plumeAnalyser

May 31, 2019

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

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 [3]:
```
## DEFINE THE DISTANCE ALONG THE AXIS, THE ANGLE AND THE WIDTH OF THE PLUME
# sexp  = exptData.distAlongAxis
# thexp = exptData.plumeAngle
# bexp  = exptData.plumeWidth
```

```python
        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 [4]:
```python
V0, p = loadICsParameters(pathname, exptNo, alpha=0.05, beta=0.5, m=2)
p
```

Out[4]: (0.05, 0.5, 0.313390044481266, 2, 4.8)

In [5]:
```python
# Load initial conditions for a given experiment, run the model for those conditions
# and then compare model and experimental data
V0, p = loadICsParameters(pathname, exptNo)



# Initialise an integrator object
r = ode(derivs).set_integrator('lsoda', nsteps=1e6)
r.set_initial_value(V0, 0.)
r.set_f_params(p)
V = [V0]     # State vector
s = [0.]     # Axial distance

# Set integration domain and step size
t1 = 30.
dt = .1
```

```
        # Integrate, whilst successful, until the domain size is reached
        while r.successful() and r.t < t1:
            r.integrate(r.t + dt)
            V.append(r.y)
            s.append(r.t)
        s = np.array(s)
        V = np.array(V)
```

```
In [6]: # 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(M)
        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))
```

```
In [7]: # Directly compare the plume width and angle for experiment and model
        # First, the model data has to be interpolated onto the same grid as the experimental da
        from scipy.interpolate import interp1d
        # Plume width
        f = interp1d(s, b)
        bmod = f(sexp)
        # Plume angle
        f = interp1d(s, theta)
        thmod = f(sexp)
```

```
In [8]: # Calculate an "objective function" which measures the misfit between data and model
        # Initially, only use width and angle
        Vexp = np.array([bexp, thexp]).T
        Vmod = np.array([bmod, thmod]).T
        sigV = np.array([sig_bexp, sig_thexp]).T

        # The residual between experimental and model data
        res = (Vexp - Vmod) / sigV
        objFn = objectiveFn(Vexp.ravel(), Vmod.ravel(), np.diagflat(sigV.ravel())**2, p)
```

```
In [9]: res
```

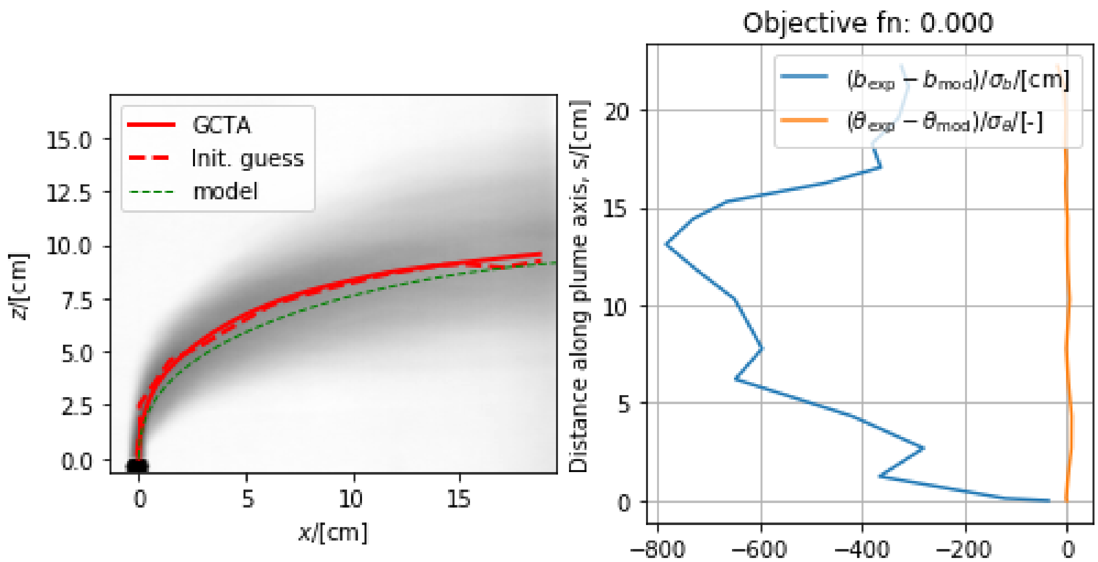
```
Out[9]: array([[-3.532460e+01,  0.000000e+00],
               [-1.201401e+02, -1.249190e+00],
               [-3.656068e+02,  2.738030e+00],
               [-2.794884e+02,  9.362794e+00],
               [-4.192654e+02,  9.905390e+00],
               [-6.470368e+02,  2.901345e+00],
               [-5.948145e+02, -9.002785e-01],
               [-6.486114e+02,  5.924460e+00],
               [-7.211442e+02,  2.334827e+00],
               [-7.825163e+02,  1.110875e+00],
               [-7.291918e+02,  1.427530e+00],
               [-6.633525e+02, -1.087277e+00],
               [-4.723200e+02, -2.372019e+00],
               [-3.632287e+02, -6.876940e-01],
               [-3.798735e+02, -3.013567e+00],
               [-3.282476e+02, -1.781407e+00],
               [-3.087669e+02, -4.467491e+00],
               [-3.229840e+02, -1.798174e+01]])
```

```
In [11]: 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)
         plt.savefig('plumeAnalyser.png', dpi=300)
```



(*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
```