

# **A Combined Droplet Evaporation/Break-up Model for the Atomization of Sprays**

Adam O'Brien

Student Number: 1000195022

Course: Multi-phase Flows

Date: December 17, 2014

# Contents

<b>1</b>	<b>Objective</b>	<b>3</b>
<b>2</b>	<b>Droplet Advection</b>	<b>3</b>
<b>3</b>	<b>Evaporation Model</b>	<b>4</b>
<b>4</b>	<b>Taylor Analogy Break-up (TAB) Model</b>	<b>5</b>
<b>5</b>	<b>Results</b>	<b>9</b>
<b>6</b>	<b>Concluding Remarks</b>	<b>9</b>
<b>A</b>	<b>boTAB Code - Main Module</b>	<b>11</b>
<b>B</b>	<b>boTAB Code - Fluid Module</b>	<b>14</b>
<b>C</b>	<b>boTAB Code - Input Module</b>	<b>22</b>
<b>D</b>	<b>boTAB Code - Output Module</b>	<b>26</b>
<b>E</b>	<b>boTAB Code - Evaporation Module</b>	<b>27</b>
<b>F</b>	<b>boTAB Code - TAB Module</b>	<b>29</b>
<b>G</b>	<b>boTAB Code - Sample Input File</b>	<b>33</b>

# 1 Objective

The objective of this report is to present a code, written in the popular programming language Python, which is used to model the evaporation and break-up of a continuous stream of liquid droplets within a cross-flow. This simplified approach is used to emulate the process of a spray undergoing atomization.

In section ??, the basic advection scheme for a droplet in cross-flow is presented. In section ??, the semi-empirical evaporation model, assuming constant droplet temperature, will be presented. In section ??, the Taylor Analogy Break-up (TAB) model will be presented. The TAB model simplifies the dynamics of an oscillating droplet by modelling it as a spring-mass-damper harmonic oscillator. In section ??, select results will be shown. Finally, in section ??, some final remarks will be made.

## 2 Droplet Advection

Droplet advection is accomplished numerically, using a second-order predictor-corrector time integration scheme. The predictor-corrector scheme has the form

$$f(t, y) = \frac{dy}{dt} \quad (1)$$

$$\tilde{y}^{n+1} = y^n + \Delta t f(t^n, y^n) \quad (2)$$

where  $\tilde{y}^{n+1}$  is the predictor step, and  $\Delta t$  is the time step. This step is identical to Euler time integration. The corrector step is given by

$$y^{n+1} = y^n + \frac{1}{2} \Delta t (f(t^n, y^n) + f(t^{n+1}, \tilde{y}^{n+1})) \quad (3)$$

In order to evaluate  $f(t, y)$ , the equation for drag on a sphere, given by

$$\vec{F}_{drag} = \frac{1}{2} \rho_g |\vec{v}_{rel}|^2 c_d A_{ref} \cdot \vec{n}_{v_{rel}} \quad (4)$$

is used, where  $\rho_g$  is the air around the droplet based on  $T_{ref}$  (introduced later),  $\vec{v}_{rel}$  is the relative velocity between the droplet and the freestream,  $c_d$  is the drag coefficient which is empirically correlated,  $A_{ref}$  is the sphere reference area, and  $\vec{n}_{v_{rel}}$  is a unit vector in the direction of  $\vec{v}_{rel}$ . The drag coefficient correlation is not given here for the sake of conciseness, but many correlations are readily available in the literature.

### 3 Evaporation Model

The evaporation model assumes that the droplet can be modelled from the well known " $D^2$ -law", which has the form [?]

$$D^2 = D_0^2 - \lambda t \quad (5)$$

where  $D_0$  is the original droplet diameter, and  $\lambda$  is the evaporation constant under quiescent conditions. For a single droplet, the rate of mass loss due to evaporation can be expressed as

$$\frac{\delta m}{\delta t} = \frac{\pi}{4} \rho_l \lambda D \quad (6)$$

where  $\rho_l$  is the liquid density. Equation (??) can be modified to account for the effects of forced convection, or cross-flow, by using the following modification:

$$\frac{\delta m'}{\delta t} = 2\pi D \left( \frac{k_g}{c_g} \right) \ln(1 + B_M) [1 + 0.3 Re_d^{0.5} Pr_g^{0.33}] \quad (7)$$

where  $k_g$ ,  $c_g$ ,  $Re_d$  and  $Pr_g$  are, respectively, the thermal conductivity, the specific heat, the Reynolds number and the Prandtl number relative to the cross-flow surrounding the droplet. The mass transfer number,  $B_M$ , is computed as

$$B_M = \frac{Y_{l,s}}{1 - Y_{l,s}} \quad (8)$$

where  $Y_{l,s}$  is the liquid mass fraction at the droplet surface, and must be obtained through the use of the Clausius-Clapeyron relationship. The Clausius-Clapeyron relationship is given by

$$Y_{l,s} = \frac{1}{1 + \frac{PM_g}{(P_{l,s}-1)M_d}} \quad (9)$$

where  $P$  is the pressure surrounding the droplet,  $M_g$  is the molecular weight of the air,  $P_{l,s}$  is the vapour pressure of water and  $M_d$  is the molecular weight of water. The vapour pressure of water can be found from any number of empirical correlations, with the temperature ( $T_{ref}$ ) assumed to be an average between the ambient air temperature and the temperature of the droplet. Here, the following average is used:

$$T_{ref} = T_{droplet} + \frac{T_{ambient} - T_{droplet}}{3} \quad (10)$$

With  $T_{ref}$ , the vapour pressure can be correlated. The correlation is not given here for conciseness, but they are readily available within the experimental literature.

Once the surface vapour mass fraction ( $Y_{l,s}$ ) is known, equation (??) is used to obtain  $B_M$ , and the modified droplet evaporation constant,  $\lambda'$ , is computed by

$$\lambda' = \frac{8k_g \ln(1 + B_M)}{c_g \rho_l} [1 + 0.3Re_d^{0.5} Pr_g^{0.33}] \quad (11)$$

where the "prime" superscript denotes a quantity which has been modified to account for advection. Equation (??) can then be used to update the droplet size at the current time step.

## 4 Taylor Analogy Break-up (TAB) Model

In addition to evaporation, the spray atomization model also includes a break-up model. The Taylor Analogy Break-up (TAB) model assumes the

droplet can be represented by a simple spring-mass-damper system, where the effects of surface tension are represented by the spring, the droplet mass distribution by the mass, and the effects of viscous dissipation by the damper [?]. The model is shown graphically in figure ??.

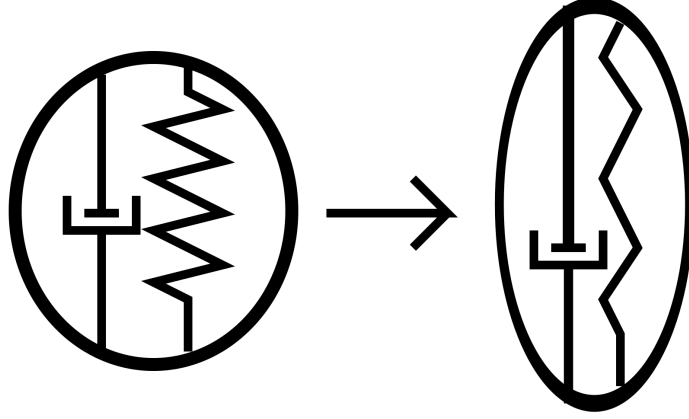


Figure 1: Droplet deformation modelled by the Taylor Analogy Break-up (TAB) model.

The equation for a spring-mass-damper system is given as follows:

$$\frac{d^2y}{dt^2} + 2\zeta\omega_0 \frac{dy}{dt} + \omega_0^2 y = F(t) \quad (12)$$

where  $y$  is the displacement,  $\zeta$  is the damping coefficient,  $\omega_0$  is the undamped natural frequency, and  $F(t)$  is a time-dependant forcing term.

For a simple droplet, the equation of motion was determined to be

$$\frac{d^2y}{dt^2} + \frac{4C_d\mu}{\rho_l d^2} \frac{dy}{dt} + \frac{8C_k\sigma}{\rho_l d^3} y = \frac{4C_f\rho_g|\vec{v}_{rel}|^2}{C_b\rho_l d^2} \quad (13)$$

where  $\mu$ ,  $d$ , and  $\sigma$  are the droplet viscosity, diameter and surface tension coefficient respectively and  $C_d$ ,  $C_k$ ,  $C_f$  and  $C_b$  are empirically determined constants with the following values:

$$C_d = 10 \quad (14a)$$

$$C_k = 8 \quad (14b)$$

$$C_f = 2/3 \text{ (gradual loading)} \quad (14c)$$

$$C_f = 1/3 \text{ (sudden loading)} \quad (14d)$$

$$C_b = 1/2 \quad (14e)$$

The value of  $y$  is non-dimensional, and takes the form

$$y = \frac{4\Delta d_{min}}{d_0} \quad (15)$$

where  $d_{min}$  describes the deviation between the minor drop diameter and that of its original diameter,  $d_0$ .

The solution to equation (??) has the form:

$$y(t) = We_c + e^{-t/\tau_D} \left\{ (y_0 + We_c) \cos(\omega t) + \frac{1}{\omega} \left( \frac{dy_0}{dt} + \frac{y_0 - We_c}{\tau_D} \right) \sin(\omega t) \right\} \quad (16)$$

where

$$\omega = \left\{ \frac{8C_k\sigma}{\rho_l d^3} - \frac{1}{\tau_D^2} \right\}^{\frac{1}{2}} \quad (17)$$

$$\tau_D = \frac{\rho_l d^2}{2C_d \mu} \quad (18)$$

and the critical Weber number,  $We_c$ , is

$$We_c = We \left( \frac{C_f}{C_k C_b} \right) \quad (19)$$

When considering a single discrete time step, and assuming that the droplet initially is not deformed nor deforming ( $y_0 = \frac{dy_0}{dt} = 0$ ), equation (??) and its derivative take the forms

$$y^{n+1} = We_c + e^{-\Delta t/\tau_D} \left\{ (y^n + We_c) \cos(\omega \Delta t) + \frac{1}{\omega} \left( \left( \frac{dy}{dt} \right)^n + \frac{y^n - We_c}{\tau_D} \right) \sin(\omega \Delta t) \right\} \quad (20)$$

$$\left( \frac{dy}{dt} \right)^{n+1} = \frac{We_c - y^{n+1}}{\tau_D} + \omega e^{-\Delta t/\tau_D} \left\{ \frac{1}{\omega} \left( \left( \frac{dy}{dt} \right)^n + \frac{y^n - We_c}{\tau_D} \right) \cos(\omega \Delta t) - (y^n - We_c) \sin(\omega \Delta t) \right\} \quad (21)$$

From equations (??) and (??), the distortion of each droplet may be tracked. Breakup is said to occur when  $y(t) \geq 1$ .

After a break-up occurs, the size of the child droplets must be determined. This is done by equating the energy of the parent droplet to the combined energy of the child droplets. The energy of the parent droplet is

$$E_{parent} = 4\pi r^2 \sigma + K \frac{\pi}{5} \rho_l r^5 \left[ \left( \frac{dy}{dt} \right)^2 + \omega^2 y^2 \right] \quad (22)$$

where  $K$  is the ratio of total energy in distortion and oscillation, and is set to

$$K = \frac{10}{3} \quad (23)$$

The energy of the child droplets can then be shown to be

$$E_{child} = 4\pi r^2 \sigma \frac{r}{r_{32}} + \frac{\pi}{6} \rho_l r^5 \left( \frac{dy}{dt} \right)^2 \quad (24)$$

where  $r_{32}$  is the Sauter mean radius of the droplet distribution. Setting  $y = 1$ , subbing in equation (??), and setting equations (??) and (??) equal to each other, one obtains the equation for the Sauter mean radius as



$$r_{32} = \frac{r}{1 + \frac{8Ky^2}{20} + \frac{\rho_l r^3 (dy/dt)^2}{\sigma} \left( \frac{6K-5}{120} \right)} \quad (25)$$

The droplets can then be assumed to be normally distributed about  $r_{32}$ , with the number of droplets determined through mass conservation.

The velocity of the child droplets must also be computed. A normal component of velocity is added onto the droplet, which has the form

$$|\vec{v}_{normal}| = C_v C_b r \left( \frac{dy}{dt} \right) \quad (26)$$

where  $C_v$  is a constant on the order of 1.

## 5 Results

The following section contains results obtained from coding the evaporation and TAB droplet advection code. The code itself has been included in the appendices.

## 6 Concluding Remarks

A code for modelling the atomization of sprays by combining the constant droplet temperature evaporation model, as well as a Taylor Analogy Break-up (TAB) model. The break-up model assumes that the droplet is a simple harmonic oscillator, with a single degree of freedom (a single mode). Improvements could be incorporated by coupling the droplet radius rate of change into the advection scheme, such that the force accounts for a varying reference area. Other improvements could be incorporated by also factoring in the droplet distortion into the advection scheme.

## References

- [1] N. Kabaliuk, M.C. Jermy, E. Williams, T.L. Laber and M.C. Taylor, *Experimental validation of a numerical model for predicting the trajectory of blood drops in typical crime scene conditions, including droplet deformation and breakup, with a study of the effect of indoor air currents and wind on typical spatter drop trajectories. Forensic Science Journal* 2014.
- [2] S.S. Sazhin, I.N. Shishkova and M. Al Qubeissi *Heating and evaporation of a two-component droplet: Hydrodynamic and kinetic models, International Journal of Heat and Mass Transfer* , 2014

## A boTAB Code - Main Module

```
1 #!/usr/bin/env python2
2 # -*- coding: utf-8 -*-
3
4 """
5 boTAB
6 =====
7 This solver uses the popular TAB model to simulate the
8 atomization of droplets
9 Author: Adam O'Brien
10 """
11
12 from input import *
13 from math import exp, cos, sin, sqrt
14 from fluid import *
15 from evaporation import *
16 from TAB import *
17 from output import *
18 import copy as cp
19
20 def main():
21     print ""
22     print "boTAB|"
23     print "_____"
24     print "Compute the break-up of a drop in a
25         uniform cross-flow", "\n"
26
27     # Open up a configuration file
28
29     userInput = readInputFile()
30
31     freestream = Freestream()
32     initialDroplet = Droplet()
33     dropletInlet = DropletInlet()
```

```

34  # Set object parameters from the input file
35
36  setObjectParametersFromInput(userInput, freestream,
    initialDroplet, dropletInlet)
37
38  # Set-up the simulation parameters in accordance
    with the input
39
40  maxTime = userInput["maxTime"]
41  nTimeSteps = userInput["nTimeSteps"]
42
43  # Initialize a droplet list, with one copy of the
    initial droplet
44
45  droplets = [cp.deepcopy(initialDroplet)]
46
47  # Initialize misc parameters
48
49  dt = maxTime/nTimeSteps
50  t = [0.]
51  nChildDroplets = 0
52
53  # Begin the simulation
54
55  print "\nBeginning_time-stepping..."
56
57  #####
58  #                                     #
59  #           Main Iteration Loop           #
60  #                                     #
61  #####
62
63  for stepNo in range(1, nTimeSteps + 1):
64
65      for droplet in droplets:
66
67          droplet.advectPredictorCorrector(freestream
            , dt)

```

```

68
69     evaporate(freestream, droplets, dt)
70     nChildDroplets += breakupTab(freestream,
71                                   droplets, dt)
72
73     dropletInlet.addDrops(initialDroplet, droplets,
74                             dt)
75     t.append(t[-1] + dt)
76
77     if stepNo%(nTimeSteps/20) == 0:
78         completionPercentage = float(stepNo)/float(
79             nTimeSteps)*100.
80
81         print "
82             "
83         print "Time-stepping_completion: %s%%"
84             %(completionPercentage)
85         print "Number_of_droplets_in_domain:", len
86             (droplets)
87         print "Simulation_time_elapsed: %s
88             seconds"%(t[-1])
89         print "Simulation_time_remaining: %s
90             seconds"%(maxTime - t[-1])
91         print "Number_of_child_drops:",
92             nChildDroplets
93
94     print "\nTime-stepping_complete. Finalizing_output
95         ..."
96
97     plotDroplets(droplets)
98
99     # Execute the main function
100
101     if __name__ == "__main__":
102         main()

```

## B boTAB Code - Fluid Module

```
1
2 #!/usr/bin/env python2
3 # -*- coding: utf-8 -*-
4
5 """
6 boTAB
7 =====
8 This module contains classes and function for the
   modelling of droplets in
9 freestream flows
10 Author: Adam O'Brien
11 """
12
13 from math import sqrt, pi, exp
14 import copy as cp
15 import random
16
17 # Vector class for position/velocity
18
19 class Vector(object):
20
21     def __init__(self, x, y):
22         self.x = x
23         self.y = y
24
25     def __repr__(self):
26
27         return "%s, %s"%(self.x, self.y)
28
29     def __add__(self, other):
30
31         return Vector(self.x + other.x, self.y + other.
32                       y)
33
34     def __sub__(self, other):
```

```

34
35         return Vector(self.x - other.x, self.y - other.
36                        y)
37
38 def __mul__(self, other):
39
40         return Vector(self.x*other, self.y*other)
41
42 def mag(self):
43
44         return sqrt(self.x**2 + self.y**2)
45
46 def normalVector(self):
47
48         return Vector(self.y, -self.x)
49
50 def scale(self, other):
51
52         return Vector(self.x*other, self.y*other)
53
54 def rVector(self, other):
55
56         return other - self
57
58 def unitVector(self):
59
60         return self.scale(1./self.mag())
61
62 # Freestream class for representing the freestream flow
63
64 class Freestream(object):
65
66     def __init__(self, velocity = Vector(40., 0.),
67                 gravity = Vector(0., 0.)):
68
69         # constructed properties

```

```

70         self.velocity = velocity
71         self.gravity = gravity
72
73         # default air proplerties (can be changed)
74
75         self.mu = 18.27e-6
76         self.Pambient = 101325.
77         self.Tambient = 800.
78         self.Cp = 1.0005
79         self.k = 0.0257
80         self.M = 28.97
81         self.Pr = self.Cp*self.mu/self.k
82
83         # Computed properties
84
85         self.rho = self.Pambient/(286.9*self.Tambient)
86
87 # Droplet class for representing droplets
88
89 class Droplet(object):
90
91     def __init__(self, radius = 5e-4, position = Vector
92                 (0., 0.), velocity = Vector(0., 10.)):
93
94         # constructed properties
95
96         self.radius = radius
97         self.position = position
98         self.velocity = velocity
99
100        # default water properties (can be changed)
101
102        self.rho = 998.
103        self.mu = 8.94e-4
104        self.sigma = 0.07262
105        self.Tboil = 373.
106        self.Tcrit = 647.096
107        self.T = 400.

```



```

107         self.L = 2257.
108         self.Cp = 4.183
109         self.k = 0.58
110         self.M = 18.01528
111
112         # TAB properties
113
114         self.y = 0.
115         self.dydt = 0.
116
117     def __repr__(self):
118
119         return "Radius: %s\nPosition: %s\nVelocity: %s"
120             %(str(self.radius), \
121               self.position, self.velocity)
122
123     def diameter(self):
124
125         return 2.*self.radius
126
127     def volume(self):
128
129         return (4./3.)*pi*self.radius**3
130
131     def area(self):
132
133         return pi*self.radius**2
134
135     def mass(self):
136
137         return self.volume()*self.rho
138
139     def Pvap(self, freestream):
140
141         Tref = (2./3.)*self.T + (1./3.)*freestream.
142             Tambient

```

```

143         a2 = 1.84408259
144         a3 = -11.7866497
145         a4 = 22.6807411
146         a5 = -15.9618719
147         a6 = 1.80122502
148
149         tau = 1. - Tref/self.Tcrit
150
151         pOverPc = exp((a1*tau + a2*tau**1.5 + a3*tau**3
152                        + a4*tau**3.5 + a5*tau**4 + a6*tau**7.5)*
153                        self.Tcrit/self.T)
154
155         return pOverPc*22064.
156
157     def We(self , freestream):
158
159         vRel = self.velocity.rVector(freestream.
160                                     velocity)
161
162         return freestream.rho*dot(vRel , vRel)*self.
163                 radius/self.sigma
164
165     def Re(self , freestream):
166
167         return freestream.rho*(freestream.velocity -
168                             self.velocity).mag()*self.diameter()/
169                             freestream.mu
170
171     def dragCoefficient(self , freestream):
172
173         # This drag coefficient for a sphere is based
174         on the correlation of
175         # F.A. Morrison in "An Introduction to Fluid
176         Mechanics"
177
178         Re = self.Re(freestream)

```

```

172         return 24./Re + 2.6*(Re/5.)/(1. + (Re/5.)
           **1.52) + 0.411*(Re/263000.)**−7.94/(1. + (
           Re/263000.)**−8.) + (Re**0.8/461000.)
173
174     def dragForce(self , freestream):
175
176         Cd = self.dragCoefficient(freestream)
177
178         vRel = freestream.velocity − self.velocity
179
180         return vRel.unitVector()*0.5*freestream.rho*dot
           (vRel , vRel)*Cd*self.area()
181
182     def acceleration(self , freestream):
183
184         return self.dragForce(freestream).scale(1./self
           .mass()) + freestream.gravity
185
186     def advectEuler(self , freestream , dt):
187
188         a = self.acceleration(freestream)
189
190         self.position += self.velocity*dt + a*0.5*dt**2
191         self.velocity += a*dt
192
193     def advectPredictorCorrector(self , freestream , dt):
194
195         originalPosition = self.position
196
197         a = self.acceleration(freestream)
198         f1 = self.velocity + a*0.5*dt
199
200         self.position += f1*dt
201
202         a = self.acceleration(freestream)
203         f2 = self.velocity + a*0.5*dt
204

```

```

205         self.position = originalPosition + (f1 + f2)
           *0.5*dt
206         self.velocity += a*dt
207
208     def printAll(self):
209
210         print "Radius:", self.radius
211         print "Rho:", self.rho
212         print "mu:", self.mu
213         print "sigma:", self.sigma
214         print "Boiling_Temp:", self.boilingTemp
215         print "Latent_Heat:", self.latentHeat
216         print "Specific_Heat:", self.specificHeat
217         print "k:", self.k
218         print "Position:", self.position
219         print "Velocity:", self.velocity
220
221     class DropletInlet(object):
222
223         def __init__(self, newDropletFrequency = 1000,
           inletWidth = 0.005, velocityDeviation = 0.):
224
225             self.newDropletFrequency = newDropletFrequency
226             self.inletWidth = inletWidth
227             self.velocityDeviation = velocityDeviation
228             self.timeSinceLastDroplet = 0.
229             self.newDropletPeriod = 1./self.
               newDropletFrequency
230             self.dropsAdded = 0
231
232         def addDrops(self, initialDroplet, droplets, dt):
233
234             self.timeSinceLastDroplet += dt
235
236             nDropsToAdd = int(self.timeSinceLastDroplet/
               self.newDropletPeriod)
237             self.timeSinceLastDroplet -= float(nDropsToAdd)
               *self.newDropletPeriod

```

```

238
239         if nDropsToAdd > 0:
240             self.dropsAdded += nDropsToAdd
241
242         for i in range(0, nDropsToAdd):
243
244             droplets.append(cp.deepcopy(initialDroplet)
245                               )
246             droplets[-1].position.x += random.uniform
247                 (-0.5*self.inletWidth, 0.5*self.
248                 inletWidth)
249             droplets[-1].velocity.x += random.
250                 normalvariate(0., self.velocityDeviation
251                 .x)
252             droplets[-1].velocity.y += random.
253                 normalvariate(0., self.velocityDeviation
254                 .y)

```

## C boTAB Code - Input Module

```
1
2 # -*- coding: utf-8 -*-
3
4 """
5 boTAB
6 =====
7 This module is for simple file input and configuration
8 Author: Adam O'Brien
9 """
10
11 from fluid import Vector
12
13 def str2num(string):
14
15     if string.partition("/")[1] == "/":
16
17         string = string.partition("/")
18
19         return float(string[0])/float(string[2])
20
21     elif string.partition(",")[1] == ",":
22
23         string = string.partition(",")
24
25         return Vector(float(string[0]), float(string
26                             [2]))
27
28     else:
29
30         try:
31
32             return int(string)
33
34         except ValueError:
```

```

35         return float(string)
36
37
38 def process(inputDict, line):
39
40     line = line.replace("\n", "")
41     line = line.replace("_", "")
42     line = line.partition("#")
43     line = line[0]
44
45     if line == "":
46         return
47
48     line = line.partition("=")
49
50     if not line[1] == "=":
51         print "Warning, _potentially_bad_input:", line
52
53     inputDict[line[0]] = str2num(line[2])
54
55     return
56
57 def readInputFile(filename = "config.in"):
58
59     userInput = {}
60
61     print "Reading_from_input_file \" config.in \" ..."
62
63     with open(filename) as inFile:
64
65         for line in inFile:
66
67             process(userInput, line)
68
69     print "The_following_input_parameters_have_been_
70         loaded_from \" {} \" ".format(filename), "\n"
71
72     for parameter in userInput:

```

```

72
73         print parameter, "=", userInput[parameter]
74
75     print "Finished reading from input file \" config.in
76         \"."
77
78     return userInput
79
80 def setObjectParametersFromInput(userInput, freestream,
81     droplet, inlet):
82
83     # Freestream properties
84
85     freestream.rho = userInput["freestreamRho"]
86     freestream.mu = userInput["freestreamMu"]
87     freestream.Tambient = userInput["freestreamTambient
88         "]
89     freestream.Cp = userInput["freestreamCp"]
90     freestream.K = userInput["freestreamK"]
91     freestream.velocity = userInput["freestreamVelocity
92         "]
93     freestream.gravity = userInput["freestreamGravity"]
94     freestream.M = userInput["freestreamM"]
95
96     # Initial droplet properties
97
98     droplet.radius = userInput["dropletRadius"]
99     droplet.rho = userInput["dropletRho"]
100    droplet.mu = userInput["dropletMu"]
101    droplet.sigma = userInput["dropletSigma"]
102    droplet.Tboil = userInput["dropletTboil"]
103    droplet.L = userInput["dropletL"]
104    droplet.Cp = userInput["dropletCp"]
105    droplet.K = userInput["dropletK"]
106    droplet.position = userInput["dropletPosition"]
107    droplet.velocity = userInput["dropletVelocity"]
108    droplet.M = userInput["dropletM"]

```



```
106      # Droplet inlet properties
107
108      inlet.newDropletFrequency = userInput["
          inletDropletCreationFrequency"]
109      inlet.inletWidth = userInput["inletWidth"]
110      inlet.velocityDeviation = userInput["
          inletVelocityDeviation"]
```

## D boTAB Code - Output Module

```
1
2 # -*- coding: utf-8 -*-
3
4 """
5 boTAB
6 =====
7 This module is for various plotting outputs
8 Author: Adam O'Brien
9 """
10
11 import matplotlib.pyplot as plt
12 import matplotlib.animation as animation
13
14 def plotDroplets(droplets):
15
16     xcoords = [droplet.position.x for droplet in
17                 droplets]
18     ycoords = [droplet.position.y for droplet in
19                 droplets]
20     radii = [50000.*droplet.radius for droplet in
21              droplets]
22
23     plt.axis('equal')
24     plt.grid(True)
25     plt.xlabel('x(m)', fontsize=16)
26     plt.ylabel('y(m)', fontsize=16)
27
28     plt.scatter(xcoords, ycoords, s=radii, alpha=0.5)
29
30     plt.show()
```

## E boTAB Code - Evaporation Module

```
1
2 # -*- coding: utf-8 -*-
3
4 """
5 boTAB
6 =====
7 This module contains classes and function for the
   modelling of droplet
8 evaporation in freestream flows
9 Author: Adam O'Brien
10 """
11
12 from math import log, sqrt, fabs, pi
13
14 def clausiusClapeyron(freestream, droplet):
15
16     # This function determines the water vapour mass
17     # fraction at the surface
18     # of a droplet
19     return 1./((1. + freestream.Pambient*freestream.M/((
20         droplet.Pvap(freestream) - 1.)*droplet.M))
21
22 def evaporate(freestream, droplets, dt):
23
24     for i in range(0, len(droplets)):
25
26         Yls = clausiusClapeyron(freestream, droplets[i])
27
28         BM = Yls/(1. - Yls)
29
30         Re = droplets[i].Re(freestream)
31         Pr = freestream.Pr
```

```

32         gamma = 8.*freestream.k*log(1. + BM)/(
           freestream.Cp*droplets[i].rho)*(1. + 0.3*Re
           **0.5*Pr**(1./3.))
33
34         D2 = droplets[i].diameter()**2 - gamma*dt
35
36         if D2 > 0.:
37
38             droplets[i].radius = 0.5*D2**0.5
39
40         else:
41
42             droplets[i].radius = 0.
43
44         # Discard any droplets that have a radius less than
           the tolerance, ie they
45         # are completely evaporated
46
47         droplets[:] = [droplet for droplet in droplets if
           not droplet.radius <= 1e-10]

```

## F boTAB Code - TAB Module

```
1
2 # -*- coding: utf-8 -*-
3
4 """
5 boTAB
6 =====
7 This module contains the functions necessary for TAB
8 breakup
9 Author: Adam O'Brien
10 """
11
12 from math import exp, sin, cos, fabs
13 import numpy as np
14 import copy as cp
15 from fluid import *
16
17 # Model constants
18 Cb = 0.5
19 Ck = 8.
20 Cd = 5.
21 Cf = 1./3.
22 Cd = 5.
23 K = 10./3.
24 Cv = 1.
25
26 def getSMR(droplet):
27
28     # This function computes the Sauter Mean Radius (
29     # SMR) of the child droplets after a break-up
30
31     rOverRmean = 1. + (K/5.)*Ck*Cb**2 + droplet.rho*
32         droplet.radius**3/droplet.sigma*Cb**2*droplet.
33         dydt**2*(6.*K - 5.)/30.
```

```

32     return droplet.radius/rOverRmean
33
34 def breakupTab(freestream , droplets , dt):
35
36     newDroplets = []
37
38     for droplet in droplets:
39
40         Wec = droplet.We(freestream)*Cf/(Ck*Cb)
41         td = droplet.rho*droplet.diameter()2/(2.*Cd*
42             droplet.mu)
43         omega = (8.*Ck*droplet.sigma/(droplet.rho*
44             droplet.diameter()3) - (1./td**2))**0.5
45
46         yn = droplet.y
47
48         droplet.y = Wec + exp(-dt/td)* \
49             ((yn - Wec)*cos(omega*dt) + 1./
50             omega*(droplet.dydt + (yn - Wec)
51             /td)*sin(omega*dt))
52
53         droplet.dydt = (Wec - droplet.y)/td + \
54             omega*exp(-dt/td)* \
55             (1./omega*(droplet.dydt + (yn -
56             Wec)/td)*cos(omega*dt) - (yn
57             - Wec)*sin(omega*dt))
58
59     if droplet.y >= 1.:
60
61         rSmr = getSMR(droplet)
62
63         # Begin sampling droplets
64
65         volOfNewDrops = 0.
66
67         while True:

```

```

63         newRadius = np.random.normal(rSmr, 0.2*
64             rSmr)
65         if newRadius < 0.:
66             continue
67
68         randomNo = np.random.uniform(-1, 1)
69
70         newVelocity = droplet.velocity +
71             droplet.velocity.normalVector().
72             unitVector()*Cb*droplet.radius*
73             droplet.dydt*(randomNo/fabs(randomNo
74             ))
75
76         newDroplet = Droplet(newRadius, cp.copy
77             (droplet.position), cp.copy(
78             newVelocity))
79
80         newDroplets.append(newDroplet)
81
82         volOfNewDrops += newDroplets[-1].volume
83             ()
84
85         if volOfNewDrops >= droplet.volume():
86             break
87
88         # Remove the old parent droplets
89
90         droplets[:] = [droplet for droplet in droplets if
91             droplet.y < 1.]
92
93         # Add the newly created child droplets
94
95         for droplet in newDroplets:
96             droplets.append(cp.deepcopy(droplet))

```

```
92
93     # Return the number of droplets created
94
95     return len(newDroplets)
```



## G boTAB Code - Sample Input File

```
1
2 ## This is the input file for boTAB
3
4 # Empirical constants for the TAB model
5 Cb = 0.5
6 Ck = 8.
7 Cd = 5.
8 Cf = 1./3.
9 K = 10./3.
10 Cv = 1.
11
12 # Freestream properties
13 freestreamRho = 1.205
14 freestreamMu = 18.27e-6
15 freestreamTambient = 298.
16 freestreamCp = 1.401
17 freestreamK = 0.024
18 freestreamVelocity = 40., 0.
19 freestreamGravity = 0., 0.
20 freestreamM = 28.97
21
22 # Initial droplet properties
23 dropletRadius = 5.e-4
24 dropletRho = 998.
25 dropletMu = 8.94e-4
26 dropletSigma = 0.07262
27 dropletTboil = 373.
28 dropletL = 2257.
29 dropletCp = 4.183
30 dropletK = 0.58
31 dropletPosition = 0., 0.
32 dropletVelocity = 0., 20.
33 dropletM = 18.01528
34
35 # Inlet parameters
```

```
36 inletDropletCreationFrequency = 10000
37 inletWidth = 0.01
38 inletVelocityDeviation = 0., 0.
39
40 # Simulation parameters
41 maxTime = 0.05
42 nTimeSteps = 2000
```