

Supplementary material for On the sea aerosol originated from bubble bursting jets

Francisco J. Blanco–Rodríguez and J. M. Gordillo[†]

Área de Mecánica de Fluidos, Departamento de Ingeniería Aeroespacial y Mecánica de Fluidos, Universidad de Sevilla, Avenida de los Descubrimientos s/n 41092, Sevilla, Spain

(Received xx; revised xx; accepted xx)

This supplementary material is devoted to compare the solution given by the leading order solution of the ballistic model reported in the main text with that provided when the capillary pressure gradient term is included. Here we also provide with the numerical code (script), developed in GERRIS, used to obtain the numerical results reported.

Capillary pressure correction to the ballistic model

The ballistic description of the jet in section §3 can be improved by including the capillary pressure gradient term to the momentum equation in (3.2) of the main text:

$$\frac{D}{D\tilde{t}}(\ln r^2) = -\frac{\partial u}{\partial \tilde{z}} = -S, \quad \frac{Du}{D\tilde{t}} = -\frac{\partial}{\partial \tilde{z}}\left(\frac{1}{r}\right). \quad (0.1)$$

The approximate solution to the system (0.1) can be found substituting the ansatz $r/\delta = r_0 + \delta^{-1}r_1$, $u = u_0 + \delta^{-1}u_1$ into (0.1) with $\delta \gg 1$, yielding

$$r(\tilde{z}, \tilde{t}) = \delta \frac{\tilde{t}^3}{\tilde{z}^{7/2}} \left[1 - \delta^{-1} \left(\frac{21}{2} \right) \frac{\tilde{z}^{3/2}}{\tilde{t}} \right], \quad u(\tilde{z}, \tilde{t}) = \frac{\tilde{z}}{\tilde{t}} \left[1 - \delta^{-1} \left(\frac{7}{3} \right) \frac{\tilde{z}^{3/2}}{\tilde{t}} \right]. \quad (0.2)$$

where $\tilde{t} = t + t_0$ and $\tilde{z} = z + z_0$ according to the notation used in §3. The leading-order solution obtained in (3.7) is slightly corrected by capillary pressure effects given by the terms of order $O(\delta^{-1})$ in (0.2).

Figure 1 shows that the inclusion of the capillary pressure gradient in the 1D ballistic equations describing the flow in the jet improves the quantitative agreement with the numerical solution for all instants of time. However, we preferred to keep in the main text the simplest possible description, since it already retains the essential feature that the both the radii of the droplets and the velocities at which they are ejected are non monotonic functions of the Ohnesorge number, Oh . The abrupt increment on the sizes of the droplets ejected and also the decrease on the droplet velocities for values of the Ohnesorge number above Oh'_c or Oh_c , respectively, is caused by the fact that the growth of capillary instabilities is inhibited when $\pi r_j(t)/b(t) > 1$, with $r_j(t)$ the instantaneous jet radius at the vertical position where the drop is located and $b(t)$ the instantaneous drop radius.

[†] Email address for correspondence: jgordill@us.es

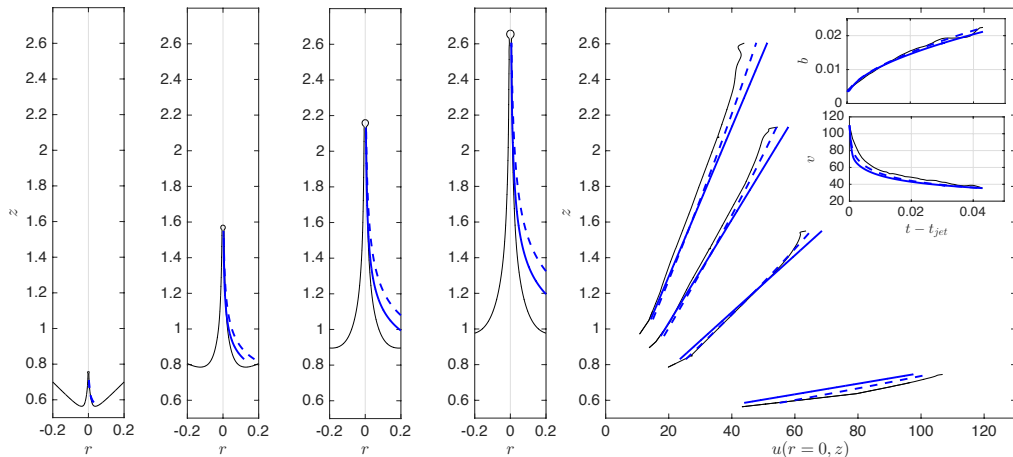


FIGURE 1. Comparison between numerical results for the jet shapes and velocity profiles using the first order correction (blue dashed lines). The proper agreement obtained for the velocity field with the leading order approach (blue solid lines) (cf. figure 2 (b)) is slightly improved adding the deceleration provoked by the capillary pressure.

Governing equations and numerical script

The dimensionless mass and momentum conservation equations for incompressible Newtonian fluids for the liquid and air phases (see Figure 1 (a) of the main document) are given by

$$\nabla \cdot \mathbf{v} = 0, \quad (0.1)$$

$$\rho(T) \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \nabla \cdot (Oh \mu(T) (\nabla \mathbf{v} + \nabla \mathbf{v}^T)) + \kappa \delta_s \mathbf{n} + Bo \rho(T) \mathbf{e}_g. \quad (0.2)$$

being \mathbf{v} the velocity vector field and p the pressure field and where $\mathbf{n} = \frac{\nabla T}{|\nabla T|}$ is the unit normal vector at the interface, $\kappa = -\nabla \cdot \mathbf{n}$ is the mean curvature of the interface, δ_s is the interface delta function and $\mathbf{e}_g = -\mathbf{e}_z$ a unit vector pointing in the direction of the gravity acceleration which coincides with the negative direction of the x -axis in the GERRIS numerical setup.

The interface is captured by solving the advection equation for the volume fraction T in its conservative form as

$$\frac{\partial T}{\partial t} + \nabla \cdot (T \mathbf{v}) = 0. \quad (0.3)$$

The density and viscosity in the biphasic medium is modeled using the VOF-function, T , as

$$\rho(T) = T + \left(\frac{\rho_g}{\rho_l} \right) (1 - T), \quad \mu(T) = T + \left(\frac{\mu_g}{\mu_l} \right) (1 - T). \quad (0.4)$$

The following lines correspond to the GERRIS script developed for obtaining the numerical calculations of the bubble bursting problem.

LISTING 1. GERRIS code for solving the bursting bubble problem

```

1 /* See run_scheduler_mpiBursting.py
2 Execution: python run_scheduler_mpiBursting.py &
3 DURATION -> Final time
4 OH       -> Ohnesorge number
5 BOND     -> Bond number
6 MAXLEVEL -> Maximum level of refinement
7 CASE    -> folder case

```

```

8
9 Numerical simulation of a bubble bursting -> Worthington Jet
10 Following Gordillo and Rodriguez-Rodriguez,
11 Journal of Fluid Mechanics, Vol 867, pp. 556-571, (2019)
12 Francisco J. Blanco-Rodriguez, Fluids Mechanics Group
13 University of Seville (SPAIN) Sep 2019
14
15 The problem has been non-dimensionalized using initial radius
16 of the bubble (lc=R), capillary velocity (vc=sqrt(sigma/(rho1*R))),
17 capillary time (tc=lc/vc=sqrt(rho*R^3/sigma)), and pressure
18 pc=rho1*vc^2 = sigma/R
19 Non-dimensional parameters are:
20     We = rho1*vc^2*R/sigma = 1;
21     Bo = rho1*g*R^2/sigma;
22     Oh = mul/sqrt(rho1*R*sigma)
23 /**/
24
25 2 1 GfsAxi GfsBox GfsGEdge {} {
26     /*Stop the simulation at t = DURATION*/
27     Time {end = DURATION dtmax = 1e-2} /*Maximum integration time*/
28     /*
29     #####
30     #####Preamble#####
31     #####
32     */
33     /*
34     Definition of the auxiliary functions used to define
35     the VOF, density and viscosity fields
36     */
37     Define VAR(T,min,max) (min + CLAMP(T,0,1)*(max - min))
38
39     /*Non-dimensional density field*/
40     Define rho1 1.
41     Define rho_ratio 1.2e-3
42     Define RHO(T) VAR(T, rho_ratio, rho1)
43
44     /*Non-dimensional dynamic viscosity field*/
45     Define mul 1.
46     Define mu_ratio 1.8e-2
47     Define MU(T) VAR(T, mu_ratio, mul)
48
49     /*Define the initial refine level, INILEVEL*/
50     Define INILEVEL 6
51
52     /*Box size*/
53     Define DSIZE 4
54
55     /*
56     Redefine max level according to the radial position
57     MAXLEVEL -> MAXLEVELred
58     */
59     Define MAXLEVELred (y<2. ? MAXLEVEL:MAXLEVEL-2)
60
61     /*Initial static shape at equilibrium contained in fscoordinatesBo0.01.h*/
62     Global {
63         #include <stdio.h>
64         #include "fscoordinatesBo0.01.h"
65
66         double init_vf_bubble(double r, double z)
67         {
68             int k;
69             double rin;
70
71             if(z<zmin) {
72                 return(1.0);
73             } else
74             if(z<=zc) {
75                 k = 0;
76                 while(zbub[k] < z) k++;
77                 rin = rbub[k] + (rbub[k+1]-rbub[k])*(z-zbub[k])/(zbub[k+1]-zbub[
78                     k]);
79                 return(r - rin);
80             } else return(1.0);
81         }
82     }

```

```

81
82     double init_vf_fs(double r, double z)
83     {
84         int k;
85         double zin;
86
87         if ((r < rc) & (z < zc)) {
88             return (1.0);
89         } else
90         if (r >= rc) {
91             k = 0;
92             while (rfs[k] < r) k++;
93             zin = zfs[k] + (zfs[k+1] - zfs[k]) * (r - rfs[k]) / (rfs[k+1] - rfs[k]);
94             return (zin - z);
95         } else return (-1.0);
96     }
97
98     double Init_T(double x, double y) {
99         return( init_vf_fs(y, x - 0.5*DSIZE + zfsmin)*
100             init_vf_bubble(y, x - 0.5*DSIZE + zfsmin) );
101     }
102 }
103 /*
104 #####
105 ##### VOF field #####
106 #####
107 **/
108
109 /*Initial refinement (cups refinement)*/
110 Refine ( (fabs(x-zc) < 0.2 && y < 4*rc) ? MAXLEVELred : INILEVEL )
111
112 /*Initializes the VOF field (tracer T)*/
113 VariableTracerVOFHeight T
114 VariableFiltered T1 T 1
115 VariableFiltered Tmesh T 10
116
117 /*
118 #####
119 ##### Navier-Stokes #####
120 #####
121 **/
122
123 /*DV/Dt = alpha(T1)*[-Grad(p) + Oh*nabla*( mu(T1)*(nablaV + nablaV^T) ) + 1*K*
124     nabla(T)] + Bo*vec(eg)*/
125
126 /*
127 This defines the inverse of the non-dimensional density of the fluid as
128 a function of T which is 1/rho-ratio times greater than that of air
129 */
130
131 /*Definition of the flow parameters*/
132 PhysicalParams { alpha = 1./RHO(T1) L = DSIZE} /*Air density*/
133
134 /*Definition of the advection parameter*/
135
136 AdvectionParams { cfl = 0.25 }
137
138 /*
139 ##### Viscosity #####
140 The non-dimensional dynamic viscosity of liquid is 1/mu-ratio times greater
141 than that of air
142 */
143
144 SourceViscosity OH*MU(T1)
145
146 /*##### Surface tension #####*/
147
148 VariableCurvature K T Kmax
149 SourceTension T 1.0 K
150
151 /*##### Gravity (-Bo*vec(ex)) #####*/
152
153 Source {} U -BOND

```

```

154
155 /*
156 #####
157 #####Initialisation#####
158 #####
159 **/
160
161 InitFraction T ({ return Init-T(x, y); })
162 Init { } { U=0 V=0 }
163
164 /*##### Auxiliary fields #####*/
165
166 Init { istep = 1 } {
167     Omega = Vorticity
168     dUdx = dx("U")
169 }
170
171 /*
172 #####
173 ##### AMR #####
174 #####
175 **/
176
177 /*
178 Adapt the mesh using the gradient criterion based on the interface Tmesh
179 and the curvature at every timestep
180 **/
181
182 AdaptGradient { istep = 1 } {
183     cmax = 1e-3
184     cfactor = 4
185     minlevel = INILEVEL
186     maxlevel = (fabs(t-0.45)<0.1 ? MAXLEVELred+2.:MAXLEVELred)
187 } Tmesh
188
189 /* Adapt according to interface curvature*/
190
191 AdaptFunction { istep = 1 } {
192     cmax = 0.1
193     maxlevel = (fabs(t-0.45)<0.1 ? MAXLEVELred+2.:MAXLEVELred)
194     cfactor = 2.
195 } (T > 0. && T < 1.)*dL*fabs(K)
196
197 /*Remove small satellite droplets (only keep the largest pieces)
198 for Bo>0.05*/
199
200 /*RemoveDroplets {start = 0.001 step = 0.001 end = 0.01} T -1*/
201
202 /*
203 #####
204 #####Saving results#####
205 #####
206 **/
207
208
209 /*Data output*/
210
211 GfsOutputTime { istep = 10 } stdout
212
213 /*Saving data*/
214
215 GfsOutputSimulation { step = 0.001 } Bursting/CASE/bubble_%.5f.gfs {
216     variables = U,V,T,P,T1,Omega,dUdx
217 }
218 }
219
220 /*
221 #####
222 #####Boundary conditions#####
223 #####
224 **/
225
226
227 GfsBox {/*Box 1*/

```

```

228     bottom = Boundary { BcDirichlet V 0 }
229     left    = Boundary { BcDirichlet U 0 }
230     top     = Boundary { BcDirichlet V 0 }
231 }
232
233 GfsBox { /*Box 2*/
234     bottom = Boundary { BcDirichlet V 0 }
235     right  = Boundary { BcDirichlet U 0 }
236     top    = Boundary { BcDirichlet V 0 }
237 }
238
239 /*
240 #####
241 #####Boxes connection#####
242 #####
243 /**/
244
245 1 2 right

```