13 Contaminante en un flujo de agua subterráneas.

Objetivo. Resolver numéricamente el transporte de un contaminante en un acuífero.

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME PE101922

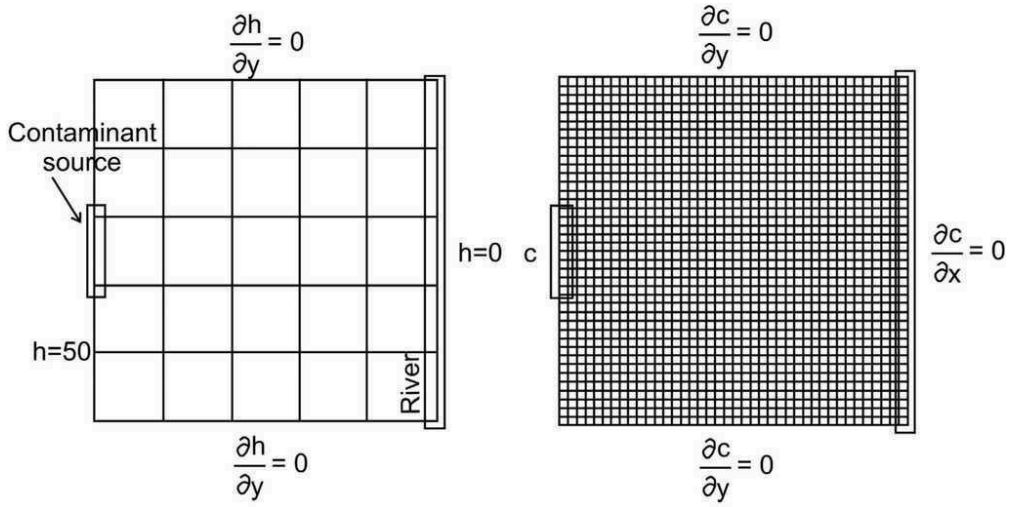
```
import sys
import numpy as np
import matplotlib.pyplot as plt
import macti.visual as mvis

from macti.evaluation import *


print('Python', sys.version)
print(np.__name__, np.__version__)
print(plt.matplotlib.__name__, plt.matplotlib.__version__)
```

```
Python 3.11.6 | packaged by conda-forge | (main, Oct  3 2023, 10:40:35) [GCC 12.3.0]
numpy 1.26.2
matplotlib 3.8.1
```

```
quizz = Quizz('1', 'notebooks', 'local')
```

Modelo conceptual.

Consideremos un acuífero de $Lx = 804.7$ [m] y $Ly = 804.7$ [m] y una fuenta de contaminante localizada en la pared izquierda y acotado por un río en la pared derecha, véase figura. Se considera que el contaminante que se modela es conservativo, es decir, que su concentración no varía al interactuar con el medio y que, por tanto, al atravesar el acuífero mantiene sus propiedades durante todo su recorrido. Se cuenta con un modelo de flujo y transporte de una sola capa, en dos dimensiones. El flujo del agua en el acuífero está en estado de equilibrio. Las condiciones de frontera son las que se muestran en la figura, para la carga hidraúlica h y para la concentración c.



El valor de la carga hidráulica en la pared izquierda es de $h = 50$ [m] y en la pared derecha es de $h = 0$ [m], en las otras paredes se considera no flujo. La fuente de contaminante que está activa durante el periodo de simulación tiene un valor constante de $c = 50$ ppm. En los otros lugares de la frontera se considera no flujo del contaminante. Inicialmente $h = 25$ [m] y $c = 0$ ppm en el interior del dominio.

La porosidad tiene un valor de $\phi = 0.25$, la dispersividad en dirección x tiene un valor de $Dx = 33$ [m] y en dirección y tiene el valor de $Dy = 3.3$ [m]. $K = 21.22$ [m/s] y $S_s = 1.0$

Fuente: Leyva-Suárez, Esther, Herrera, Graciela S., & de la Cruz, Luis M.. (2015). A parallel computing strategy for Monte Carlo simulation using groundwater models. Geofísica internacional, 54(3), 245-254. https://doi.org/10.1016/j.gi.2015.04.020

```
# Parámetros físicos
K = 1.0 # 21.22  # Conductividad
Dx = 1.0 # 33.0
Dy = 1.0 # 3.3
ϕ = 1.0 # 0.25
Lx = 1.0 # 804.7  # Longitud del dominio en dirección x
Ly = 1.0 # 804.7  # Longitud del dominio en dirección y

print('Parámetros físicos' + '\n' + 40*'-')
print('Conductividad K = {}'.format(K))
print('Conductividad (Dx, Dy) = ({}, {})'.format(Dx, Dy))
print('Porosidad ϕ = {}'.format(ϕ))
print('Longitud en x = {} | Longitud en y = {}'.format(Lx,Ly))
```

```
Parámetros físicos
----------------------------------------
Conductividad K = 1.0
Conductividad (Dx, Dy) = (1.0, 1.0)
Porosidad ϕ = 1.0
Longitud en x = 1.0 | Longitud en y = 1.0
```

Modelo matemático. Para este modelo usaremos las ecuaciones de flujo y transporte acopladas por la ley de Darcy para describir la evolución de la pluma del contaminante. Estas ecuaciones se van a resolver para las cargas

hidráulicas y las concentraciones del contaminante y se escriben como sigue:

$$S_s \frac{\partial h}{\partial t} = K \left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} \right) \tag{1}$$

$$V = -K \nabla h \tag{2}$$

$$\phi \frac{\partial c}{\partial t} + V \cdot \nabla c = Dx \frac{\partial^2 c}{\partial x^2} + Dy \frac{\partial^2 c}{\partial y^2} \tag{3}$$

donde S_s es el almacenamiento específico, K es la conductividad hidráulica, h la carga hidráulica, c la concentración del soluto, $D = (Dx, Dy)$ es la dispersión hidrodinámica, $V = (Vx, Vy)$ la velocidad de poro y la ϕ porosidad efectiva.

La ecuación de flujo (1) describe el flujo del agua a través del acuífero, la ecuación de transporte (3) describe los cambios en la concentración del contaminante a través del tiempo para un soluto conservativo. La ley de Darcy (2) acopla las ecuaciones (1) y (3) y con ella se calcula la velocidad de poro del agua subterránea utilizando las cargas y la conductividad hidráulica.

Las condiciones de frontera, de acuerdo con la figura de la sección anterior, son las siguientes:

$$\begin{aligned} h(t,x,y) &= 50\,[m] && \text{para} \quad x = 0, \quad \forall t \\ h(t,x,y) &= 0\,[m] && \text{para} \quad x = Lx, \quad \forall t \\ \frac{\partial h(t,x,y)}{\partial y} &= 0 && \text{para} \quad y = 0, Ly, \quad \forall t \end{aligned}$$

$$\begin{aligned} c(t,x,y) &= 50\,[ppm] && \text{para} \quad x = 0, \quad y \in [\tfrac{3}{8}Ly, \tfrac{5}{8}Ly], \quad \forall t \\ \frac{\partial c(t,x,y)}{\partial y} &= 0 && \text{para la frontera restante}, \quad \forall t \end{aligned}$$

Modelo numérico.

El dominio se discretiza en una malla de 40×40. Se va a simular durante 48 pasos de tiempo, con un paso de 15.2083 días.

La forma discreta del modelo matemático, ecuaciones (1), (2) y (3), usando diferencias finitas de segundo orden es la siguiente:

$$h_{i,j}^{n+1} = h_{i,j}^n + \frac{\delta_t K_{i,j}}{\delta^2} \left(h_{i+1,j}^n + h_{i-1,j}^n + h_{i,j+1}^n + h_{i,j-1}^n - 4h_{i,j}^n \right)$$

$$(Vx_{i,j}, Vy_{i,j}) = -\frac{K_{i,j}}{2\delta} (h_{i+1,j} - h_{i-1,j}, h_{i,j+1} - h_{i,j-1})$$

$$c_{i,j}^{n+1} = c_{i,j}^n + \frac{\delta_t Dx_{i,j}}{\delta^2 \phi} \left(c_{i+1,j}^n - 2c_{i,j}^n + c_{i-1,j}^n \right) + \frac{\delta_t Dx_{i,j}}{\delta^2 \phi} \left(c_{i,j+1}^n - 2c_{i,j}^n + c_{i,j-1}^n \right) - \frac{\delta_t Vx_{i,j}}{2\delta\phi} (c_{i+1,j} - c_{i-1,j}) - \frac{\delta_t Vy_{i,j}}{2\delta\phi} (c_{i,j+1} - c_{i,j-1})$$

donde δ representa el tamaño de las celdas en ambas direcciones y δ_t el paso de tiempo.

```
# Parámetros numéricos
Nx = 28 # Número de incógnitas en dirección x
Ny = 28 # Número de incógnitas en dirección y
δ = Lx / (Nx+1) # Espaciamiento entre los puntos de la rejilla
δt = 0.001     # Paso de tiempo
N = (Nx + 2)* (Ny + 2) # Número total de puntos en la rejilla

print('\nParámetros numéricos' + '\n' + 40*'-')
print('Nodos en x = {} | Nodos en y = {}'.format(Nx+2,Ny+2))
print('δ = {} | δt = {}'.format(δ, δt))
```

```
Parámetros numéricos
----------------------------------------
Nodos en x = 30 | Nodos en y = 30
δ = 0.034482758620689655 | δt = 0.001
```

Modelo computacional.

Algoritmo. Los pasos a seguir son los siguientes.

1. Definir los parámetros físicos y numéricos del problema: (ya se definiron antes)

```
print('Parámetros físicos' + '\n' + 40*'-')
print('Conductividad K = {}'.format(K))
print('Conductividad (Dx, Dy) = ({}, {})'.format(Dx, Dy))
print('Porosidad ϕ = {}'.format(ϕ))
print('Longitud en x = {} | Longitud en y = {}'.format(Lx,Ly))
print('\nParámetros numéricos' + '\n' + 40*'-')
print('Nodos en x = {} | Nodos en y = {}'.format(Nx+2,Ny+2))
print('δ = {} | δt = {}'.format(δ, δt))
```

```
Parámetros físicos
----------------------------------------
Conductividad K = 1.0
Conductividad (Dx, Dy) = (1.0, 1.0)
Porosidad ϕ = 1.0
Longitud en x = 1.0 | Longitud en y = 1.0

Parámetros numéricos
----------------------------------------
Nodos en x = 30 | Nodos en y = 30
δ = 0.034482758620689655 | δt = 0.001
```

2. Definir la rejilla donde se hará el cálculo (malla):

Ejercicio 1. Calcule los arreglos `x` y `y` correctos para generar la malla.

```
### BEGIN SOLUTION
x = np.linspace(0,Lx,Nx+2) # Arreglo con las coordenadas en x
```

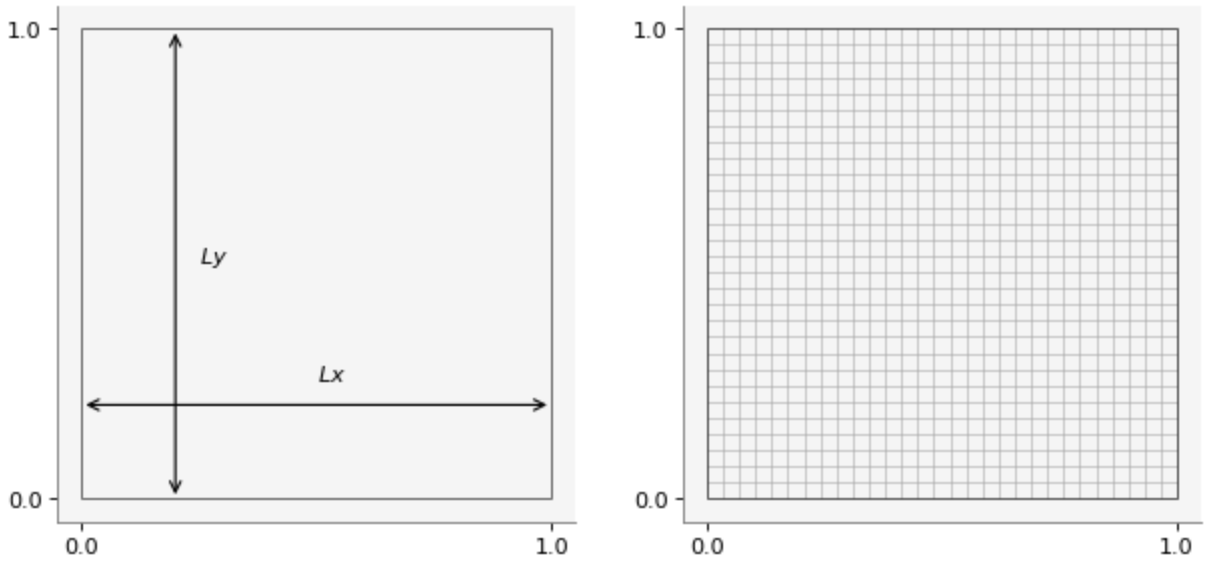
```
        y = np.linspace(0,Ly,Ny+2) # Arreglo con las coordenadas en y

        file_answer = FileAnswer()
        file_answer.write('1', x, 'Revisa la definición de las coordenadas en dirección x
        file_answer.write('2', y, 'Revisa la definición de las coordenadas en dirección y
        ### END SOLUTION

        xg, yg = np.meshgrid(x,y,indexing='ij', sparse=False) # Creamos la rejilla para u
```

```
Creando el directorio :/home/jovyan/macti_notes/notebooks/.ans/RAUGM2023/
Respuestas y retroalimentación almacenadas.
```

```
        quizz.eval_numeric('1', x)
        quizz.eval_numeric('2', y)
```

```
----------------------------------------
1 | Tu resultado es correcto.
----------------------------------------
----------------------------------------
2 | Tu resultado es correcto.
----------------------------------------
```

```
        vis = mvis.Plotter(1,2,[dict(aspect='equal'), dict(aspect='equal')], dict(figsize

        vis.draw_domain(1, xg, yg)
        vis.plot_mesh2D(2, xg, yg)
        vis.plot_frame(2, xg, yg)
```



3. Definir las condiciones iniciales y de frontera:

Para el caso de la concentración, necesitamos definir la fuente de contaminante en la pared izquierda:

Ejercicio 2. Calcule el lugar donde se debe aplicar la condición de frontera distinta de cero para la concentración c. Recuerde que el intervalo es $y \in [\frac{3}{8}Ly, \frac{5}{8}Ly]$.

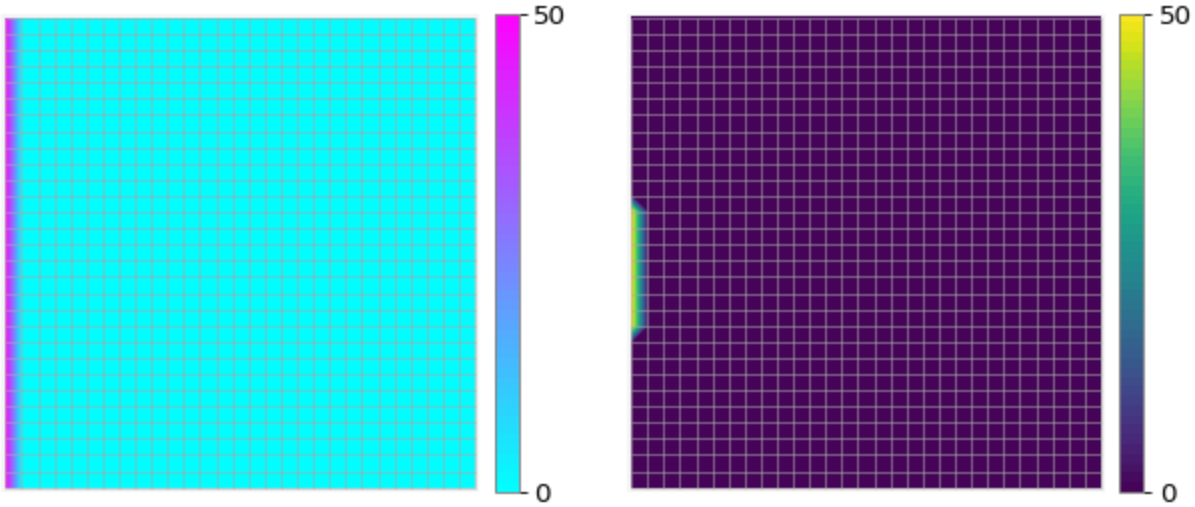
```
        h = np.zeros((Nx+2, Ny+2))
        h[0,:]    = 50  # Pared izquierda
        h[Nx+1,:] = 0   # Pared derecha

        c = np.zeros((Nx+2, Ny+2))
        c[0,:]    = 0  # Pared izquierda
        c[Nx+1,:] = 0   # Pared derecha
        c[:,0]    = 0  # Pared inferior
        c[:,Ny+1] = 0   # Pared superior

        ### BEGIN SOLUTION
        N1 = int((Ly * 3.0 / 8.0) / δ)
        N2 = int((Ly * 5.0 / 8.0) / δ)

        file_answer.write('3', N1, 'Revisa el cálculo de límite inferior del intervalo')
        file_answer.write('4', N2, 'Revisa el cálculo de límite superior del intervalo')
        ### END SOLUTION
        c[0, N1:N2]    = 50  # Pared izquierda

        print('N1 = {}'.format(N1))
        print('N2 = {}'.format(N2))
```

```
N1 = 10
N2 = 18
```

```
        quizz.eval_numeric('3', N1)
        quizz.eval_numeric('4', N2)
```

```
----------------------------------------
3 | Tu resultado es correcto.
----------------------------------------
----------------------------------------
4 | Tu resultado es correcto.
----------------------------------------
```

```
        vis = mvis.Plotter(1,2,[dict(aspect='equal'), dict(aspect='equal')], dict(figsize

        cax1 = vis.set_canvas(1,Lx,Ly)
        c_h = vis.contourf(1, xg, yg, h, levels=50, cmap='cool')
        vis.fig.colorbar(c_h, cax=cax1, ticks = [h.min(), h.max()], shrink=0.5, orientati
        vis.plot_mesh2D(1, xg, yg)

        cax2 = vis.set_canvas(2,Lx,Ly)
        c_c = vis.contourf(2, xg, yg, c, levels=50, cmap='viridis')
        vis.fig.colorbar(c_c, cax=cax2, ticks = [c.min(), c.max()], shrink=0.5, orientati
        vis.plot_mesh2D(2, xg, yg)
```



4. Implementar el algoritmo de solución:

$$h_{i,j}^{n+1} = h_{i,j}^{n} + \frac{\delta_t K_{i,j}}{\delta^2}\left(h_{i+1,j}^{n} + h_{i-1,j}^{n} + h_{i,j+1}^{n} + h_{i,j-1}^{n} - 4h_{i,j}^{n}\right)$$

$$(Vx_{i,j}, Vy_{i,j}) = -\frac{K_{i,j}}{2\delta}(h_{i+1,j} - h_{i-1,j}, h_{i,j+1} - h_{i,j-1})$$

$$c_{i,j}^{n+1} = c_{i,j}^{n} + \frac{\delta_t Dx_{i,j}}{\delta^2}\left(c_{i+1,j}^{n} - 2c_{i,j}^{n} + c_{i-1,j}^{n}\right) + \frac{\delta_t Dx_{i,j}}{\delta^2}\left(c_{i,j+1}^{n} - 2c_{i,j}^{n} + c_{i,j-1}^{n}\right) - \frac{\delta_t Vx_{i,j}}{2\delta}(c_{i+1,j} - c_{i-1,j}) - \frac{\delta_t Vy_{i,j}}{2\delta}(c_{i,j+1} - c_{i,j-1})$$

```
δt = 0.0001
r = δt / δ**2
s = 1.0 / 2*δ
t = δt / 2*δ

h_new = h.copy()
c_new = c.copy()

tolerancia = 1.0e-3
error = 1.0
error_lista = []

Vx = np.zeros((Nx+2, Ny+2))
Vy = Vx.copy()
```

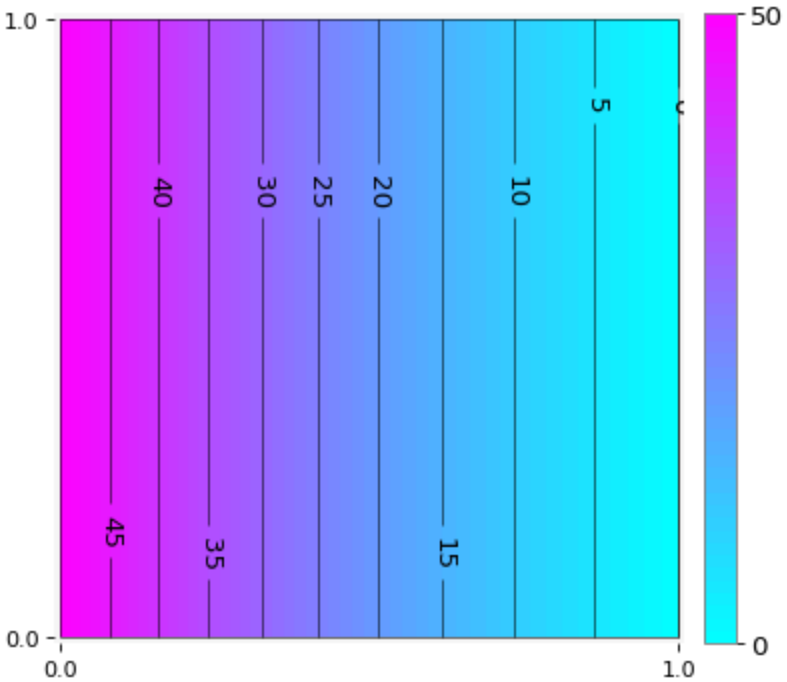
4.1 Hacemos una prueba resolviendo primero la ecuación de flujo

Ejercicio 3. Implemente la fórmula en diferencias para la carga hidráulica h.

```
iteraciones_max = 1000
iteraciones = 0
while(error > tolerancia and iteraciones < iteraciones_max):
    iteraciones += 1
```

```
        for i in range(1,Nx+1):
            for j in range(1,Ny+1):
                if j == 1:  # No flujo en la pared inferior
                    h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j+1] - 3
                if j == Ny: # No flujo en la pared superior
                    h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j-1] - 3
                else:
                    h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j+1] + h

            # Condición de frontera de no flujo
            h_new[:,0] = h_new[:,1]
            h_new[:,Ny+1] = h_new[:,Ny]

            # Actualización de la carga hidráulica
            h[:] = h_new[:]

        #    print(iteraciones, sep=' ', end= ' ')
```

```
        vis = mvis.Plotter(1,1,[dict(aspect='equal')])

        cax1 = vis.set_canvas(1,Lx,Ly)
        c_h = vis.contourf(1, xg, yg, h, levels=50, cmap='cool')
        cl_h = vis.contour(1, xg, yg, h, levels=10, colors='k', linewidths=0.5)
        plt.clabel(cl_h, inline=True, fontsize=12.0)
        vis.fig.colorbar(c_h, cax=cax1, ticks = [h.min(), h.max()], shrink=0.5, orientati
        vis.plot_frame(1, xg, yg)
```



4.2 Calculamos ahora la velocidad con la h antes calculada

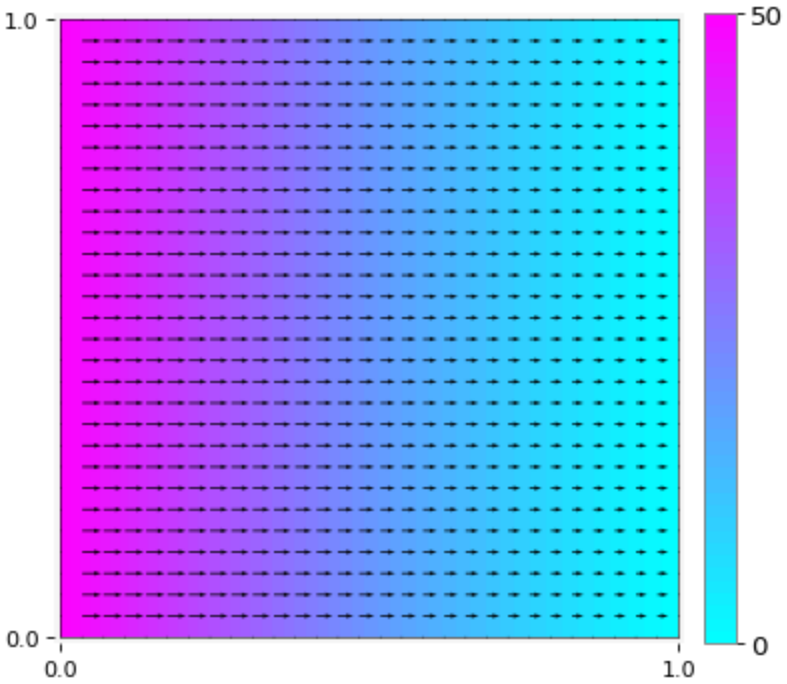
Ejercicio 4. Calcule la velocidad con la fórmula correspondiente.

```
        ### BEGIN SOLUTION
        for i in range(1,Nx+1):
            for j in range(1,Ny+1):
                Vx[i,j] = -K * s * (h[i+1,j] - h[i-1,j])
                Vy[i,j] = -K * s * (h[i,j+1] - h[i,j-1])

        file_answer.write('5', Vx,'')
        file_answer.write('6', Vy,'')
        ### END SOLUTION
```

```
        quizz.eval_numeric('5', Vx)
        quizz.eval_numeric('6', Vy)
```

```
----------------------------------------
5 | Tu resultado es correcto.
----------------------------------------
----------------------------------------
6 | Tu resultado es correcto.
----------------------------------------
```

```
        vis = mvis.Plotter(1,1,[dict(aspect='equal')])

        cax1 = vis.set_canvas(1,Lx,Ly)
        c_h = vis.contourf(1, xg, yg, h, levels=50, cmap='cool')
        vis.fig.colorbar(c_h, cax=cax1, ticks = [h.min(), h.max()], shrink=0.5, orientati
        vis.plot_frame(1, xg, yg)
        vis.quiver(1, xg, yg, Vx, Vy, scale=2.5)
        vis.show()
```



4.3 Implementamos el algoritmo completo

$$h_{i,j}^{n+1} = h_{i,j}^{n} + \frac{\delta_t K_{i,j}}{\delta^2}\left(h_{i+1,j}^{n} + h_{i-1,j}^{n} + h_{i,j+1}^{n} + h_{i,j-1}^{n} - 4h_{i,j}^{n}\right)$$

$$(Vx_{i,j}, Vy_{i,j}) = -\frac{K_{i,j}}{2\delta}(h_{i+1,j} - h_{i-1,j}, h_{i,j+1} - h_{i,j-1})$$

$$c_{i,j}^{n+1} = c_{i,j}^{n} + \frac{\delta_t Dx_{i,j}}{\delta^2\phi}\left(c_{i+1,j}^{n} - 2c_{i,j}^{n} + c_{i-1,j}^{n}\right) + \frac{\delta_t Dx_{i,j}}{\delta^2\phi}\left(c_{i,j+1}^{n} - 2c_{i,j}^{n} + c_{i,j-1}^{n}\right) - \frac{\delta_t Vx_{i,j}}{2\delta\phi}(c_{i+1,j} - c_{i-1,j}) - \frac{\delta_t Vy_{i,j}}{2\delta\phi}(c_{i,j+1} - c_{i,j-1})$$

Ejercicio 5. Complete el cálculo de la concentración c en el código que sigue.

```
h = np.ones((Nx+2, Ny+2)) * 25
h[0,:]    = 50  # Pared izquierda
h[Nx+1,:] = 0   # Pared derecha

c = np.zeros((Nx+2, Ny+2))
c[0,:]    = 0  # Pared izquierda
c[Nx+1,:] = 0   # Pared derecha
c[:,0]    = 0  # Pared inferior
c[:,Ny+1] = 0   # Pared superior

c[0, N1:N2]    = 50  # Pared izquierda

iteraciones_max = 1000
iteraciones = 0
while(error > tolerancia and iteraciones < iteraciones_max):
    iteraciones += 1
    for i in range(1,Nx+1):
        for j in range(1,Ny+1):
            if j == 1:  # No flujo
                h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j+1] - 3
            if j == Ny: # No flujo
                h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j-1] - 3
            else:
                h_new[i,j] = h[i,j] + K * r * (h[i+1,j] + h[i-1,j] + h[i,j+1] + h

    h_new[:,0] = h_new[:,1]
    h_new[:,Ny+1] = h_new[:,Ny]

    for i in range(1,Nx+1):
        for j in range(1,Ny+1):
            Vx[i,j] = -K * s * (h[i+1,j] - h[i-1,j])
            Vy[i,j] = -K * s * (h[i,j+1] - h[i,j-1])

    for i in range(1,Nx+1):
        for j in range(1,Ny+1):
            if j == 1:  # No flujo
```

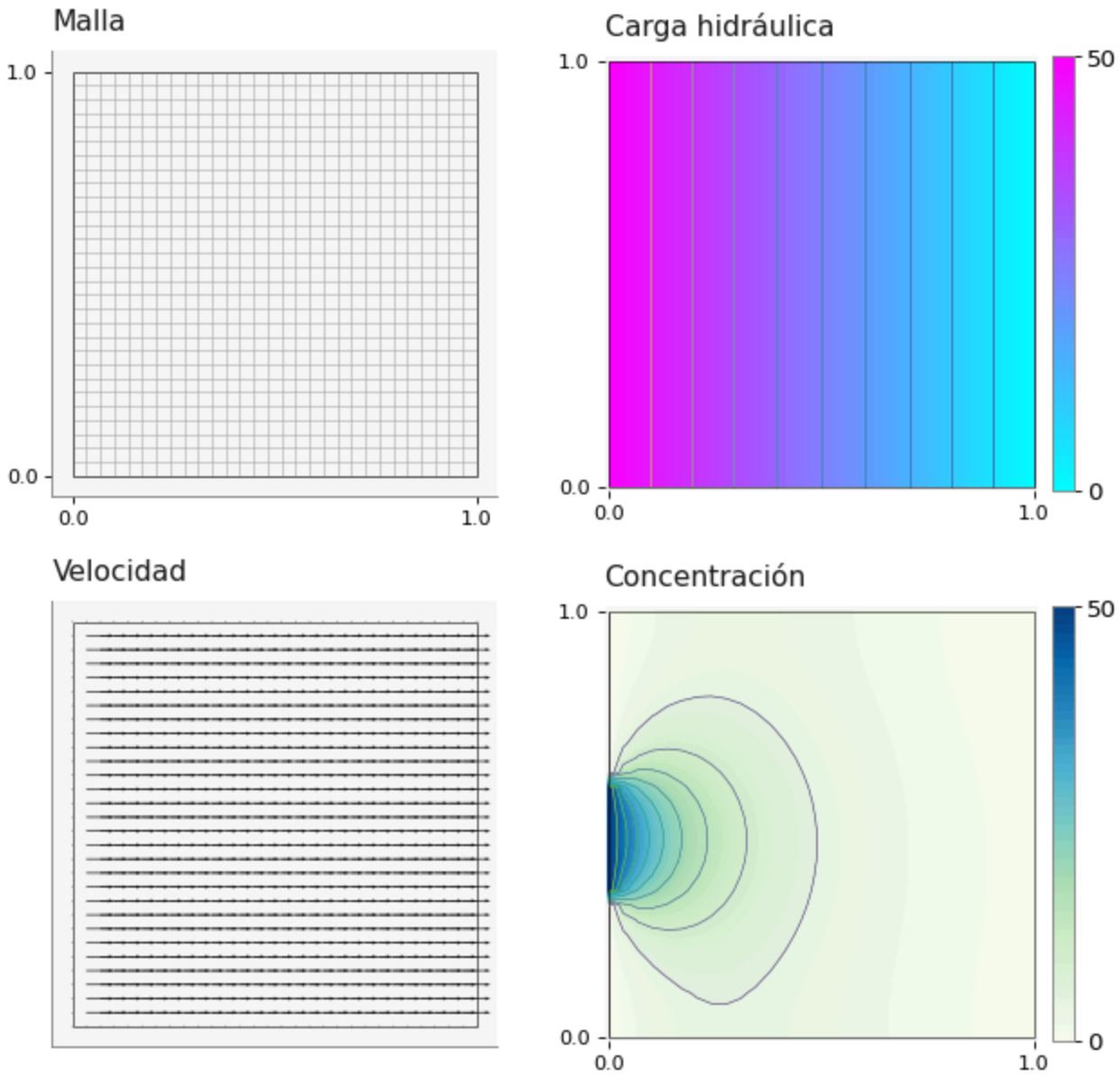
```
                    c_new[i,j] = c[i,j] + Dx * r * (c[i+1,j] - 2*c[i,j]+ c[i-1,j]) /
                                        + Dy * r * (c[i,j+1] - c[i,j]) / ϕ \
                                        - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
                                        - t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ
                if j == Ny: # No flujo
                    c_new[i,j] = c[i,j] + Dx * r * (c[i+1,j] - 2*c[i,j]+ c[i-1,j]) /
                                        + Dy * r * (c[i,j-1] - c[i,j]) / ϕ \
                                        - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
                                        - t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ
                if i == Nx: # No flujo
                    c_new[i,j] = c[i,j] + Dx * r * (c[i-1,j] - c[i,j]) / ϕ \
                                        + Dy * r * (c[i,j+1] - 2*c[i,j]+ c[i,j-1]) /
                                        - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
                                        + t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ
                else:
                    c_new[i,j] = c[i,j] + Dx * r * (c[i+1,j] - 2*c[i,j]+ c[i-1,j]) /
                                    + Dy * r * (c[i,j+1] - 2*c[i,j]+ c[i,j-1]) / ϕ\
                                    - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ\
                                    - t * Vy[i,j] * (c[i,j+1] - c[i,j-1])/ ϕ

            c_new[:,0] = c_new[:,1]
            c_new[:,Ny+1] = c_new[:,Ny]
            c_new[Nx+1:] = c_new[Nx,:]

            error = np.linalg.norm(h_new - h)
            error_lista.append(error)
            h[:] = h_new[:]
            c[:] = c_new[:]

            print(iteraciones, sep=' ', end= ' ')
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205
206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249
250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293
294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337
338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381
382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403
404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425
426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447
```

```
448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469
470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491
492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513
514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535
536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557
558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579
580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601
602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623
624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645
646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667
668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689
690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711
712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733
734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755
756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777
778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799
800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821
822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843
844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865
866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887
888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909
910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931
932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953
954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975
976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997
998 999 1000
```

```
        ax1 = dict(aspect='equal', title='Malla')
        ax2 = dict(aspect='equal', title='Carga hidráulica')
        ax3 = dict(aspect='equal', title='Velocidad')
        ax4 = dict(aspect='equal', title='Concentración')

        vis = mvis.Plotter(2,2,[ax1, ax2, ax3, ax4],
                           dict(figsize=(8,8)))

        vis.plot_mesh2D(1, xg, yg)
        vis.plot_frame(1, xg, yg)

        cax3 = vis.set_canvas(2,Lx,Ly)
        c_h = vis.contourf(2, xg, yg, h, levels=50, cmap='cool')
        vis.fig.colorbar(c_h, cax=cax3, ticks = [h.min(), h.max()], shrink=0.5, orientati
        vis.contour(2, xg, yg, h, levels=10, linewidths=0.5)
        vis.plot_frame(2, xg, yg)

        vis.plot_frame(3, xg, yg)
        vis.quiver(3, xg, yg, Vx, Vy, scale=1)

        cax4 = vis.set_canvas(4,Lx,Ly)
        c_c = vis.contourf(4, xg, yg, c, levels=50, cmap='GnBu')
        vis.fig.colorbar(c_c, cax=cax4, ticks = [c.min(), c.max()], shrink=0.5, orientati
```

```
vis.contour(4, xg, yg, c, levels=10, linewidths=0.5)
vis.plot_frame(4, xg, yg)

vis.show()
```

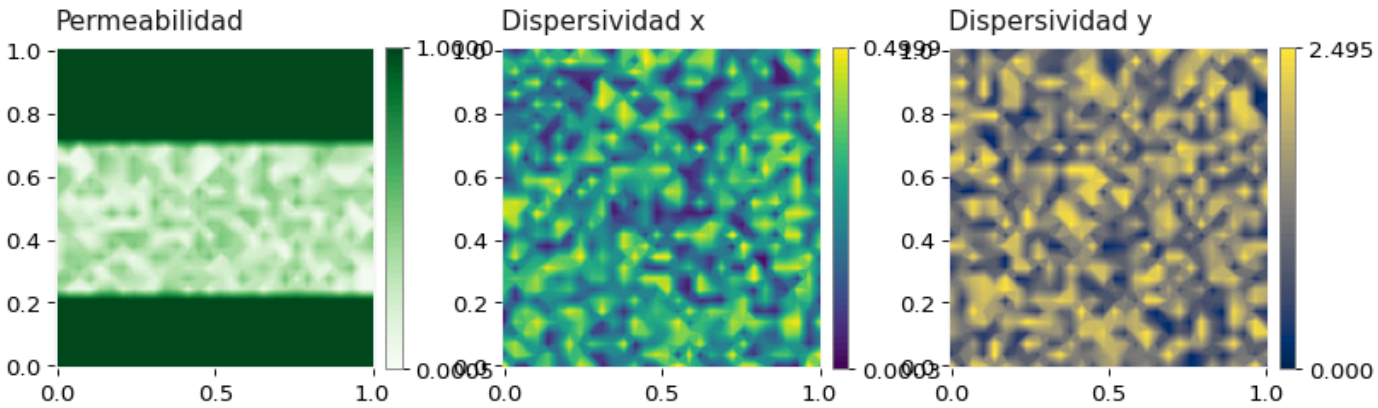


¿Podría hacer el mismo cálculo con una permeabilidad hidráulica y dipersividad variable en el dominio de estudio?

```
K = np.ones((Nx+2, Ny+2))

nn1 = int(Ny*0.25)
nn2 = int(Ny*.75)
print(nn1, nn2, nn2-nn1)
K[:, nn1:nn2] = np.random.rand(Nx+2, nn2-nn1) * 0.5

Dx = np.random.rand(Nx+2, Ny+2) * 0.5
Dy = np.random.rand(Nx+2, Ny+2) * 2.5
```

```
7 21 14
```

```
ax1 = dict(aspect='equal', title='Permeabilidad')
ax2 = dict(aspect='equal', title='Dispersividad x')
ax3 = dict(aspect='equal', title='Dispersividad y')

vis = mvis.Plotter(1,3,[ax1, ax2, ax3],
                   dict(figsize=(10,8)))

cax1 = vis.set_canvas(1,Lx,Ly)
p_v = vis.contourf(1, xg, yg, K, levels=50, cmap='Greens')
vis.fig.colorbar(p_v, cax=cax1, ticks = [K.min(), K.max()], shrink=0.5, orientati

cax2 = vis.set_canvas(2,Lx,Ly)
d_x = vis.contourf(2, xg, yg, Dx, levels=50, cmap='viridis')
vis.fig.colorbar(d_x, cax=cax2, ticks = [Dx.min(), Dx.max()], shrink=0.5, orienta

cax3 = vis.set_canvas(3,Lx,Ly)
d_y = vis.contourf(3, xg, yg, Dy, levels=50, cmap='cividis')
vis.fig.colorbar(d_y, cax=cax3, ticks = [Dy.min(), Dy.max()], shrink=0.5, orienta

vis.show()
```



13.0.1 Ejercicio 6.

Copie y modifique el código que calcula toda la solución, de tal manera que tome en cuenta la permeabilidad hidráulica y la dispersividad variables.

```
### BEGIN SOLUTION
h = np.ones((Nx+2, Ny+2)) * 25
h[0,:]    = 50  # Pared izquierda
h[Nx+1,:] = 0   # Pared derecha

c = np.zeros((Nx+2, Ny+2))
c[0,:]    = 0  # Pared izquierda
c[Nx+1,:] = 0   # Pared derecha
c[:,0]    = 0  # Pared inferior
```

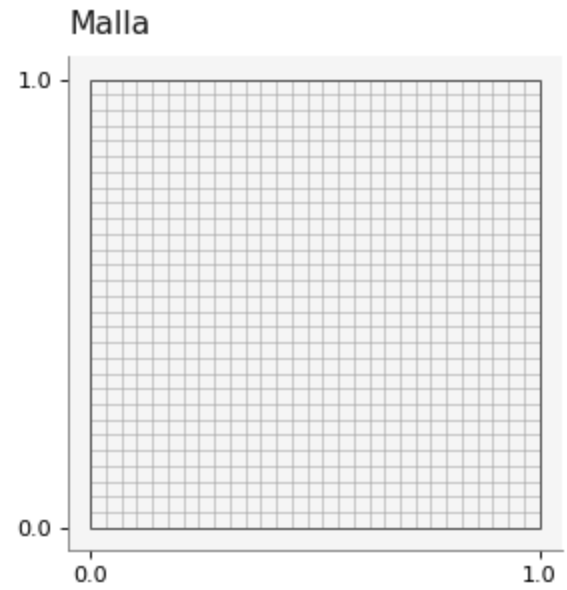
```
c[:,Ny+1] = 0   # Pared superior

c[0, N1:N2]    = 50  # Pared izquierda

iteraciones_max = 1000
iteraciones = 0
while(error > tolerancia and iteraciones < iteraciones_max):
    iteraciones += 1
    for i in range(1,Nx+1):
        for j in range(1,Ny+1):
            if j == 1:  # No flujo
                h_new[i,j] = h[i,j] + K[i,j] * r * (h[i+1,j] + h[i-1,j] + h[i,j+1
            if j == Ny: # No flujo
                h_new[i,j] = h[i,j] + K[i,j] * r * (h[i+1,j] + h[i-1,j] + h[i,j-1
            else:
                h_new[i,j] = h[i,j] + K[i,j] * r * (h[i+1,j] + h[i-1,j] + h[i,j+1

    h_new[:,0] = h_new[:,1]
    h_new[:,Ny+1] = h_new[:,Ny]

    for i in range(1,Nx+1):
        for j in range(1,Ny+1):
            Vx[i,j] = -K[i,j] * s * (h[i+1,j] - h[i-1,j])
            Vy[i,j] = -K[i,j] * s * (h[i,j+1] - h[i,j-1])

    for i in range(1,Nx+1):
        for j in range(1,Ny+1):
            if j == 1:  # No flujo
                c_new[i,j] = c[i,j] + Dx[i,j] * r * (c[i+1,j] - 2*c[i,j]+ c[i-1,j
                                    + Dy[i,j] * r * (c[i,j+1] - c[i,j]) / ϕ \
                                    - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
                                    - t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ
            if j == Ny: # No flujo
                c_new[i,j] = c[i,j] + Dx[i,j] * r * (c[i+1,j] - 2*c[i,j]+ c[i-1,j
                                    + Dy[i,j] * r * (c[i,j-1] - c[i,j]) / ϕ \
                                    - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
                                    - t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ
            if i == Nx: # No flujo
                c_new[i,j] = c[i,j] + Dx[i,j] * r * (c[i-1,j] - c[i,j]) / ϕ \
                                    + Dy[i,j] * r * (c[i,j+1] - 2*c[i,j]+ c[i,j-1
                                    - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ \
                                    + t * Vy[i,j] * (c[i,j+1] - c[i,j-1]) / ϕ
            else:
                c_new[i,j] = c[i,j] + Dx[i,j] * r * (c[i+1,j] - 2*c[i,j]+ c[i-1,j
                                + Dy[i,j] * r * (c[i,j+1] - 2*c[i,j]+ c[i,j-1]) /
                                - t * Vx[i,j] * (c[i+1,j] - c[i-1,j]) / ϕ\
                                - t * Vy[i,j] * (c[i,j+1] - c[i,j-1])/ ϕ

    c_new[:,0] = c_new[:,1]
    c_new[:,Ny+1] = c_new[:,Ny]
    c_new[Nx+1:] = c_new[Nx,:]
```

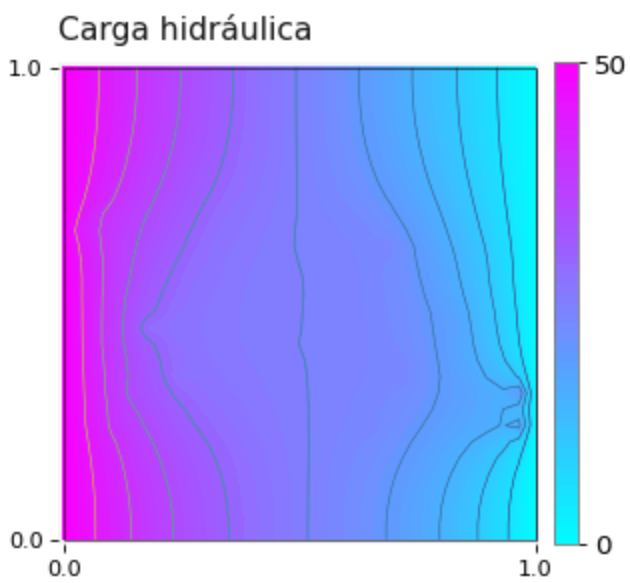
```
            error = np.linalg.norm(h_new - h)
            error_lista.append(error)
            h[:] = h_new[:]
            c[:] = c_new[:]

        #    print(iteraciones, sep=' ', end= ' ')
        ### END SOLUTION
```

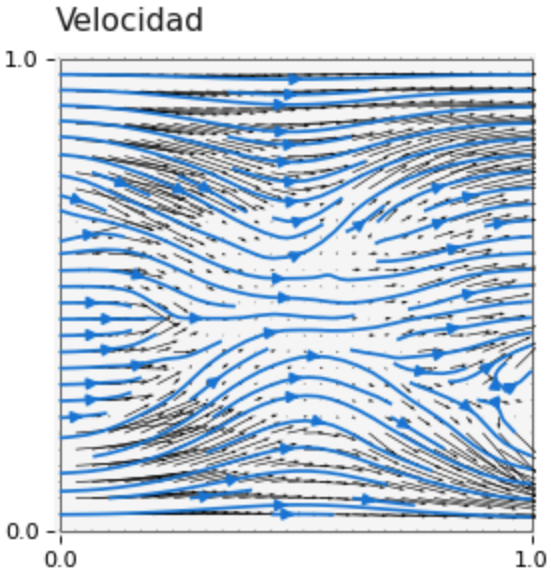
```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93
94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205
206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227
228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249
250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293
294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337
338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381
382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403
404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425
426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469
470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491
492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513
514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535
536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557
558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579
580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601
602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623
624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645
646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667
668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689
690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711
712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733
734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755
756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777
778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799
800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821
822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843
844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865
866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887
888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909
910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931
```

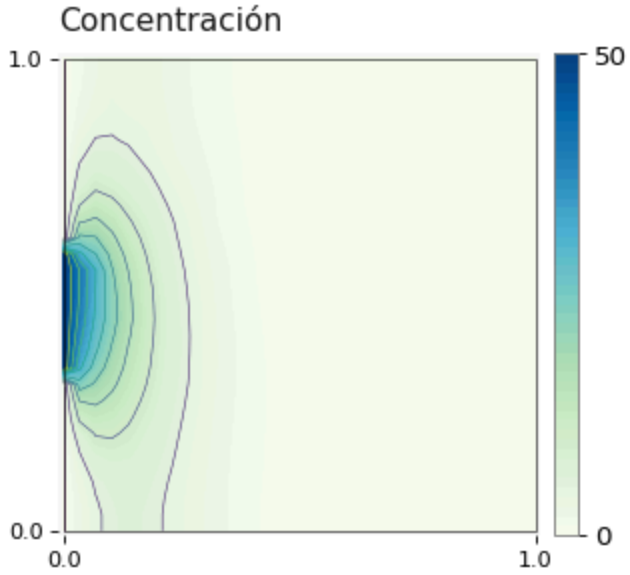
```
932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953
954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975
976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997
998 999 1000
```

```
        ax1 = dict(aspect='equal', title='Malla')
        ax2 = dict(aspect='equal', title='Carga hidráulica')
        ax3 = dict(aspect='equal', title='Velocidad')
        ax4 = dict(aspect='equal', title='Concentración')

        vis = mvis.Plotter(2,2,[ax1, ax2, ax3, ax4],
                           dict(figsize=(8,8)))

        vis.plot_mesh2D(1, xg, yg)
        vis.plot_frame(1, xg, yg)

        cax3 = vis.set_canvas(2,Lx,Ly)
        c_h = vis.contourf(2, xg, yg, h, levels=50, cmap='cool')
        vis.fig.colorbar(c_h, cax=cax3, ticks = [h.min(), h.max()], shrink=0.5, orientati
        vis.contour(2, xg, yg, h, levels=10, linewidths=0.5)
        vis.plot_frame(2, xg, yg)

        vis.set_canvas(3,Lx,Ly)
        vis.plot_frame(3, xg, yg)
        vis.streamplot(3, xg, yg, Vx, Vy)
        vis.quiver(3, xg, yg, Vx, Vy, scale=0.5)

        cax4 = vis.set_canvas(4,Lx,Ly)
        c_c = vis.contourf(4, xg, yg, c, levels=50, cmap='GnBu')
        vis.fig.colorbar(c_c, cax=cax4, ticks = [c.min(), c.max()], shrink=0.5, orientati
        vis.contour(4, xg, yg, c, levels=10, linewidths=0.5)
        vis.plot_frame(4, xg, yg)

        plt.savefig('contaminante.pdf')
        vis.show()
```


Malla
1.0
0.0
0.0
1.0


Carga hidráulica
1.0
0.0
0.0
1.0
50
0


Velocidad
1.0
0.0
0.0
1.0


Concentración
1.0
0.0
0.0
1.0
50
0