plumeAnalyserInversionbase

May 31, 2019

In [1]:
```
from bentPlumeAnalyser import *
from fumarolePlumeModel import *
from scipy.io.matlab import loadmat
from itertools import product

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

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

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

In [2]:
```
exptNo    = 3
plotResults = True

# Read analysed experimental data from file
fname    = './data/ExpPlumes_for_Dai/GCTA_plumeData.xlsx'
exptData = pandas.read_excel(fname, sheet_name='exp%02d' % exptNo)

# Display the first 5 lines of the data frame
exptData.head()
```

Out[2]:

	axisLocn_x	axisLocn_y	distAlongAxis	plumeAngle	plumeWidth
0	-0.082073	0.122375	0.147349	1.510677	0.277366
1	0.079714	1.257440	1.293886	1.449785	0.319537
2	0.024641	2.499398	2.537064	1.176742	0.650992
3	1.443832	4.561191	5.040081	0.783431	1.048499
4	2.855947	5.234923	6.604684	0.565428	1.329422

In [4]:
```
## DEFINE THE DISTANCE ALONG THE AXIS, THE ANGLE AND THE WIDTH OF THE PLUME
# sexp  = exptData.distAlongAxis
# thexp = exptData.plumeAngle
```

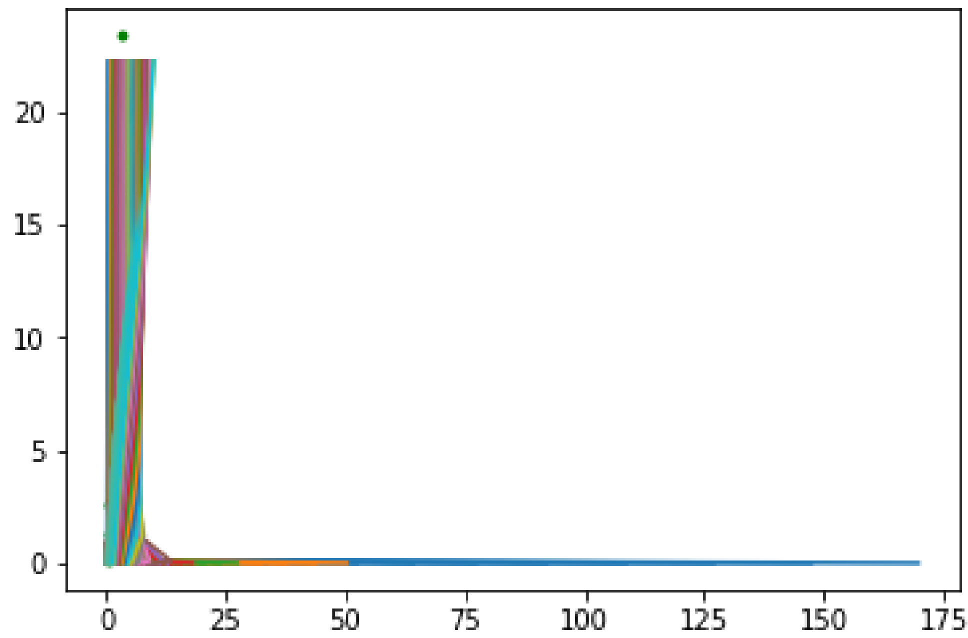
```
    # bexp  = exptData.plumeWidth

    path = './data/ExpPlumes_for_Dai/exp%02d/' % exptNo
    with open(path + 'exp%02d_initGuess.json' % exptNo) as f:
        data = json.load(f)
    p = np.array(data['data'])

    data = np.flipud(loadmat(path + 'gsplume.mat')['gsplume'])
    xexp = loadmat(path + 'xcenter.mat')['xcenter'][0]
    zexp = loadmat(path + 'zcenter.mat')['zcenter'][0]
    Ox, Oz = (xexp[0], zexp[0])
    xexp = (xexp - Ox) / scaleFactor
    zexp = (Oz - zexp) / scaleFactor

    pPixels = p.copy() * scaleFactor
    pPixels[:,0] += Ox
    pPixels[:,1] -= Oz
    pPixels[:,1] *= -1

    # Calculate angle, width and distance along plume and errors
    thexp, sig_thexp = plumeAngle(p[:,0], p[:,1], errors=[1/scaleFactor]*2)
    _, bexp, sig_p, sig_bexp = trueLocationWidth(pPixels, data, errors=[1/scaleFactor])
    sexp     = distAlongPath(p[:,0], p[:,1])
    bexp     /= scaleFactor
    sig_bexp /= scaleFactor
    bexp[0]    = 0.55 / 2
    thexp[0]   = np.pi / 2
```

In [5]:
```
# 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'] == 3]
rhoa0 = GCTA[GCTA['exptNo'] == 3]['rhoa0']
rho0  = GCTA[GCTA['exptNo'] == 3]['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 [6]:
```
# Load initial conditions for a given experiment, run the model for those conditions
# and then compare model and experimental data
fig, ax = plt.subplots()
ax.plot(exptData.plumeWidth, exptData.distAlongAxis, 'g.', label='natural')
```

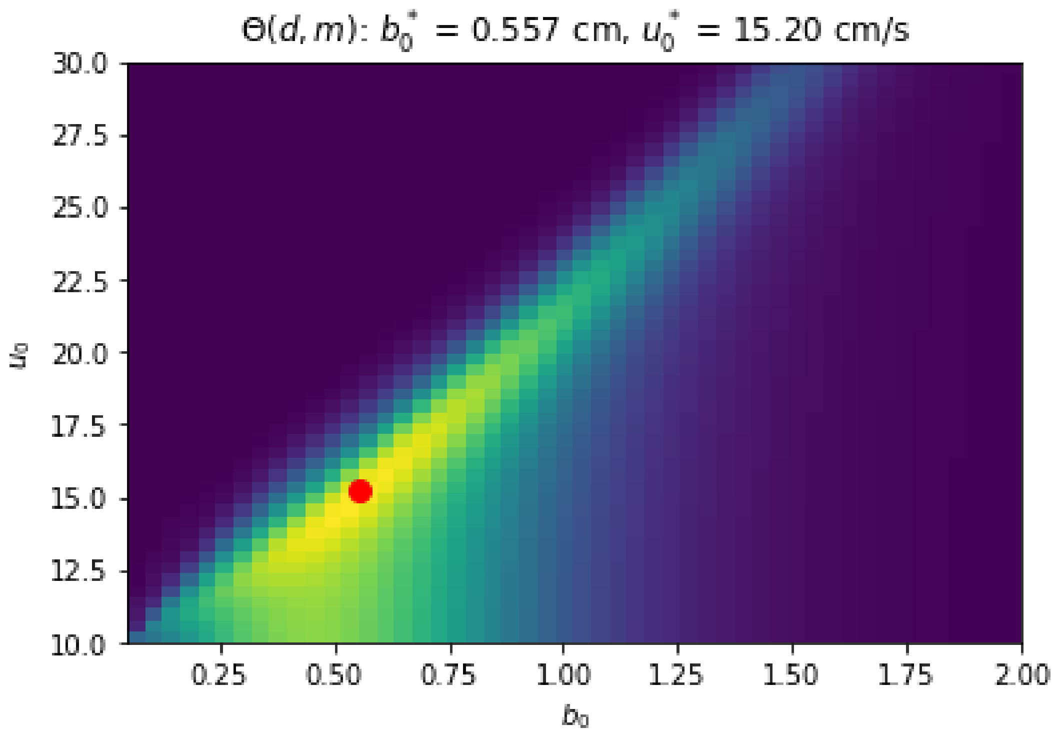
```
V0, p = loadICsParameters(pathname, exptNo, alpha=0.05, beta=0.5, m=2)

#p = (0.05, .5, .012, 2., 4.)
nGrid = 20   # Number of grid points
b0Vec = np.linspace(.05, 2, nGrid) #cm
u0Vec = np.linspace(10, 30, nGrid) #cm/s
Q0Vec = u0Vec * b0Vec**2 #cm3/s
M0Vec = Q0Vec * u0Vec #cm4/s2

theta0 = np.pi / 2

objFn, initialConds = [], []
# Set integration domain and step size
t1    = sexp.max()    # Domain of integration
dsexp = np.diff(sexp) # Discretise the distance along plume axis - use as integration st

sequence = [Q0Vec, M0Vec]

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

    ####################################

    # Initialise an integrator object
    r = ode(derivs).set_integrator('lsoda', nsteps=1e6)
    r.set_initial_value(V0, 0.)
    r.set_f_params(p)

    # Define state vector and axial distance
    V = []     # State vector
    s = []     # Axial distance
    V.append(V0)
    s.append(sexp[0])

    # Define the individual variables - these will be calculated at run time
    Q, M, F, theta = [], [], [], []
    Q.append(Q0)
    M.append(M0)
    F.append(F0)
    theta.append(theta0)

    ####################################

    # Integrate, whilst successful, until the domain size is reached
    ind = 0
    while r.successful() and r.t < t1 and M[-1] >= 0.:
        dt = dsexp[ind]
```

```
        r.integrate(r.t + dt)
        V.append(r.y)
        s.append(r.t)
        Q_, M_, F_, theta_ = r.y
        Q.append(Q_)
        M.append(M_)
        F.append(F_)
        theta.append(theta_)
        ind += 1
    s = np.array(s)
    V = np.array(V)
    Q = np.array(Q)
    M = np.array(M)
    F = np.array(F)

    #######################################

    b  = Q / np.sqrt(M) / np.sqrt(2) # Factor of sqrt{2} to correspond with top-hat mode
    u  = M / Q
    gp = F / Q

    Vexp = np.array(thexp)
    Vsyn = np.array(theta)

    objFn.append(objectiveFn(Vexp, Vsyn, p=p))
    initialConds.append(V0)
    ax.plot(b, s, '-', label='Model %.4f %.4f %.4f' % (Q0, M0, F0))

#ax.legend(loc=5)

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



0.0.1 Plot the objective function for the parameter space that we have calculated

```
In [7]: # Optimal values
        bi, ui = np.where(objFn == objFn.max())
        bOpt = b0Vec[bi[0]]
        uOpt = u0Vec[ui[0]]

        plt.pcolor(b0Vec, u0Vec, objFn)
        hX = plt.xlabel(r'$b_0$')
        hY = plt.ylabel(r'$u_0$')
        hL = plt.title((r'$\Theta(d, m)$: ' +
                        '$b^*_0$ = %.3f cm, $u^*_0$ = %.2f cm/s' % (bOpt, uOpt)))


        plt.plot(bOpt, uOpt, 'ro', ms=8, label='Locn. max. prob.')

        plt.savefig('ProbabilityDistribution_b0u0.png', dpi=300)

        # plt.pcolor(Q0Vec, M0Vec, objFn)
        # hX = plt.xlabel(r'$Q_0$')
        # hY = plt.ylabel(r'$M_0$')
        # hL = plt.title(r'Probability distribution, $\Theta(d, m)$')
```



In [ ]:
```
### Now extract
```

In [2]:
```
# Calculate the model predictions
# Extract the plume parameters from the state vector
Q     = V[:,0]
M     = V[:,1]
F     = V[:,2]
theta = V[:,3]

# Calculate more intuitive plume parameters (width, speed and specific gravity)
b  = Q / np.sqrt(M0)
u  = M / Q
gp = F / Q

xmod, zmod = [0.], [0.]
ds_ = np.diff(s)
for (ds, th) in zip(ds_, theta):
    xmod.append(xmod[-1] + ds * np.cos(th))
    zmod.append(zmod[-1] + ds * np.sin(th))
```

```
NameError                                 Traceback (most recent call last)

<ipython-input-2-36191124e511> in <module>
  1 # Calculate the model predictions
  2 # Extract the plume parameters from the state vector
----> 3 Q     = V[:,0]
  4 M     = V[:,1]
  5 F     = V[:,2]


NameError: name 'V' is not defined
```

```
In [1]: fig, ax = plt.subplots(1, 2, figsize=(8,4))

        # On an image of the experimental plume, show the plume trajectories for
        # 1) GCTA, 2) our initial guess, 3) the model solution
        data, xexp, zexp, extent = loadExptData(exptNo)
        if data.mean() < .5:
            data = 1. - data
        ax[0].imshow(data, extent=extent, cmap=plt.cm.gray)
        ax[0].invert_yaxis()
        ax[0].set_xlabel(r'$x$/[cm]')
        ax[0].set_ylabel(r'$z$/[cm]')
        # 1) GCTA
        ax[0].plot(xexp, zexp, 'r-', label='GCTA', lw=2)
        # 2) our initial guess
        ax[0].plot(exptData.axisLocn_x, exptData.axisLocn_y, 'r--', label='Init. guess', lw=2)
        # 3) the model solution
        ax[0].plot(xmod, zmod, 'g--', label='model', lw=1)
        ax[0].set_xlim((extent[:2]))
        ax[0].legend(loc=2)

        ax[1].plot(res[:,0], sexp, '-', label=r'$(b_{\mathrm{exp}} - b_{\mathrm{mod}})/\sigma_b$
        ax[1].plot(res[:,1], sexp, '-', label=r'$(\theta_{\mathrm{exp}} - \theta_{\mathrm{mod}})
        ax[1].legend(loc=1, fancybox=True, framealpha=.8)
        ax[1].grid()
        ax[1].set_ylabel('Distance along plume axis, s/[cm]')

        ax[1].set_title('Objective fn: %.3f' % objFn)
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-1-6e3806dfeef2> in <module>
----> 1 fig, ax = plt.subplots(1, 2, figsize=(8,4))
```

```

# On an image of the experimental plume, show the plume trajectories for
# 1) GCTA, 2) our initial guess, 3) the model solution
data, xexp, zexp, extent = loadExptData(exptNo)
```

```
NameError: name 'plt' is not defined
```

(*Left*) Image of experimental plume with calculated and estimated trajectories. (*Right*) Difference between experimental (guessed) and model solutions as a function of the distance along the plume axis.

1 To do:

- Make a grid of possible initial conditions and run the model for each case
- Compare these solutions against the experimental data, computing an objective funtion for each case
- Identify which cases produce minima in the objective function

```
In [10]: # Uncomment the following line to transform this notebook into a latex file
         !jupyter nbconvert --to latex plumeAnalyser.ipynb
```

```
[NbConvertApp] Converting notebook plumeAnalyser.ipynb to latex
[NbConvertApp] Support files will be in plumeAnalyser_files/
[NbConvertApp] Making directory plumeAnalyser_files
[NbConvertApp] Writing 36000 bytes to plumeAnalyser.tex
```

```
In [11]: # Now run pdflatex plumeAnalyser from the command line
```