

```

1 import context
2 import numpy as np
3 from cr507.utils import plt_set
4 import matplotlib.pyplot as plt
5 from collections import namedtuple
6
7
8 class Approximator:
9
10     #####
11     # initialize condtions
12     #####
13     def __init__(self, valueDict):
14         """
15         Create the grid and initial conditions
16         """
17         ## Defined conditions from dictionary
18         self.__dict__.update(valueDict)
19
20         ## Define number of time steps
21         nsteps = (self.gridx - 300) / (self.u0 * self.dt / self.dx)
22         nsteps = np.arange(0,nsteps)
23         self.nsteps = nsteps
24
25         ## Calculate the Courant number
26         cr = self.u0 * self.dt / self.dx
27         self.cr = cr
28
29         ## Create initial concentration anomaly
30         # distribution in the x-direction
31         conc = np.zeros(self.gridx)
32         conc[100:151] = np.linspace(0.,self.cmax,51)
33         conc[150:201] = np.linspace(self.cmax, 0.,51)
34         conc[20:41] = np.linspace(0., -0.5 * self.cmax, 21)
35         conc[40:61] = np.linspace(-0.5 * self.cmax, 0., 21)
36         self.Pj = np.array(conc)
37
38         ## Define the ideal exact final solution
39         cideal = np.zeros(self.gridx)
40         cideal[800:851] = np.linspace(0., self.cmax,51)
41         cideal[850:901] = np.linspace(self.cmax, 0., 51)
42         cideal[720:741] = np.linspace(0., -0.5 * self.cmax, 21)
43         cideal[740:761] = np.linspace(-0.5 * self.cmax, 0., 21)
44         self.cideal = np.array(cideal)
45
46         #####
47         # spatial discretization methods
48         #####
49         def centdif(self):
50             """
51             Centered difference spatial approximation
52             """
53             # print(self.Pj[50],"centdif start")
54             Pj = -self.u0 * ((np.roll(self.Pj,-1) - np.roll(self.Pj,1)) / (2 *
self.dx))
55
56             # print(Pj[50],"centdif end")
57             return Pj
58
59         def backdif(self):

```

```

60         """
61         Backward difference spatial approximation
62         """
63         # print(self.Pj[50], "backdif start")
64         Pj = -self.u0 * ((self.Pj - np.roll(self.Pj, 1)) / (self.dx))
65
66         # print(Pj[50], "backdif end")
67         return Pj
68
69         #####
70         # time discretization methods
71         #####
72     def forward(self):
73         Pj_0G = self.Pj
74
75         Pjn_1 = []
76         for n in range(len(self.nsteps)):
77             Pj = self.Pj
78             Pn = Pj + self.dt * self.backdif()
79             self.Pj = Pn
80
81             Pjn_1.append(Pn)
82
83         Pjn_1 = np.array(Pjn_1)
84         print(Pjn_1.shape)
85         self.Pj = Pj_0G
86
87         return Pjn_1
88
89
90
91     def rk3(self):
92         """
93         Runge-Kutta 3rd order Centred in Space
94         """
95         Pj_0G = self.Pj
96
97         Pjn_1 = []
98
99         for n in range(len(self.nsteps)):
100
101             Pj = self.Pj
102             # print(Pj[50], "Pj var")
103             P_str = Pj + (self.dt/3) * self.centdif()
104             # print(P_str[50], 'P_str')
105
106             self.Pj = P_str
107             # print(self.Pj[50], 'self Pj should be Pjstr')
108
109             P_str_str = Pj + (self.dt/2) * self.centdif()
110             # print(P_str_str[50], 'P_str_str')
111
112             self.Pj = P_str_str
113             # print(self.Pj[50], 'self Pj should be Pj_str_str')
114
115             Pn = Pj + self.dt * self.centdif()
116             Pn = np.array(Pn)
117             # print(Pn[50], "Pn pre append")
118             Pjn_1.append(Pn)
119

```

```

120         self.Pj = Pn
121         # print(self.Pj[50], "self Pj or Pn")
122
123     Pjn_1 = np.array(Pjn_1)
124     self.Pj = Pj_0G
125
126     return Pjn_1
127
128 def plot_functions(self, method):
129     if method == 'Initial':
130         fig, ax = plt.subplots(1,1, figsize=(12,4))
131         fig.suptitle('HW7 Initial concentration', \
132                     fontsize= plt_set.title_size, fontweight="bold")
133         ax.plot(self.xx, self.Pj, color = 'blue', \
134                label = "Initial concentration", zorder = 9)
135         ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
136         ax.set_ylabel('Quantity', fontsize = plt_set.label)
137         ax.xaxis.grid(color='gray', linestyle='dashed')
138         ax.yaxis.grid(color='gray', linestyle='dashed')
139         ax.set_ylim(-10,15)
140         ax.legend()
141         plt.show()
142
143     elif method == 'Final':
144         fig, ax = plt.subplots(1,1, figsize=(12,4))
145         fig.suptitle('HW7 Final Ideal', \
146                     fontsize= plt_set.title_size, fontweight="bold")
147         ax.plot(self.xx, self.Pj, color = 'blue', \
148                label = "Initial concentration", zorder = 9)
149         ax.plot(self.xx, self.cideal, color = 'red', \
150                label = "Final Ideal", zorder = 8)
151         ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
152         ax.set_ylabel('Quantity', fontsize = plt_set.label)
153         ax.xaxis.grid(color='gray', linestyle='dashed')
154         ax.yaxis.grid(color='gray', linestyle='dashed')
155         ax.set_ylim(-10,15)
156         ax.legend()
157         plt.show()
158
159     elif method == 'RK3':
160         fig, ax = plt.subplots(1,1, figsize=(12,4))
161         fig.suptitle("Runge-Kutta 3rd order Centred in Space CR: 0.5", \
162                     fontsize= plt_set.title_size, fontweight="bold")
163         ax.plot(self.xx, self.Pj, color = 'blue', \
164                label = "Initial concentration", zorder = 10)
165         ax.plot(self.xx, self.cideal, color = 'red', \
166                label = "Final Ideal", zorder = 8)
167         Prk3 = self.rk3()
168         ax.plot(self.xx, Prk3.T[:, -1], color = 'green', \
169                label = "RK3", zorder = 9)
170         ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
171         ax.set_ylabel('Quantity', fontsize = plt_set.label)
172         ax.xaxis.grid(color='gray', linestyle='dashed')
173         ax.yaxis.grid(color='gray', linestyle='dashed')
174         ax.set_ylim(-10,15)
175         ax.legend()
176         plt.show()
177
178     elif method == 'FTBS':
179         fig, ax = plt.subplots(1,1, figsize=(12,4))

```

```
180     fig.suptitle("Forward in time, Backward in space CR: 0.5", \
181                 fontsize= plt_set.title_size, fontweight="bold")
182     ax.plot(self.xx, self.Pj, color = 'blue', \
183            label = "Initial concentration", zorder = 10)
184     ax.plot(self.xx, self.cideal, color = 'red', \
185            label = "Final Ideal", zorder = 8)
186     Ftbs = self.forward()
187     print(Ftbs.shape)
188     ax.plot(self.xx, Ftbs.T[:, -1], color = 'green', \
189            label = "FTBS", zorder = 9)
190     ax.set_xlabel('Grid Index (i)', fontsize = plt_set.label)
191     ax.set_ylabel('Quantity', fontsize = plt_set.label)
192     ax.xaxis.grid(color='gray', linestyle='dashed')
193     ax.yaxis.grid(color='gray', linestyle='dashed')
194     ax.set_ylim(-10, 15)
195     ax.legend()
196     plt.show()
197
198     else:
199         pass
200
201
202     return
203
204
205
206
207
208
209
210
```