt.plot(tab['t1-3_esp'], tab['s'], color = 'grey', ls = '--', label = 'Total creep, n = 3, d = grainSize table')
96:
97: for i in mylist:
98:     plt.scatter(tab.loc[i, 't1-3'], tab.loc[i, 's'], color = 'grey', alpha = 0.1)
99:
100:
101: plt.legend()
102: plt.xlabel(r'Log(Strain rate ($s^{-1}$))')
103: plt.ylabel('Log(Stress (MPa))')
104: plt.xscale('log')
105: plt.yscale('log')
106: plt.show()
107: plt.savefig('creep_grainSize-esperance.png')
108:
109:
110: # In[6]:
111:
112:
113: T = 1473
114: plt.rcParams["font.family"] = "serif"
115: plt.figure(figsize=(10, 8), dpi=300)
116: plt.scatter(tab['t1-3_percent'], tab['t1-3'], c = tab['s'], cmap = 'bone')
117: plt.yscale('log')
118: cb = plt.colorbar()
119: cb.set_label(label='stress (MPa)')
120: plt.ylabel(r'strain rate ($s^{-1}$)')
121: plt.xlabel(f'percentage of dislocation creep on total strain, T = {T}K')
122: plt.savefig('creep_grainSize-percent.png')
123:
124:
125: # In[9]:
126:
127:
128: sto.GenerateCompositeSeed()
129:
130: plt.rcParams["font.family"] = "serif"
131: plt.rcParams.update({'font.size': 8})
132: plt.figure(figsize=(5, 3), dpi=300)
133:
134: out = Output()
135: out.Describe('test', fixed = fix, stochastic = sto, creep = creep)
136: out.SaveAttributes(sto)
137: out.SaveAttributes(creep)
138:
139: _ = plt.hist(sto.grainSizeComp, bins='auto')
140:
141: plt.text(1000,400, f'nbStep = {sto.nbStep}\nnbDraw = {sto.nbDraw}')
142:
143: plt.xlabel(r'Grain size ($\mu$ m$')
144: plt.savefig('distribGrainSizeComp.png')
145:
146:

```

```

147: creep.SetCalibration('Hirth2003', 'diffusion', 'dry')
148: sto.SetVariable('d', sto.stoNameComp, clear = True)
149: creep.SetParams(fix, sto)
150: creep.StochasticCreep(sto.grainSizeComp, 'd', creep.Hirth2003, prop = 2)
151: a = ['r_Gouriet2018', 'r_Gouriet2018', 'r_Gouriet2018', 'r_Gouriet2018']
152: b = ['r_Hirth2003', f'sto_{creep.name}', 'sto_Hirth2003_SubSeed1', 'sto_Hirth2003_SubSeed2']
153: creep.TotalCreep(a, b, prop = 2)
154:
155:
156: # In[16]:
157:
158:
159: plt.rcParams["font.family"] = "serif"
160: plt.figure(figsize=(12, 10), dpi=300)
161:
162: tab = creep.S_d_grainSizeComp_1473
163:
164: plt.plot(tab['t1-2'], tab['s'], color = 'grey', label = r'Total creep, n = 3, d = 200 $\mu$ m$')
165:
166: plt.plot(tab['t1-3_esp'], tab['s'], color = 'grey', ls = '--', label = 'Total creep, n = 3, d = grainSizeComp table')
167: plt.plot(tab['t1-4_esp'], tab['s'], color = 'orange', ls = '--', label = 'Total creep, n = 3, d = subSeed1 table')
168: plt.plot(tab['t1-5_esp'], tab['s'], color = 'teal', ls = '--', label = 'Total creep, n = 3, d = subSeed2 table')
169:
170: for i in mylist:
171:     plt.scatter(tab.loc[i, 't1-3'], tab.loc[i, 's'], color = 'grey', alpha = 0.1)
172:     plt.scatter(tab.loc[i, 't1-4'], tab.loc[i, 's'], color = 'orange', alpha = 0.1)
173:     plt.scatter(tab.loc[i, 't1-5'], tab.loc[i, 's'], color = 'teal', alpha = 0.1)
174:
175:
176: plt.legend()
177: plt.xlabel(r'Log(Strain rate ($s^{-1}$))')
178: plt.ylabel('Log(Stress (MPa))')
179: plt.xscale('log')
180: plt.yscale('log')
181: plt.show()
182: plt.savefig('creep_grainSizeComp-esperance.png')
183:
184:
185: # In[11]:
186:
187:
188: T = 1473
189: plt.rcParams["font.family"] = "serif"
190: plt.figure(figsize=(10, 8), dpi=300)
191: plt.scatter(tab['t1-3_percent'], tab['t1-3'], c = tab['s'], cmap = 'bone')
192: plt.yscale('log')
193: cb = plt.colorbar()
194: cb.set_label(label='stress (MPa)')
195: plt.ylabel(r'strain rate ($s^{-1}$)')
196: plt.xlabel(f'percentage of dislocation creep on total strain, T = {T}K')
197: plt.savefig('creep_grainSizeComp-percent.png')
198:
199:
200: # In[12]:
201:
202:
203: T = 1473
204: plt.rcParams["font.family"] = "serif"
205: plt.figure(figsize=(10, 8), dpi=300)
206: plt.scatter(tab['t1-4_percent'], tab['t1-4'], c = tab['s'], cmap = 'bone')
207: plt.yscale('log')
208: cb = plt.colorbar()
209: cb.set_label(label='stress (MPa)')
210: plt.ylabel(r'strain rate ($s^{-1}$)')
211: plt.xlabel(f'percentage of dislocation creep on total strain, T = {T}K')
212: plt.savefig('creep_subSeed1-percent.png')
213:
214:
215: # In[13]:
216:
217:
218: T = 1473

```

```
219: plt.rcParams["font.family"] = "serif"
220: plt.figure(figsize=(10, 8), dpi=300)
221: plt.scatter(tab['t1-5_percent'], tab['t1-5'], c = tab['s'], cmap = 'bone')
222: plt.yscale('log')
223: cb = plt.colorbar()
224: cb.set_label(label='stress (MPa)')
225: plt.ylabel(r'strain rate ( $s^{-1}$ )')
226: plt.xlabel(f'percentage of dislocation creep on total strain, T = {T}K')
227: plt.savefig('creep_subSeed2-percent.png')
228:
229:
230: # In[ ]:
231:
232:
233:
234:
```

```

1: class CreepLaw():
2:     def __init__(self):
3:
4:         self.R = 8.3145
5:         self.e = 2.718281828459045
6:
7:     def _convert(self, s, var):
8:         '''
9:         Convert the variable array to the same as stress
10:        '''
11:        self.s = s
12:        self.T = T
13:        if type(var) == 'float':
14:            self.var = np.full((len(s), ), var)
15:        else:
16:            self.var = var
17:
18:    def Hirth2003(self, s, var, T, param):
19:        '''
20:        Creep law associated to the review of G. Hirth and D. Kohlstedt, 2003.
21:
22:        Arguments :
23:            s      stress                [MPa]
24:            var    variable value (d)   [1/μm]
25:            T      temperature           [Kelvin]
26:
27:        Output :
28:            S      strain rate           [s-1]
29:        '''
30:
31:        self._convert(s, var)
32:        self.S = param.A * self.s**param.n * (1/self.var)**param.p * param.fH2O**param.r * np.exp(p
aram.alpha*param.phi) * np.exp( -param.Q /(self.R * T))
33:        return self.S
34:
35:    def Tommasi2021(self, s, var, T, param):
36:        '''
37:        Creep law associated to disocation creep.
38:
39:        Arguments :
40:            s      stress                [MPa]
41:            var    variable value (d)   [1/μm]
42:            T      temperature           [Kelvin]
43:
44:        Output :
45:            S      strain rate           [s-1]
46:        '''
47:
48:        self._convert(s, var)
49:        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)
50:        return self.S
51:
52:    def Demouchy2013(self, s, var, T, param):
53:        '''
54:        Creep law semi-empirical determined on dro olivine experiments by S. Demouchy et al, 2013.
55:
56:        Arguments :
57:            s      stress                [MPa]
58:            var    variable value (d)   [1/μm]
59:            T      temperature           [Kelvin]
60:
61:        Output :
62:            S      strain rate           [s-1]
63:        '''
64:
65:        self._convert(s, var)
66:        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)
67:        return self.S
68:
69:    def Creep(self, first, *args):
70:        '''
71:        Sum values of strain rate calculated by creep laws, calculate percentage part of the first a
rgument

```

```

72:
73:     Argument :
74:
75:
76:     Output :
77:         St      strain rate total      [s-1]
78:         perct   percentage of the 1st   0 < perct < 1
79:         array
80:     '''
81:
82:     self.St = first
83:     for a in args:
84:         self.St += a
85:
86:     self.perct = (self.St - first) / self.St
87:     return self.St, self.perct
88:
89:
90: def CREEP(self, s, var, T, param, phi, conditions = 'all'):
91:
92:     if conditions == 'all':
93:         conditions = p.param.index
94:
95:     plt.rcParams["font.family"] = "serif"
96:     plt.figure(figsize=(10, 8), dpi=300)
97:
98:     self.Tommasi2021(s, var, T, param)
99:     plt.plot(self.S, self.s, label = 'dislocation Tommasi')
100:
101:     for cond in conditions:
102:         param.SetCondition(cond, T, phi)
103:         self.Hirth2003(s, var, T, param)
104:         plt.plot(self.S, self.s, label = cond)
105:
106:     plt.legend()
107:     plt.xlabel('Log(Strain rate (s-1))')
108:     plt.ylabel('Log(Stress (MPa))')
109:     plt.xscale('log')
110:     plt.yscale('log')
111:     plt.show()
112:
113: #for i in range(1,nStress):
114: #     stress = st.loc[i, 'stress']
115: #print(stress)
116: #     strainStoch = 0
117: #     for j in range(1, nSample+1):
118: #         varStoch = varMinDiff
119: #         while (varStoch >= varMaxDiff) or (varStoch <= varMinDiff) :
120: #             #print(varStoch)
121: #             varStoch = varMeanDiff + np.random.standard_normal() * varStdDiff
122: #             #print(varStoch)
123: #             misfit = creepDiffusion(stress, varStoch, T) - strainStoch
124: #             strainStoch = strainStoch + misfit / j
125: #     st.loc[i, 'strainStochDiffusion'] = strainStoch

```



```
1: from laws import *
2:
3: #decorators
4:
5:
6: #percent
7:
8:
9: #iterstrain
```

```
1:
2: def plotFit(x, y, dfs = True, allSamples = False):
3:
4:     if dfs == False :
5:         dfw = pd.read_csv('TUR_Olivine.txt', sep = ';')
6:
7:     if allSamples == True :
8:         d = dfs[dfs['indexSets'] == i]
9:
10:    d = d.sort_values(by = 'sse', inplace = True)
11:
12:    dist = getattr(stats, distribution)
13:    parameters = dist.fit(data)
14:    loc = parameters[-2]
15:    scale = parameters[-1]
16:    arg = parameters[:-2]
17:
18:
19:    pdf = dist.pdf(x, *arg, loc=loc, scale=scale)
20:
21:    plt.rcParams["font.family"] = "serif"
22:    plt.figure(figsize=(8, 5), dpi=300)
23:    plt.plot(x, y, label="Donnaes", color = 'red')
24:    plt.plot(x, pdf, label=f'{distribution}', linewidth=1)
25:    plt.legend(loc='upper right')
26:    plt.show()
```

```
1: import numpy as np
2: import pandas as pd
3:
4: def GenerateVariable(Min, Max, ds, mode = 'range'):
5:
6:     if mode == 'range':
7:         var = np.arange(Min, Max, ds)
8:
9:     if mode == 'linearSpace':
10:         var = np.linspace(Min, Max, num = ds)
11:
12:     if mode == 'uniform':
13:         var = Min + np.random.uniform(0, 1, Nb) * (Max - Min)
14:         var = np.sort(var)
15:
16:     return var
17:
18:
19: def stochasticSeed(Min, Max, Std, NbDraw, NbVal):
20:
21:     df = pd.DataFrame()
22:
23:     for i in range(0, NbVal):
24:
25:         for j in range(0, NbDraw):
26:
27:             varStoch = 0
28:
29:             while (varStoch >= Max) or (varStoch <= Min):
30:                 varStoch = Min + np.random.standard_normal() * Std
31:                 df.loc[i, j] = varStoch
32:
33:     return df
34:
35:
36: def statisticSeed(df,
37:                  calculation = ['min', 'max', 'std', 'mean', 'median', 'mad'],
38:                  quantile = [0.2, 0.4, 0.6, 0.8],
39:                  option = 1):
40:
41:
42:     for element in calculation:
43:
44:         if element == 'min':
45:             df[element] = df.min(axis = option)
46:
47:         if element == 'mean':
48:             df[element] = df.mean(axis = option)
49:
50:         if element == 'max':
51:             df[element] = df.max(axis = option)
52:
53:         if element == 'var':
54:             df[element] = df.var(axis = option)
55:
56:         if element == 'std':
57:             df[element] = df.std(axis = option)
58:
59:         if element == 'median':
60:             df[element] = df.median(axis = option)
61:
62:         if element == 'mad':
63:             df[element] = df.mad(axis = option)
64:
65:
66:     for q in quantile:
67:         df[f'q{q}'] = df.quantile(q, axis = option)
68:     return df
```

```
1: def loadF(filename):
2:
3:     file = open(filename, 'r')
4:     fixedDict = {}
5:     stoDict = {}
6:
7:     for line in file:
8:         if not line.startswith('###') and 'sto' not in line and 'nb' not in line:
9:             k, v = line.strip().split('=')
10:            try:
11:                fixedDict[k.strip()] = float(v.strip())
12:            except ValueError:
13:                fixedDict[k.strip()] = v.strip()
14:
15:        if line.startswith('sto') or 'nb' in line:
16:            k, v = line.strip().split('=')
17:            try:
18:                stoDict[k.strip()] = float(v.strip())
19:            except ValueError:
20:                stoDict[k.strip()] = v.strip()
21:    file.close()
22:
23:    return fixedDict, stoDict
```

```
1: import numpy as np
2: import load
3:
4: class Fixed():
5:     def __init__(self, Dict):
6:
7:         self.attribute = []
8:
9:         for k, v in Dict.items():
10:             setattr(self, k, v)
11:
12:     def GenerateVariable(self, mode = 'uniform', **kwargs):
13:
14:         if len(kwargs) == 0:
15:             Min = self.Min
16:             Max = self.Max
17:             ds = self.ds
18:             name = self.name
19:
20:         else:
21:             Min = kwargs['Min']
22:             Max = kwargs['Max']
23:             ds = kwargs['ds']
24:             name = kwargs['name']
25:
26:         if mode == 'range':
27:             var = np.arange(Min, Max, ds)
28:
29:         if mode == 'linearSpace':
30:             var = np.linspace(Min, Max, num = ds)
31:
32:         if mode == 'uniform':
33:             ds = int(ds)
34:             var = Min + np.random.uniform(0, 1, ds) * (Max - Min)
35:             var = np.sort(var)
36:
37:         self.mode = mode
38:         self.attribute.append(name)
39:
40:         setattr(self, name, var)
41:
42:         txt = f'{name} :      {Min}   {Max}   {ds}   {mode}'
43:         setattr(self, f'm{name}', txt)
44:
45:
46:         print(f'Attribute "{name}" set.')
```

```

1: import numpy as np
2: import pandas as pd
3:
4:
5: class Stochastic():
6:     def __init__(self, Dict):
7:
8:         for k, v in Dict.items():
9:             setattr(self, k, v)
10:
11:         self.attribute = []
12:
13:     def stochasticSeed(self, **kwargs):
14:
15:         if len(kwargs) == 0:
16:             Min = self.stoMin
17:             Max = self.stoMax
18:             Std = self.stoStd
19:             NbStep = int(self.nbStep)
20:             NbDraw = int(self.nbDraw)
21:             name = self.stoName
22:
23:         else:
24:             Min = kwargs['Min']
25:             Max = kwargs['Max']
26:             Std = kwargs['Std']
27:             NbStep = int(kwargs['nbStep'])
28:             NbDraw = int(kwargs['nbDraw'])
29:             name = kwargs['name']
30:
31:         self.tab = np.zeros((NbStep, NbDraw))
32:
33:         for i in range(0, NbStep):
34:
35:             for j in range(0, NbDraw):
36:
37:                 varStoch = 0
38:
39:                 while (varStoch >= Max) or (varStoch <= Min):
40:                     varStoch = Min + np.random.standard_normal() * Std
41:                     self.tab[i, j] = varStoch
42:
43:         col = [str(x) for x in range(0, NbDraw)]
44:         self.tab = pd.DataFrame(self.tab, columns = col)
45:
46:         self.attribute.append(name)
47:
48:         setattr(self, name, self.tab)
49:
50:         txt = f'{name} :      {Min}  {Max}  {Std}  normal[{NbStep}, {NbDraw}]'
51:         setattr(self, f'm{name}', txt)
52:
53:         print(f'Stochastic seed "{name}" set')
54:
55:
56:     def statisticSeed(self,
57:                       calculation = ['min', 'max', 'std', 'mean', 'median', 'mad'],
58:                       quantile = [0.2, 0.4, 0.6, 0.8],
59:                       option = 1, **kwargs):
60:
61:         if len(kwargs) == 0:
62:             df = self.tab
63:             name = self.stoName
64:
65:         else:
66:             df = kwargs['df']
67:             name = kwargs['name']
68:
69:         for element in calculation:
70:
71:             if element == 'min':
72:                 df[element] = df.min(axis = option)
73:
74:             if element == 'mean':
75:                 df[element] = df.mean(axis = option)

```

```
76:
77:         if element == 'max':
78:             df[element] = df.max(axis = option)
79:
80:         if element == 'var':
81:             df[element] = df.var(axis = option)
82:
83:         if element == 'std':
84:             df[element] = df.std(axis = option)
85:
86:         if element == 'median':
87:             df[element] = df.median(axis = option)
88:
89:         if element == 'mad':
90:             df[element] = df.mad(axis = option)
91:
92:
93:     for q in quantile:
94:         df[f'q{q}'] = df.quantile(q, axis = option)
95:
96:
97:     setattr(self, f'{name}', df)
98:
99:     print(f'Characterisation of stochastic seed "{name}" made')
```

```

1: # functions
2:
3: def load(i, title = 'EGD', plot = False):
4:     df = pd.read_csv(sets[i], sep = ';')
5:     name = sets[i]
6:     # Array
7:     if title == 'EGD':
8:         data = df.equivalentRadius*2
9:     elif title == 'log':
10:         data = np.log10(df.equivalentRadius*2)
11:     elif title == 'areaPond':
12:         data = (df.equivalentRadius*2 * df.area)/0.7
13:
14:     # Histogramme des donnÃ©es
15:     bins='auto'
16:     y, x = np.histogram(data, bins=bins, density=True)
17:     # Milieu de chaque classe
18:     x = (x + np.roll(x, -1))[:-1] / 2.0
19:
20:     if plot == True:
21:
22:         plt.rcParams["font.family"] = "serif"
23:         plt.figure(figsize=(8, 5), dpi=300)
24:         n, bins, patches = plt.hist(data, bins=bins, density=True)
25:         plt.xlabel(r'$\mu$ m$')
26:         plt.title(f'{sets[i]}')
27:         plt.savefig(f'{name}_{title}.png', transparent = False)
28:         plt.show()
29:
30:     return data, x, y, name
31:
32:
33: def Characterise(data, x, y, name, distNames = stats._distr_params.distcont):
34:
35:
36:     y, x = np.histogram(data, bins='auto', density=True)
37:     # Milieu de chaque classe
38:     x = (x + np.roll(x, -1))[:-1] / 2.0
39:
40:     for distribution in distNames:
41:
42:         if distNames == stats._distr_params.distcont:
43:             distribution = distribution[0]
44:             print(distribution)
45:
46:             with warnings.catch_warnings(record=True) as w:
47:                 # Cause all warnings to always be triggered.
48:                 warnings.simplefilter("always")
49:                 w = 0
50:
51:                 if w == 0:
52:                     try:
53:                         dist = getattr(stats, distribution)
54:                         parameters = dist.fit(data)
55:                         loc = parameters[-2]
56:                         scale = parameters[-1]
57:                         arg = parameters[:-2]
58:
59:                         ##### Sum square error
60:                         pdf = dist.pdf(x, *arg, loc=loc, scale=scale)
61:                         sse = np.sum( (y - pdf)**2 )
62:
63:                         ##### Kolmogorov-Smirnov test for goodness of fit
64:                         p = stats.kstest(data, distribution, args = parameters)[1]
65:
66:                         param = "_".join([str(_) for _ in parameters])
67:                     except:
68:                         pass
69:
70:                     with open('OLIVINElog_.txt', 'a') as file:
71:                         file.write(f'{name};{distribution};{param[0]};{param[1]};{sse};{p}\n')
72:                         print('done')
73:                 elif w == 1:
74:                     pass
75:                 print('RunTime Warning : bad distribution')

```



```
76:
77:     del w
78:
79:     loc = parameters[-2]
80:     scale = parameters[-1]
81:     arg = parameters[:-2]
82:
83:
84:     pdf = dist.pdf(x, *arg, loc=loc, scale=scale)
85:
86:     plt.rcParams["font.family"] = "serif"
87:     plt.figure(figsize=(8, 5), dpi=300)
88:     plt.plot(x, y, label="Data", color = 'red')
89:     plt.plot(x, pdf, label=f'{{distribution}}', linewidth=1)
90:     plt.legend(loc='upper right')
91:     plt.show()
92:     plt.savefig(f'{{name}}_{{distribution}}.png')
93:
94:     return x, y, pdf, parameters
95:
96:
97: def fitter(data, sets, i):
98:     f = Fitter(data)
99:     f.fit()
100:    f.summary()
101:    plt.savefig(f'{{sets[i]}}.png', transparent = False)
102:
```

```

1: #!/usr/bin/env python3
2: # -*- coding: utf-8 -*-
3: """
4: Created on Wed May 5 11:26:06 2021
5:
6: @author: antoinemaitre
7: """
8:
9: import numpy as np
10: import matplotlib.pyplot as plt
11:
12:
13: """Construction du profil rhÃ©ologique du gabbro du Queyras"""
14:
15:
16: 'Gradient gÃ©othermique'
17:
18: z = np.arange(0,100,0.1)          #Profondeur (km)
19: q = 15*10**(-3)                  #Chaleur (J = N.m)
20: T = (q*(z*10**3))+273            #TempÃ©rature (K)
21:
22:
23:
24: """Comportement fragile - Byerlee"""
25:
26: 'Pression lithostatique'
27:
28: rho = 3300                        #Masse volumique (kg/m3)
29:
30: g = 9.81                          #AccÃ©lÃ©ration de pesanteur (m/s2)
31: Plith = rho*g*(z*10**3)          #Pression lithostatique (Pa)
32: mu = 0.6                          #Coefficient de friction (/)
33: pf=0.9                            #Pression fluide (comprise entre 0 et 1)
34:
35: taub = (mu*Plith*(1-pf))*10**(-6) #Contrainte dÃ©viatorique (MPa)
36:
37:
38:
39: """Comportement ductile - fluage dislocation"""
40:
41: 'Albite'
42:
43: n = 3                            #Exposant de contrainte (/)
44: logA1 = 3.4
45: A1 = 10**logA1                  #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
46: #fH2O= 0.2                      #Pourcentage d'H2O (wt%)
47: Q = 332*10**3                   #Ã©nergie d'activation (J.mol^(-1))
48: R = 8.314                       #Constante des gaz parfaits ((kPa.L)/(mol.K))
49: strainrate = 10**(-14)          #Taux de dÃ©formation (s-1)
50:
51: taud1 = (strainrate/A1)**(1/n)*np.exp(Q/(R*T*n)) #Contrainte dÃ©viatorique (MPa)
52:
53:
54: # 'Anorthite 1'
55:
56: # n = 3                          #Exposant de contrainte (/)
57: # logA2 = 12.7
58: # A2 = 10**logA2                 #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
59: # #fH2O= 0.2                     #Pourcentage d'H2O (wt%)
60: # Q = 648*10**3                  #Ã©nergie d'activation (J.mol^(-1))
61: # R = 8.314                      #Constante des gaz parfaits ((kPa.L)/(mol.K))
62: # strainrate = 10**(-14)         #Taux de dÃ©formation (s-1)
63:
64: # taud2 = (strainrate/A2)**(1/n)*np.exp(Q/(R*T*n)) #Contrainte dÃ©viatorique (MPa)
65:
66: # 'Anorthite 2'
67:
68: # n = 3                          #Exposant de contrainte (/)
69: # logA3 = 2.6
70: # A3 = 10**logA3                 #Constante diffÃ©rente chaque phase minÃ©rale (MPa)
71: # #fH2O= 0.2                     #Pourcentage d'H2O (wt%)
72: # Q = 356*10**3                  #Ã©nergie d'activation (J.mol^(-1))
73: # R = 8.314                      #Constante des gaz parfaits ((kPa.L)/(mol.K))
74: # strainrate = 10**(-14)         #Taux de dÃ©formation (s-1)
75:

```

```

76: # taud3 = (strainrate/A3)**(1/n)*np.exp(Q/(R*T*n))      #Contrainte dÃ©viatorique (MPa)
77:
78:
79: 'ClinopyroxÃ©ne (diopside wet)'
80:
81: n = 5.5                      #Exposant de contrainte (/)
82: logA4 = 0.8
83: A4 = 10**logA4              #Constante diffÃ©rentielle chaque phase minÃ©rale (MPa)
84: #fH2O= 0.2                  #Pourcentage d'H2O (wt%)
85: Q = 534*10**3               #Ã©nergie d'activation (J.mol-1)
86: R = 8.314                   #Constante des gaz parfaits ((kPa.L)/(mol.K))
87: strainrate = 10**(-14)      #Taux de dÃ©formation (s-1)
88:
89: taud4 = (strainrate/A4)**(1/n)*np.exp(Q/(R*T*n))      #Contrainte dÃ©viatorique (MPa)
90:
91:
92: 'Amphibole'
93:
94: #n = 3
95: #logA5 =
96: #A5 = 10**logA5
97: #fH2O= 0.2
98: #Q =
99: #R = 8.314
100: #strainrate = 10**(-14)      #Taux de dÃ©formation (s-1)
101:
102: #taud5 = (strainrate/A5)**(1/n)*np.exp(Q/(R*T*n))      #Contrainte dÃ©viatorique (MPa)
103:
104:
105: ""ReprÃ©sentation""
106:
107:
108: 'Profil rhÃ©ologique (Loi de fluage - Profondeur)'
109:
110: plt.figure(1)
111:
112: #Loi de Byerlee :
113:
114: #plt.plot(taub,-z,'black', label='Loi de Byerlee', linewidth = 2)
115:
116: #Loi de fluage : Albite - Arnothite 1 - Anorthite 2 - ClinopyroxÃ©ne :
117:
118: #plt.plot(taud1,-z,'-g',label='Albite wet', linewidth = 2)
119: #plt.plot(taud2,-z,'--b',label='Anorthite 1',linewidth = 1)
120: #plt.plot(taud3,-z,'-g',label='Anorthite 2',linewidth = 1)
121: #plt.plot(taud4,-z,'-r',label='ClinopyroxÃ©ne',linewidth = 2)
122: #plt.plot(T-273, -z,'b')
123:
124: plt.legend(loc = 'lower right')
125: plt.grid()
126: plt.xlim(0,200)
127: plt.ylim(-100,0)
128: plt.xlabel('Contrainte dÃ©viatorique (MPa)') ; plt.ylabel('Profondeur (km)')
129: plt.title("Profil rhÃ©ologique")
130:
131:
132: 'GÃ©otherme'
133:
134: plt.figure(2)
135:
136: plt.plot(T-273, -z,'r')
137:
138: plt.xlabel('TempÃ©rature (Ã©C)') ; plt.ylabel('Profondeur (km)')
139: plt.title("GÃ©otherme")
140: plt.grid()
141: plt.legend()
142:
143:
144: 'Loi de fluage'
145:
146: plt.figure(3)
147:
148: plt.plot(T-273,taud1,'-y',label='Albite wet')
149: #plt.plot(T-273,taud2,'--b',label='Anorthite 1')
150: #plt.plot(T-273,taud3,'-g',label='Anorthite 2')

```

```
151: plt.plot(T-273,taud4,'-r',label='Clinopyrox@ne')
152:
153: plt.ylim(0,800)
154: plt.grid()
155: plt.legend(loc = 'upper right')
156: plt.xlabel('Temp@rature (@C)') ; plt.ylabel('Contrainte d@viatorique (MPa)')
157: plt.title("Lois de fluage")
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
```

```

1: import numpy as np
2: import pandas as pd
3: import os
4:
5: class Creep():
6:     def __init__(self, **kwargs):
7:
8:         self.__dict__.update(kwargs)
9:
10:        self.attribute = [] # attribute created
11:        self.configurations = []
12:
13:        self.R = 8.3145
14:        self.e = 2.718281828459045
15:
16:
17:        def SetCalibration(self, calib, process, condition, clear = []): # read the wanted datafra
me and the line associated to the right deformation process and the right condition
18:            df = pd.read_csv(f'/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/creeepy/calibration/{c
alib}.txt', sep = ';')
19:            df = df[df['process'] == process]
20:            df = df[df['condition'] == condition]
21:            df.reset_index(drop = True, inplace = True)
22:
23:            self.name = calib
24:            self.calibration = df
25:            self.process = process
26:            self.condition = condition
27:            self.listParam = self.calibration.columns
28:
29:            if len(clear) >= 1: # clean parameters b
etween two loadings
30:                for e in range(2, len(self.listParam)):
31:                    attr = self.listParam[e]
32:                    if hasattr(self, attr) and attr not in clear:
33:                        delattr(self, attr)
34:
35:
36:        def SetParams(self, fixed, stochastic):
37:            self.txt = f'Paramters used in the law {self.name}, for {self.condition} {self.process} : \n
,
38:
39:            for e in self.listParam:
40:                if 'process' not in e and 'condition' not in e:
41:                    if hasattr(fixed, e):
42:                        setattr(self, e, getattr(fixed, e))
43:                        self.txt = self.txt + f'{e} = see tables fixed \n'
44:
45:                    elif hasattr(stochastic, e):
46:                        setattr(self, e, getattr(stochastic, e))
47:                        self.txt = self.txt + f'{e} = see tables stochastic \n'
48:
49:                    elif hasattr(self, e):
50:                        self.txt = self.txt + f'{e} = {getattr(self, e)} set manually \n'
51:
52:                    else:
53:                        var = float(self.calibration.loc[0, e])
54:                        setattr(self, e, var)
55:                        self.txt = self.txt + f'{e} = {getattr(self, e)} from calibration \n'
56:
57:            self.varSto = stochastic.variables
58:            self.varFix = fixed.variables
59:
60:            self.txt = self.txt + f'\nFixed array : {"", ".join(self.varFix)}'
61:            self.txt = self.txt + f'\nStochastic array : {"", ".join(self.varSto)}\n\n'
62:
63:            self.configurations.append(self.txt)
64:
65:
66:        def SetVariables(self, variable, values):
67:            setattr(self, variable, values)
68:
69:
70:        def StochasticCreep(self, tab, variable, func, prop = 0):
71:

```

```

72:         self.stoS = pd.DataFrame()
73:
74:         for col in range(0, tab.shape[1]):
75:
76:             var = tab.iloc[:,col]
77:             self.SetVariables(variable, var)
78:             func()
79:             self.stoS[tab.columns[col]] = self.S
80:
81:         columns = self.stoS.columns
82:         self.stoS['meanSeed'] = self.stoS.mean(axis = 1)
83:
84:         if prop >= 0:
85:             for e in range(1, prop+1):
86:                 col = b = [x for x in columns if x.startswith(f'{e}_')]
87:                 self.stoS[f'meanSubSeed{e}'] = self.stoS.loc[:, col].mean(axis = 1)
88:
89:
90:     def RegularCreep(self, func):
91:         if hasattr(self, 'rS') == False:
92:             self.rS = pd.DataFrame()
93:             self.rS['s'] = self.s
94:             func()
95:             self.rS[f'r_{self.name}'] = self.S
96:
97:
98:     def SetPercentage(self, df, col1, total):
99:         df[f'{total}_percent'] = df[col1]/df[total]
100:
101:
102:     def Esperance(self, tab, col):
103:
104:         for j in range(1, len(tab.index)):
105:             tab.loc[j, f'{col}_esp'] = sum(tab.loc[0:j, col])/(j)
106:         return tab
107:
108:     def TotalCreep(self, a, b, prop = 0):
109:
110:         df = self.rS.copy()
111:         df[f'sto_{self.name}'] = self.stoS['meanSeed']
112:
113:         if prop >= 0:
114:             for e in range(1, prop+1):
115:                 df[f'sto_{self.name}_SubSeed{e}'] = self.stoS[f'meanSubSeed{e}']
116:
117:
118:         columns = list(df.columns)
119:
120:         for i in range(0, len(a)):
121:             ind1 = columns.index(a[i])
122:             ind2 = columns.index(b[i])
123:             df[f't{ind1}-{ind2}'] = df.iloc[:,ind1] + df.iloc[:,ind2]
124:
125:             self.SetPercentage(df, columns[ind1], f't{ind1}-{ind2}')
126:             df = self.Esperance(df, f't{ind1}-{ind2}')
127:
128:         var = '-'.join(self.varSto)
129:         name = f'S_{var}_{self.T}'
130:         self.attribute.append(name)
131:         setattr(self, name, df)
132:
133:
134:
135:
136: #####
#
137:
138:     def Hirth2003(self):
139:         '''
140:         Creep law associated to the review of G. Hirth and D. Kohlstedt, 2003.
141:
142:         Arguments :
143:             s          stress          [MPa]
144:             var        variable value (d) [1/4m]
145:             T          temperature      [Kelvin]

```

```

146:
147:     Output :
148:         S         strain rate         [s-1]
149:     '''
150:
151:     self.S = self.A * self.s**self.n * (1/self.d)**self.p * self.fH2O**self.r * np.exp(self.alp
ha*self.phi) * np.exp( -self.Q /(self.R * self.T))
152:
153:
154:     def Gouriet2018(self):
155:         '''
156:         Creep law associated to disocation creep.
157:
158:         Arguments :
159:             s         stress         [MPa]
160:             var       variable value (d) [ $\hat{\Gamma}^{1/4}$ m]
161:             T         temperature      [Kelvin]
162:
163:         Output :
164:             S         strain rate         [s-1]
165:         '''
166:
167:         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)
168:
169:
170:
171:
172:
173:
174:
175:
176:

```

```

1: import numpy as np
2: import pandas as pd
3:
4: class Data():
5:     def __init__(self, filename):
6:
7:         file = open(filename, 'r')                                # open
config file
8:
9:         self.fixedDict = {}                                       # create
dictionaries for both fixed and stochastic parameters
10:         self.stoDict = {}
11:
12:         self.variables = []
13:
14:         for line in file:
15:             if not line.startswith('###') and 'sto' not in line and 'nb' not in line: # lines w
ich begin by 'sto' or 'nb' are affected to the stochastic dictionary
16:                 k, v = line.strip().split('=')
17:                 try:
18:                     self.fixedDict[k.strip()] = float(v.strip()) # converte
rt numbers to float for further calculations...
19:                 except ValueError:
20:                     self.fixedDict[k.strip()] = v.strip()
21:
22:             if line.startswith('sto') or 'nb' in line:
23:                 k, v = line.strip().split('=')
24:                 try:
25:                     self.stoDict[k.strip()] = float(v.strip())
26:                 except ValueError:
27:                     self.stoDict[k.strip()] = v.strip()
28:         file.close()
29:
30:     def SetVariable(self, var, newVar, clear = False):
31:         if clear == True:
32:             self.variables = []
33:             # duplicate an attribute under another name
34:             setattr(self, var, getattr(self, newVar))
35:             self.variables.append(f'{var}_{newVar}')
36:
37:
38:
39:
40: class Fixed(Data):
41:     def __init__(self, filename = 'config'):
42:         super().__init__(filename)
43:
44:         self.attribute = []                                       # keep a
trace of the name of the variables generated for output
45:
46:         for k, v in self.fixedDict.items():                       # updat
e the class attribute with dictionary keys
47:             setattr(self, k, v)
48:
49:     def GenerateVariable(self, mode = 'uniform', **kwargs):       # genera
te variables by three different manners ('uniform' or 'linearSpace' better, to make the number of stress va
lues correspond to the number of points wanted)
50:
51:         if len(kwargs) == 0:                                     # take t
he attributes of the class
52:             Min = self.Min
53:             Max = self.Max
54:             ds = self.ds
55:             name = self.name
56:
57:         else:                                                     # take t
he kwargs (possibility to create other variables manually if all the wanted kwargs are declared)
58:             Min = kwargs['Min']
59:             Max = kwargs['Max']
60:             ds = kwargs['ds']
61:             name = kwargs['name']
62:
63:         if mode == 'range':
64:             var = np.arange(Min, Max, ds)
65:

```



```

66:         if mode == 'linearSpace':
67:             var = np.linspace(Min, Max, num = ds)
68:
69:         if mode == 'uniform':
70:             ds = int(ds)
71:             var = Min + np.random.uniform(0, 1, ds) * (Max - Min)
72:             var = np.sort(var)
73:
74:         self.mode = mode
75:         self.attribute.append(name)
76:
77:         setattr(self, name, var)
78:
79:         txt = f'{name} :      {Min}   {Max}   {ds}   {mode} \n'
80:
81:         tname = f'gen_{name}'                                     # create th
e txt for output
82:         setattr(self, tname, txt)
83:
84:         print(f'Attribute "{name}" set.')
85:
86:
87: class Stochastic(Data):
88:     def __init__(self, filename = 'config'):
89:         super().__init__(filename)
90:
91:         for k, v in self.stoDict.items():
92:             setattr(self, k, v)
93:
94:         self.attribute = []
95:
96:         def GenerateSeed(self, **kwargs):                         # same principle as
Fixed()
97:
98:         if len(kwargs) == 0:
99:             Min = self.stoMin
100:             Max = self.stoMax
101:             Std = self.stoStd
102:             NbStep = int(self.nbStep)
103:             NbDraw = int(self.nbDraw)
104:             name = self.stoName
105:
106:         else:
107:             Min = kwargs['Min']
108:             Max = kwargs['Max']
109:             Std = kwargs['Std']
110:             NbStep = int(kwargs['nbStep'])
111:             NbDraw = int(kwargs['nbDraw'])
112:             name = kwargs['name']
113:
114:         self.tab = np.zeros((NbStep, NbDraw))
115:
116:         for i in range(0, NbStep):
117:
118:             for j in range(0, NbDraw):
119:
120:                 varStoch = 0
121:
122:                 while (varStoch >= Max) or (varStoch <= Min):      # affect the r
andom value to the numpy 2D array
123:                     varStoch = Min + np.random.standard_normal() * Std
124:                     self.tab[i, j] = varStoch
125:
126:             col = [str(x) for x in range(0, NbDraw)]
127:             self.tab = pd.DataFrame(self.tab, columns = col)      # convert info
dataframe
128:
129:         self.attribute.append(name)
130:
131:         setattr(self, name, self.tab)
132:
133:         txt = f'{name} :      {Min}   {Max}   {Std}   normal[{NbStep}, {NbDraw}] \n'
134:
135:         tname = f'gen_{name}'
136:         setattr(self, tname, txt)

```

```

137:
138:         print(f'Stochastic seed "{name}" set')
139:
140:
141:     def GenerateCompositeSeed(self):                                     # experimental
, no possibility to generate manually
142:
143:         tab = pd.DataFrame()
144:         p = ''
145:
146:         for e in range(1, int(self.stoDistrib) + 1):                     # for the good
number of wanted distribution
147:
148:             prop = f'stoProp{e}'
149:             snbdraw = self.nbDraw * getattr(self, prop)                 # proportional
ly take the good number of random value to draw from nbDraw and the percentage of the distirbution
150:
151:             smin = str(f'stoMin{e}')                                     # update the d
istributu numbers
152:             smax = str(f'stoMax{e}')
153:             sstd = str(f'stoStd{e}')
154:             sname = str(f'stoName{e}')
155:
156:             self.GenerateSeed(Min = getattr(self, smin),                # generate a s
ub seed manually
157:                               Max = getattr(self, smax),
158:                               Std = getattr(self, sstd),
159:                               nbDraw = snbdraw,
160:                               nbStep = self.nbStep,
161:                               name = getattr(self, sname))
162:
163:             self.tab.columns = [f'{e}_{col}' for col in self.tab.columns] # update the c
olumns number with the number of the distribution : e = number of the distribution, col = number of the va
lue
164:             tab = pd.concat([tab, self.tab], axis = 1)                  # concat to t
he upper seed
165:
166:             p = p + f'          {getattr(self, sname)}, {getattr(self, prop)}\n' # create a su
b txt with the name and the proportion of the sub seeds
167:
168:
169:             self.attribute.append(self.stoNameComp)                     # add to the
created attribute the upper seed name
170:
171:             setattr(self, self.stoNameComp, tab)
172:
173:             txt = f'{self.stoNameComp} : \n' + p                        # create a tx
t wich gather the upper seed name, the sub seeds name and their proportion for output
174:
175:             tname = f'comp_{self.stoNameComp}'
176:             setattr(self, tname, txt)
177:
178:             print(f'    ---> added to the stochastic seed {self.stoNameComp} \n')
179:
180:     def CharacteriseSeed(self,                                           # can charact
erise the values of the values by lines..., possibility to juste make few calculation and not all with kwa
rgs...
181:         calculation = ['min', 'max', 'std', 'mean', 'median', 'mad'],
182:         quantile = [0.2, 0.4, 0.6, 0.8],
183:         option = 1, **kwargs):
184:
185:         if len(kwargs) == 0:
186:             df = self.tab
187:             name = self.stoName
188:
189:         else:
190:             df = kwargs['df']
191:             name = kwargs['name']
192:
193:         for element in calculation:
194:
195:             if element == 'min':
196:                 df[element] = df.min(axis = option)
197:
198:             if element == 'mean':

```

```
199:         df[element] = df.mean(axis = option)
200:
201:         if element == 'max':
202:             df[element] = df.max(axis = option)
203:
204:         if element == 'var':
205:             df[element] = df.var(axis = option)
206:
207:         if element == 'std':
208:             df[element] = df.std(axis = option)
209:
210:         if element == 'median':
211:             df[element] = df.median(axis = option)
212:
213:         if element == 'mad':
214:             df[element] = df.mad(axis = option)
215:
216:
217:     for q in quantile:
218:         df[f'q{q}'] = df.quantile(q, axis = option)
219:
220:
221:     setattr(self, f'{name}', df)
222:
223:     txt = f'Characterised by : \n parameters {calculation}, quantiles {quantile} on {option}'
224:     tname = f'cha_{name}'
225:     setattr(self, tname, txt)
226:
227:     print(f'Characterisation of stochastic seed "{name}" made')
```

culations names in output # keep cal

```

1: import random
2: import pandas as pd
3: import matplotlib.pyplot as plt
4:
5:
6: class Plot():
7:     def __init__(self):
8:
9:         self.plot = 0
10:
11:     def ParamPlot(self, width = 10, height = 8, dpi = 300):
12:         plt.rcParams["font.family"] = "serif"
13:         plt.figure(figsize=(10, 8), dpi=300)
14:
15:
16:     def PlotsS(self, tab, a, color, label, esp = 0, nb = 100, Max = 1000):
17:
18:         self.ParamPlot()
19:
20:         columns = list(tab.columns)
21:         ind1 = columns.index(a[0])
22:         ind2 = columns.index(a[1])
23:
24:         col = f't{ind1}-{ind2}'
25:
26:         plt.plot(tab[col], tab['s'], color = color, label = label)
27:
28:         if esp == 'esp':
29:             plt.plot(tab['s'], tab[f'{col}_esp'], color = color, ls = '--', label = label)
30:         elif esp == 'points':
31:             plt.scatter(tab.iloc[i, 's'], tab.iloc[i, f'{col}'], color = color, marker = 'o', alpha
= 0.5, label = label)
32:
33:         elif esp == 'esppoint':
34:             plt.plot(tab['s'], tab[f'{col}_esp'], color = color, ls = '--', label = label)
35:             plt.scatter(tab.iloc[0:nb, 's'], tab.iloc[i, f'{col}'], color = color, marker = 'o', alp
ha = 0.5, label = label)
36:
37:         plt.legend()
38:         plt.xlabel(r'Log(Strain rate (s-1))')
39:         plt.ylabel('Log(Stress (MPa))' )
40:         plt.xscale('log')
41:         plt.yscale('log')
42:
43:     def Save(self, title):
44:         plt.savefig(title)

```

```

1: import matplotlib.pyplot as plt
2: from datetime import datetime
3:
4:
5: class Output():
6:     def __init__(self):
7:
8:         self.date = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
9:
10:    def Describe(self, title, **kwargs):
11:
12:        name = f'EXP_{title}'
13:
14:        file = open(f'{title}.txt', 'w')
15:
16:        file.write(f'{title}, {self.date} \n \n')
17:
18:
19:        for e in kwargs:
20:
21:            if e == 'fixed':
22:                fixed = kwargs[e]
23:                file.write(f'\nFIXED PARAMETERS      : \n
e \n')
24:
25:                for e in fixed.attribute:
26:                    gen = f'gen_{e}'
27:                    cha = f'cha_{e}'
28:                    if hasattr(fixed, gen):
29:                        file.write(getattr(fixed, gen))
30:                    if hasattr(fixed, cha):
31:                        file.write(getattr(fixed, cha))
32:
33:            if e == 'stochastic':
34:                stochastic = kwargs[e]
35:                file.write(f'\nSTOCHASTIC PARAMETERS      : \n
distribution[NbStep, NbDraw] \n')
36:
37:                for e in stochastic.attribute:
38:                    gen = f'gen_{e}'
39:                    cha = f'cha_{e}'
40:                    comp = f'comp_{e}'
41:                    if hasattr(stochastic, gen):
42:                        file.write(getattr(stochastic, gen))
43:                    if hasattr(stochastic, cha):
44:                        file.write(getattr(stochastic, cha))
45:                    if hasattr(stochastic, comp):
46:                        file.write(getattr(stochastic, comp))
47:
48:            if e == 'creep':
49:                creep = kwargs[e]
50:                file.write(f'\n \nCREEP LAW      : \n')
51:                config = getattr(creep, 'configurations')
52:                for element in config:
53:                    file.write(f'{element} \n \n ')
54:
55:    def SaveAttributes(self, obj, attribute = 'all'):
56:
57:        if attribute == 'all':
58:            attribute = getattr(obj, 'attribute')
59:
60:        for element in attribute:
61:            df = getattr(obj, element)
62:            df.to_csv(f'{element}.txt', sep = ';')
63:
64:
65:
66:
67:

```

```

1: # functions
2:
3: def load(i, title = 'EGD', plot = False):
4:     df = pd.read_csv(sets[i], sep = ';')
5:     name = sets[i]
6:     # Array
7:     if title == 'EGD':
8:         data = df.equivalentRadius*2
9:     elif title == 'log':
10:         data = np.log10(df.equivalentRadius*2)
11:     elif title == 'areaPond':
12:         data = (df.equivalentRadius*2 * df.area)/0.7
13:
14:     # Histogramme des donnÃ©es
15:     bins='auto'
16:     y, x = np.histogram(data, bins=bins, density=True)
17:     # Milieu de chaque classe
18:     x = (x + np.roll(x, -1))[:-1] / 2.0
19:
20:     if plot == True:
21:
22:         plt.rcParams["font.family"] = "serif"
23:         plt.figure(figsize=(8, 5), dpi=300)
24:         n, bins, patches = plt.hist(data, bins=bins, density=True)
25:         plt.xlabel(r'$\mu$ m$')
26:         plt.title(f'{sets[i]}')
27:         plt.savefig(f'{name}_{title}.png', transparent = False)
28:         plt.show()
29:
30:     return data, x, y, name
31:
32:
33: def Characterise(data, x, y, name, distNames = stats._distr_params.distcont):
34:
35:
36:     y, x = np.histogram(data, bins='auto', density=True)
37:     # Milieu de chaque classe
38:     x = (x + np.roll(x, -1))[:-1] / 2.0
39:
40:     for distribution in distNames:
41:
42:         if distNames == stats._distr_params.distcont:
43:             distribution = distribution[0]
44:             print(distribution)
45:
46:             with warnings.catch_warnings(record=True) as w:
47:                 # Cause all warnings to always be triggered.
48:                 warnings.simplefilter("always")
49:                 w = 0
50:
51:                 if w == 0:
52:                     try:
53:                         dist = getattr(stats, distribution)
54:                         parameters = dist.fit(data)
55:                         loc = parameters[-2]
56:                         scale = parameters[-1]
57:                         arg = parameters[:-2]
58:
59:                         ##### Sum square error
60:                         pdf = dist.pdf(x, *arg, loc=loc, scale=scale)
61:                         sse = np.sum( (y - pdf)**2 )
62:
63:                         ##### Kolmogorov-Smirnov test for goodness of fit
64:                         p = stats.kstest(data, distribution, args = parameters)[1]
65:
66:                         param = "_".join([str(_) for _ in parameters])
67:                     except:
68:                         pass
69:
70:                     with open('OLIVINElog_.txt', 'a') as file:
71:                         file.write(f'{name};{distribution};{param[0]};{param[1]};{sse};{p}\n')
72:                         print('done')
73:                 elif w == 1:
74:                     pass
75:                 print('RunTime Warning : bad distribution')

```

```
76:
77:     del w
78:
79:     loc = parameters[-2]
80:     scale = parameters[-1]
81:     arg = parameters[:-2]
82:
83:
84:     pdf = dist.pdf(x, *arg, loc=loc, scale=scale)
85:
86:     plt.rcParams["font.family"] = "serif"
87:     plt.figure(figsize=(8, 5), dpi=300)
88:     plt.plot(x, y, label="Data", color = 'red')
89:     plt.plot(x, pdf, label=f'{{distribution}}', linewidth=1)
90:     plt.legend(loc='upper right')
91:     plt.show()
92:     plt.savefig(f'{{name}}_{{distribution}}.png')
93:
94:     return x, y, pdf, parameters
95:
96:
97: def fitter(data, sets, i):
98:     f = Fitter(data)
99:     f.fit()
100:    f.summary()
101:    plt.savefig(f'{{sets[i]}}.png', transparent = False)
102:
```

```
1: #!/usr/bin/env python3
2: # -*- coding: utf-8 -*-
3: """
4: Created on Wed May 5 11:26:06 2021
5:
6: @author: antoinemaitre
7: """
8:
9: import numpy as np
10: import matplotlib.pyplot as plt
11:
12:
13: """Construction du profil rh  ologique du gabbro du Queyras"""
14:
15:
16: 'Gradient g  othermique'
17:
18: z = np.arange(0,100,0.1)          #Profondeur (km)
19: q = 15*10**(-3)                  #Chaleur (J = N.m)
20: T = (q*(z*10**3))+273            #Temp  rature (K)
21:
22:
23:
24: """Comportement fragile - Byerlee"""
25:
26: 'Pression lithostatique'
27:
28: rho = 3300                        #Masse volumique (kg/m3)
29:
30: g = 9.81                          #Acc  l  ration de pesanteur (m/s2)
31: Plith = rho*g*(z*10**3)          #Pression lithostatique (Pa)
32: mu = 0.6                          #Coefficient de friction (/)
33: pf=0.9                           #Pression fluide (comprise entre 0 et 1)
34:
35: taub = (mu*Plith*(1-pf))*10**(-6) #Contrainte d  viatorique (MPa)
36:
37:
38:
39: """Comportement ductile - fluage dislocation"""
40:
41: 'Albite'
42:
43: n = 3                            #Exposant de contrainte (/)
44: logA1 = 3.4
45: A1 = 10**logA1                   #Constante diff  rente chaque phase min  rale (MPa)
46: #fH2O= 0.2                       #Pourcentage d'H2O (wt%)
47: Q = 332*10**3                    #  nergie d'activation (J.mol^(-1))
48: R = 8.314                        #Constante des gaz parfaits ((kPa.L)/(mol.K))
49: strainrate = 10**(-14)           #Taux de d  formation (s-1)
50:
51: taud1 = (strainrate/A1)**(1/n)*np.exp(Q/(R*T*n)) #Contrainte d  viatorique (MPa)
52:
53:
54: # 'Anorthite 1'
55:
56: # n = 3                            #Exposant de contrainte (/)
57: # logA2 = 12.7
58: # A2 = 10**logA2                   #Constante diff  rente chaque phase min  rale (MPa)
59: # #fH2O= 0.2                       #Pourcentage d'H2O (wt%)
60: # Q = 648*10**3                    #  nergie d'activation (J.mol^(-1))
61: # R = 8.314                        #Constante des gaz parfaits ((kPa.L)/(mol.K))
62: # strainrate = 10**(-14)           #Taux de d  formation (s-1)
63:
64: # taud2 = (strainrate/A2)**(1/n)*np.exp(Q/(R*T*n)) #Contrainte d  viatorique (MPa)
65:
66: # 'Anorthite 2'
67:
68: # n = 3                            #Exposant de contrainte (/)
69: # logA3 = 2.6
70: # A3 = 10**logA3                   #Constante diff  rente chaque phase min  rale (MPa)
71: # #fH2O= 0.2                       #Pourcentage d'H2O (wt%)
72: # Q = 356*10**3                    #  nergie d'activation (J.mol^(-1))
73: # R = 8.314                        #Constante des gaz parfaits ((kPa.L)/(mol.K))
74: # strainrate = 10**(-14)           #Taux de d  formation (s-1)
75:
```



```

76: # taud3 = (strainrate/A3)**(1/n)*np.exp(Q/(R*T*n))      #Contrainte dÃ©viatorique (MPa)
77:
78:
79: 'ClinopyroxÃ©ne (diopside wet)'
80:
81: n = 5.5                      #Exposant de contrainte (/)
82: logA4 = 0.8
83: A4 = 10**logA4              #Constante diffÃ©rentielle chaque phase minÃ©rale (MPa)
84: #fH2O= 0.2                  #Pourcentage d'H2O (wt%)
85: Q = 534*10**3               #Ã©nergie d'activation (J.mol-1)
86: R = 8.314                   #Constante des gaz parfaits ((kPa.L)/(mol.K))
87: strainrate = 10**(-14)      #Taux de dÃ©formation (s-1)
88:
89: taud4 = (strainrate/A4)**(1/n)*np.exp(Q/(R*T*n))      #Contrainte dÃ©viatorique (MPa)
90:
91:
92: 'Amphibole'
93:
94: #n = 3
95: #logA5 =
96: #A5 = 10**logA5
97: #fH2O= 0.2
98: #Q =
99: #R = 8.314
100: #strainrate = 10**(-14)      #Taux de dÃ©formation (s-1)
101:
102: #taud5 = (strainrate/A5)**(1/n)*np.exp(Q/(R*T*n))      #Contrainte dÃ©viatorique (MPa)
103:
104:
105: ""ReprÃ©sentation""
106:
107:
108: 'Profil rhÃ©ologique (Loi de fluage - Profondeur)'
109:
110: plt.figure(1)
111:
112: #Loi de Byerlee :
113:
114: #plt.plot(taub,-z,'black', label='Loi de Byerlee', linewidth = 2)
115:
116: #Loi de fluage : Albite - Arnothite 1 - Anorthite 2 - ClinopyroxÃ©ne :
117:
118: #plt.plot(taud1,-z,'-g',label='Albite wet', linewidth = 2)
119: #plt.plot(taud2,-z,'--b',label='Anorthite 1',linewidth = 1)
120: #plt.plot(taud3,-z,'-g',label='Anorthite 2',linewidth = 1)
121: #plt.plot(taud4,-z,'-r',label='ClinopyroxÃ©ne',linewidth = 2)
122: #plt.plot(T-273, -z,'b')
123:
124: plt.legend(loc = 'lower right')
125: plt.grid()
126: plt.xlim(0,200)
127: plt.ylim(-100,0)
128: plt.xlabel('Contrainte dÃ©viatorique (MPa)') ; plt.ylabel('Profondeur (km)')
129: plt.title("Profil rhÃ©ologique")
130:
131:
132: 'GÃ©otherme'
133:
134: plt.figure(2)
135:
136: plt.plot(T-273, -z,'r')
137:
138: plt.xlabel('TempÃ©rature (Ã©C)') ; plt.ylabel('Profondeur (km)')
139: plt.title("GÃ©otherme")
140: plt.grid()
141: plt.legend()
142:
143:
144: 'Loi de fluage'
145:
146: plt.figure(3)
147:
148: plt.plot(T-273,taud1,'-y',label='Albite wet')
149: #plt.plot(T-273,taud2,'--b',label='Anorthite 1')
150: #plt.plot(T-273,taud3,'-g',label='Anorthite 2')

```

```
151: plt.plot(T-273,taud4,'-r',label='Clinopyrox@ne')
152:
153: plt.ylim(0,800)
154: plt.grid()
155: plt.legend(loc = 'upper right')
156: plt.xlabel('Temp@rature (@C)') ; plt.ylabel('Contrainte d@viatorique (MPa)')
157: plt.title("Lois de fluage")
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
```

```

1: #####
2: #           Makefile for Python scripts for Adeli #
3: #-----#
4:
5: #####
6: # Define exec name and obj directory:
7: #-----#
8:
9: PS2P := creeepy2p.ps
10: PDF2P := creeepy2p.pdf
11:
12: PS := creeepy.ps
13: PDF := creeepy.pdf
14:
15: #####
16: # Source and object files:
17: #-----#
18:
19: SRCS := `find ~/RhEoVOLUTION/CODES/SCRIPTS/creeepy -type f -name "*.py"`
20:
21: #####
22: # Make a pdf :
23:
24: SRCS += makefile
25:
26: write:
27:      enscript -2r --line-numbers --highlight=fortran --toc --fancy-header=header $(SRCS) -o $(PS2
P)
28:      gs -sDEVICE=pdfwrite -o $(PDF2P) $(PS2P)
29:
30: print:
31:      enscript -MA4 --line-numbers --highlight=fortran --toc --fancy-header=headerB5 -fCourier8.5
$(SRCS) -o $(PS)
32:      gs -sDEVICE=pdfwrite -o $(PDF) $(PS)
33:      #gs -q -sDEVICE=pdfwrite -dBATCH -dNOPAUSE -sOutputFile=print.pdf \
34:      -dDEVICEWIDTHPOINTS=595 -dDEVICEHEIGHTPOINTS=842 -dFIXEDMEDIA \ -c "<< /CurrPageNum 1 def /I
nsta11 { /CurrPageNum CurrPageNum 1 add def CurrPageNum 2 mod 1 eq {} {96 0 translate} ifelse } bind >> se
tpagedevice " -f "out.pdf"
35:      #rm out.pdf
36:
37:
38: #####
39: # Delete objects:
40: #-----#
41:
42: clean:
43:      /bin/rm $(DIROBJ)/*.o
44:
45: cleanmod:
46:      /bin/rm $(DIROBJ)/*.mod
47:
48: all:
49:      @echo $(OBSJS)

```

1	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/tests/run.py	2 pages	125 lines	22/04/19 03:16:10
2	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/tests/test19_04.py	4 pages	234 lines	22/04/19 14:16:01
3	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/backup/creep/laws.py	2 pages	125 lines	22/04/05 10:25:59
4	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/backup/creep/law.py	1 pages	8 lines	22/04/05 10:25:59
5	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/backup/plot/plot.py	1 pages	25 lines	22/04/05 10:25:59
6	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/backup/parameters/save.py	1 pages	68 lines	22/04/05 10:25:59
7	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/backup/parameters/load.py	1 pages	23 lines	22/04/05 10:25:59
8	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/backup/parameters/fixed.py	1 pages	46 lines	22/04/05 10:25:59
9	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/backup/parameters/stochastic.py	2 pages	99 lines	22/04/05 10:25:59
10	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/backup/distrib/characterise.py	2 pages	101 lines	22/04/05 10:25:59
11	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/backup/xmastree/Profils-rhÃ©o_phases-minÃ©rales-sÃ©parÃ©es.py	3 pages	173 lines	22/04/05 10:25:59
12	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/creeepy/creep.py	3 pages	175 lines	22/04/19 11:56:58
13	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/creeepy/parameters.py	4 pages	227 lines	22/04/19 00:21:11
14	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/creeepy/plot.py	1 pages	43 lines	22/04/19 01:47:03
15	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/creeepy/output.py	1 pages	66 lines	22/04/19 01:08:19
16	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/creeepy/characterise.py	2 pages	101 lines	22/03/30 15:45:28
17	/home/Marialine/RhEoVOLUTION/CODES/SCRIPTS/creeepy/creeepy/Profils-rhÃ©o_phases-minÃ©rales-sÃ©parÃ©es.py	3 pages	173 lines	22/03/22 23:04:20
18	makefile	1 pages	49 lines	22/04/19 16:11:40