plumeAnalyserSyntheticHorizontal

May 31, 2019

In [1]:
```
from scipy.io.matlab import loadmat
from itertools import product
from scipy.optimize import fmin

import pandas
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import os, sys
import json
import matplotlib.animation as animation

# Set numpy options, notably for the printing of floating point numbers
np.set_printoptions(precision=6)

# Set matplotlib options
mpl.rcParams['figure.dpi'] = 300

%reload_ext autoreload
%autoreload 2
```

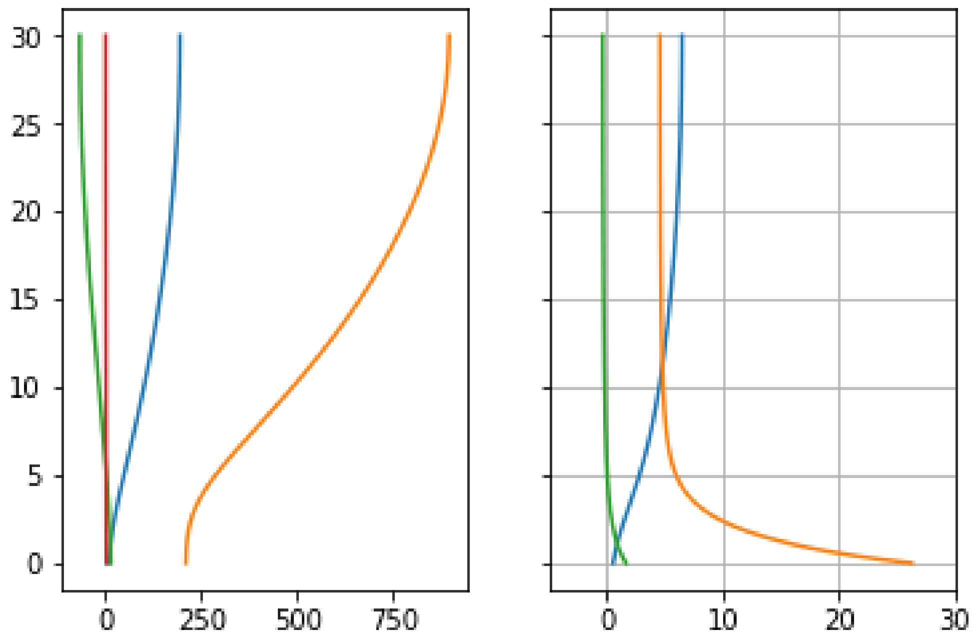
In [2]:
```
run = 3

# Import table of experimental conditions
GCTA = pandas.read_excel('./data/ExpPlumes_for_Dai/TableA1.xlsx', sheet_name='CGSdata',
                         names=('exptNo', 'rhoa0', 'sig_rhoa0', 'N', 'sig_N', 'rho0', 's
                                'gp', 'sig_gp', 'Q0','sig_Q0', 'M0', 'sig_M0', 'F0', 'si

# Extract densities of ambient and plume, and calculate g' at the source
expt  = GCTA[GCTA['exptNo'] == run]
rhoa0 = expt['rhoa0']
rho0  = expt['rho0']
g = 981 #cm/sš
gp0   = (rhoa0 - rho0) / rhoa0 * g

parameters = pandas.read_excel('./data/ExpPlumes_for_Dai/TableA1.xlsx', sheet_name='CGSp
b0theoretical = parameters[parameters['property'] == 'nozzleSize']['value'].values[0]
u0theoretical = expt['U0'].values[0]
```

```
In [3]: from bentPlumeAnalyser import *
        from fumarolePlumeModel import *
```

0.0.1 Create a synthetic dataset for a vertical plume

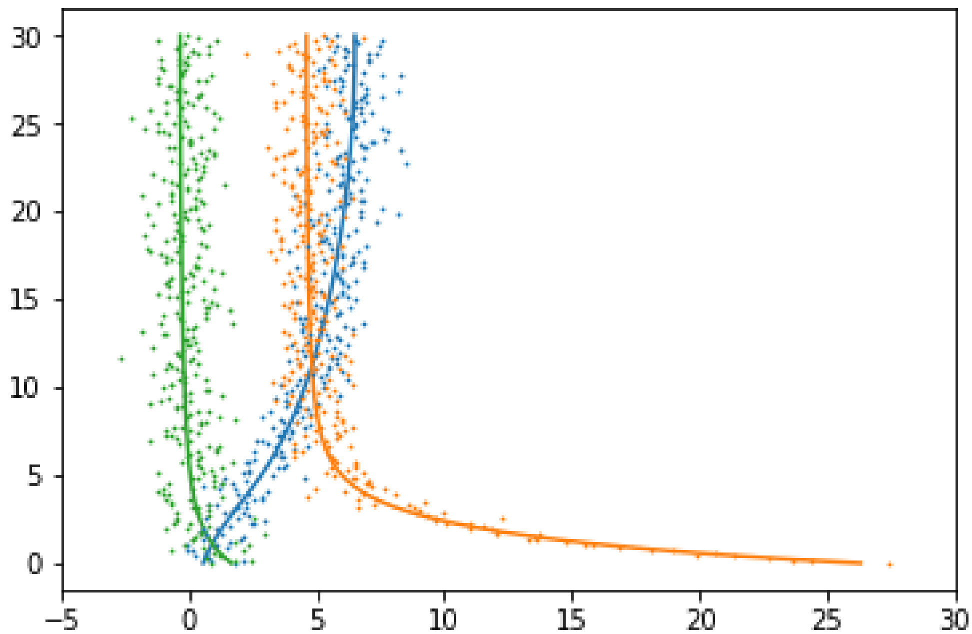
```
In [4]: exptNo      = 3
        plotResults = True

        V0, p = loadICsParameters(pathname, exptNo, alpha=0.05, beta=0.5, m=1)

        #p = list(p)
        #p[1], p[4] = 0., 0.
        #p = tuple(p)

        t1 = 30.
        dt = .1
        sexp = np.arange(0., t1 + dt, dt)
        dsexp = np.diff(sexp)
```

```
In [5]: Q0, M0, F0, theta0 = V0
        s, V = integrator(derivs, V0, p, sexp)

        fig, ax = plt.subplots(1, 2, sharey=True)
        ax[0].plot(V, s, '-')

        Q, M, F, theta = [V[:,i] for i in range(4)]
        b, u, gp = Q / np.sqrt(M), M / Q, F / Q

        V2 = np.array([b, u, gp]).T
        ax[1].plot(V2, s, '-')
        ax[1].set_xlim((-5, 30))
        ax[1].grid()

        sexp = np.copy(s)
        dsexp = np.diff(sexp)
```



0.0.2 Add noise to signal

In [20]:
```
plt.plot(V2, s, '-')
V3 = V2.copy()
ecart_type= 0.3
for i in range(3):
    noise = np.random.normal(0, .8, len(sexp))
    V3[:,i] = V2[:,i] + noise
    plt.plot(V3[:,i], s, '.', c='C%d' % i, ms=1.5)
plt.xlim((-5, 30))

bexp, uexp, gpexp = [V3[:,i] for i in range(3)]
```




```
In [21]: gp0 = V0[2] / V0[0]
```

```
In [22]: nGrid = 51   # Number of grid points
         b0Vec = np.linspace(.05, 2, nGrid) #cm
         u0Vec = np.linspace(5, 30, nGrid) #cm/s
         Q0Vec = u0Vec * b0Vec**2 #cm3/s
         M0Vec = Q0Vec * u0Vec #cm4/s2

         theta0 = np.pi / 2

         objFn, initialConds = [], []

         sequence = [Q0Vec, M0Vec]

         for (Q0, M0) in list(product(*sequence)):
             F0 = Q0 * gp0
             V0 = [Q0, M0, F0, theta0]

             # Call the 'integrator' function (defined above) to solve
             # the model

             dexp = np.array([gpexp]).ravel(order='C')
             sig_dexp = np.array([])

             sig_V= np.array([ecart_type*np.ones(len(dexp))]).ravel(order='C')
```

```
            objFn.append(objectiveFn2(V0, derivs , p, sexp, dexp, sig_dexp=None, mode='lsq'))
            initialConds.append(V0)

        # Transform initialConds and objFn from lists to arrays,
        # reshaping the latter
        initialConds = np.array(initialConds)
        objFn = np.array(objFn).reshape((nGrid, nGrid))
```

In [23]:
```
s, V = integrator(derivs, V0, p, sexp)

Q, M, F, theta = [V[:,i] for i in range(4)]
b, u, gp = Q / np.sqrt(M), M / Q, F / Q
d = np.array([gp]).ravel(order='C')
    # Some conditions mean that the plume doesn't reach the same altitude as the experi
    # the experimental observable vector to be the same length as the model, or vice ve
if len(d) < len(dexp):
        dexp = dexp[:len(d)]
else:
        d = d[:len(dexp)]
res = dexp - d
res = res[:-1]

kernel = .5 * res.dot(res)
```

0.0.3 Optimisation by fmin

In [24]:
```
Vopt, fopt, Niter, NFCalls, warnFlags, Viter = fmin(objectiveFn2, V0, (derivs,p,sexp, d
```

```
Optimization terminated successfully.
         Current function value: -1.000000
         Iterations: 116
         Function evaluations: 228


/run/user/1000/gvfs/smb-share:server=docobs,share=donnees/Thermographie/MCG/Python Scripts/fumar
  b, u, gp = Q / np.sqrt(M), M / Q, F / Q
/run/user/1000/gvfs/smb-share:server=docobs,share=donnees/Thermographie/MCG/Python Scripts/fumar
  b, u, gp = Q / np.sqrt(M), M / Q, F / Q
```

In [25]:
```
fig = plt.figure(figsize=(2,3))

plt.pcolor(b0Vec, u0Vec,(-objFn))

### Optimal values
ui, bi = np.where(objFn == objFn.min())
bOpt = b0Vec[bi[0]]
uOpt = u0Vec[ui[0]]
```

```
plt.plot(bOpt, uOpt, 'r.', ms=8, label='Values with max. prob.')
plt.xlabel('b0')
plt.ylabel('u0')

#plt.colorbar()
Q=Vopt[0]
M=Vopt[1]
b= Q / np.sqrt(M)
u= M/ Q
plt.plot(b, u, 'co')

plt.plot(b0theoretical, u0theoretical, 'w.', ms=8, label='Supposed values')
plt.plot()
plt.savefig('/home/ovsg/Documents/Stage_Domoison/Rapport/image/bi.png', dpi=300)
```

```
/home/ovsg/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:17: RuntimeWarning: inval
```