concentration_velocity_fields

December 18, 2017

In [1]:
```
import numpy as np
from scipy.integrate import odeint
from scipy.interpolate import interp1d
from scipy.linalg import solve as linsolve

from matplotlib import pyplot as plt
%matplotlib inline
```

In [2]:
```
def solver_convection(t, cp_0, fpp_0, Schmidt):
    """
    Solve the differential equations.

    cp_0 and fpp_0 are unknown from the physics
    and must be determied numerically.
    Schmidt : Schmidt number = nu / D
    """

    def convection(y, this_t, Schmidt):
        """
        Returns the derivatives from the two diff eq.

        y : Derivatives
        This_t : current time
        Schmidt : Schmidt number = nu / D
        """
        # Unpack with meaningful names
        c, cp, f, fp, fpp = y
        # Our two differential equations
        cpp = - 3 * Schmidt * f * cp
        fppp = - 3 * f * fpp + 2 * fp * fp - c
        # Derivative of the y vector
        dydt = [cp, cpp, fp, fpp, fppp]
        return dydt


    # Initial values from BC
    c_0 = 1
```

```
    f_0 = 0
    fp_0 = 0

    y_0 = [c_0, cp_0, f_0, fp_0, fpp_0]

    sol = odeint(convection, y_0, t, args=(Schmidt,))
    c = sol[:, 0]
    cp = sol[:, 1]
    f = sol[:, 2]
    fp = sol[:, 3]
    fpp = sol[:, 4]

    return c, cp, f, fp, fpp


def solver_perturbation(t, sol, Schmidt):
    """
    Solve the differential equations to guess the parameters.

    sol : c, cp, f, fp, fpp
    Schmidt : Schmidt number = nu / D
    """

    def perturbation_XY(y, this_t, sol, t, Schmidt):
        """
        Differential equations from the perturbation.

        """
        # Unpack with meaningful names
        c, cp, f, fp, fpp = sol

        # Interpolate each function to calculate the value
        #ãat time this_t.
        c_interp = interp1d(t, c, fill_value="extrapolate")
        cp_interp = interp1d(t, cp, fill_value="extrapolate")
        f_interp = interp1d(t, f, fill_value="extrapolate")
        fp_interp = interp1d(t, fp, fill_value="extrapolate")
        fpp_interp = interp1d(t, fpp, fill_value="extrapolate")

        c = c_interp(this_t)
        cp = cp_interp(this_t)
        f = f_interp(this_t)
        fp = fp_interp(this_t)
        fpp = fpp_interp(this_t)

        # Unpack with meaningful names
        cX, cXp, fX, fXp, fXpp, cY, cYp, fY, fYp, fYpp = y
```

```
            fXppp = -3 * (fX * fpp + f * fXpp) + 4 * fp * fXp - cX
            cXpp = - 3 * Schmidt * (fX * cp + f * cXp)

            fYppp = -3 * (fY * fpp + f * fYpp) + 4 * fp * fYp - cY
            cYpp = -3 * Schmidt * (fY * cp + f * cYp)

            dydt = [cXp, cXpp, fXp, fXpp, fXppp,
                    cYp, cYpp, fYp, fYpp, fYppp]
            return dydt

        # Initial values from BC
        cX_0 = 0
        cXp_0 = 0
        fX_0 = 0
        fXp_0 = 0
        fXpp_0 = 1

        cY_0 = 0
        cYp_0 = 1
        fY_0 = 0
        fYp_0 = 0
        fYpp_0 = 0

        y_0 = [cX_0, cXp_0, fX_0, fXp_0, fXpp_0,
               cY_0, cYp_0, fY_0, fYp_0, fYpp_0]

        return odeint(perturbation_XY, y_0, t, args=(sol, t, Schmidt,))
```

In [3]:
```
# Physical parameters
D = 2 * 1e-5
nu = 1.5 * 1e-5
Schmidt = nu / D
print(Schmidt)
```

```
0.75
```

In [4]:
```
# Initial guess
fpp_0 = 1
cp_0 = -1
# Initial value of +infty
t_max = 4
# Number of calculation steps
num_steps = 15
# Number of points for spatial resolution
num_points = 150

for i in range(num_steps):
```

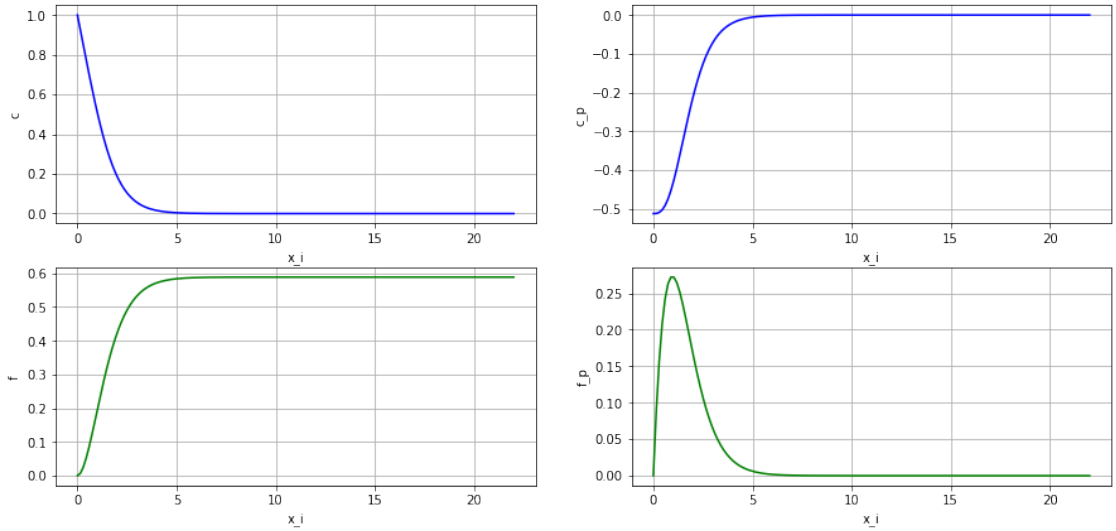
```
        print('Step: ', i)
        # Define a time grid
        t = np.linspace(0, t_max, num_points)
        # Solve the diff eqs
        c, cp, f, fp, fpp = solver_convection(t, cp_0, fpp_0, Schmidt=Schmidt)

        # Find the solution to determine
        #ăthe corrections on the guessed parameters
        sol = solver_perturbation(t, (c, cp, f, fp, fpp), Schmidt=Schmidt)
        # Values at the edge
        cX, cXp, fX, fXp, fXpp, cY, cYp, fY, fYp, fYpp = sol[-1]
        # Calculate the corrections from least square
        # Solve the system of the form A delta = B
        a11 = fXp**2 + cX**2 + fXpp**2 + cXp**2
        a12 = fXp*fYp + cX*cY + fXpp*fYpp + cXp*cYp
        a21 = fXp*fYp + cX*cY + fXpp*fYpp + cXp*cYp
        a22 = fYp**2 + cY**2 + fYpp*2 + cYp**2
        b1 = fp[-1]*fXp + c[-1]*cX + fpp[-1]*fXpp + cp[-1]*cXp
        b2 = fp[-1]*fYp + c[-1]*cY + fpp[-1]*fYpp + cp[-1]*cYp
        A = np.array([[a11, a12], [a21, a22]])
        B = -1 * np.array([[b1], [b2]])
        delta = np.linalg.solve(A, B)
        # and estimate the error
        error = fp[-1]**2 + c[-1]**2 + fpp[-1]**2 + cp[-1]**2
        print('error', error)
        # Apply the correction
        fpp_0 += delta[0]
        cp_0 += delta[1]

        # if small enough error, push forward the virtual infty
        if error < 1e-2:
            t_max += 2
```

```
Step:  0
error 21.1703062632
Step:  1
error 50.4772506122
Step:  2
error 5.24497358027
Step:  3
error 0.28337868711
Step:  4
error 0.0481483401396
Step:  5
error 0.00166378088204
Step:  6
error 0.000290275221938
Step:  7
```

```
error 1.93289696841e-05
Step:  8
error 3.43778643724e-08
Step:  9
error 9.51789683585e-11
Step:  10
error 4.31324670795e-14
Step:  11
error 3.40586660754e-16
Step:  12
error 5.32666578923e-18
Step:  13
error 2.46230163006e-19
Step:  14
error 1.13399766904e-18
```

```
In [5]: fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(15, 7))
        ax[0, 0].plot(t, c, 'b', label='c')
        ax[0, 1].plot(t, cp, 'b', label='c_p')
        ax[1, 0].plot(t, f, 'g', label='f')
        ax[1, 1].plot(t, fp, 'g', label='f_p')
        for a in ax.ravel():
            a.set_xlabel('x_i')
            a.grid()
        ax[0, 0].set_ylabel('c')
        ax[0, 1].set_ylabel('c_p')
        ax[1, 0].set_ylabel('f')
        ax[1, 1].set_ylabel('f_p')
```

```
Out[5]: <matplotlib.text.Text at 0x7fe5e664cfd0>
```



```
In [6]: # Print the value of c'_0
        print(cp[0])
```

```
-0.512123873139
```