Equation de convection - diffusion 1D

Méthode des volumes-finis

Cas 1d : écoulement à vitesse constante

La variable scalaire Φ est transportée par convection et diffusion dans un écoulement à vitesse constante et connue (u) traversant un domaine mono-dimensionnel, voir la figure ci-dessous où figurent les conditions limites du problème.

La solution analytique du problème s'écrit $ \dfrac{\Phi -\Phi_0}{\Phi_L -\Phi_0} = \left({\exp{\left(\dfrac{\rho u x}{\Gamma}\right)} -1} \right) / \left({\exp{\left(\dfrac{\rho u L}{\Gamma}\right)} -1} \right)$

In [ ]:
```
from IPython.display import display, Image, Math
i=Image(filename='fig/diff_conv_exemple.png', width=600)
display(i)
```

Soit à résoudre le problème de transport 1D par convection-diffusion

$ \dfrac{d}{dx}(\rho u \Phi) = \dfrac{d}{dx} \left(\Gamma \frac{d\Phi}{dx} \right) \quad \textrm{pour} \quad 0<x<L, \textrm{avec} \quad \Phi(0)=\Phi_A=1, \quad \Phi(L)=\Phi_B=0 $

On utilisera un maillage décalé **régulier** à faces centrées, tout d'abord à 5 volumes de contrôle.

In [ ]:
```
i=Image(filename='fig/dessin.png', width=600)
display(i)
```

In [3]:
```
# python packages
import sys
sys.path.append('src')
get_ipython().run_line_magic('matplotlib', 'inline')

import numpy as np
import scipy.sparse as sp
from scipy.sparse.linalg import spsolve
import matplotlib.pylab as plt

np.set_printoptions(linewidth=130)
```

In [4]:
```
# source files
from grid_module import *           # mesh
from properties_module import *     # physical properties definition
from solver_module import *         # operators
from bc_module import *             # boundary conditions     Gamx = gami
```

1. Set the problem parameters: boundary conditions and mesh

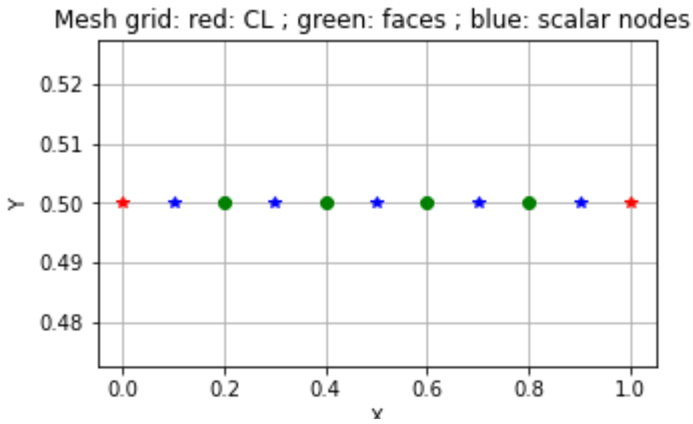
In [5]:
```
# physical parameters
Lx = 1.
xstart = 0 ; xend = Lx
Ly = 1.
ystart = 0 ; yend = 1.

Uval = 0.1          # 0.1 2.5 -0.1 -2.5
rhoval = 1.
gamma_x = 0.1

# boundary conditions
C0 = 1 # BC x = 0 at A
CL = 0  # BC x = L at B
U0 = Uval # BC x = 0
UL = Uval # BC x = L
```

In [22…
```
# mesh

# number of inner nodes  = number of unknowns
m = 5               # F-V number (5 10 50)
# 1D test case
n=1

# Numerical grid for the fluid
x = reg_grid (m,xstart, xend)     # x: velocity positions
y = reg_grid (n,ystart, yend)     # x: velocity positions

# Mesh and sizes of variables on inner nodes
dx,xp,dim_sca,dxp,xu,dim_U,dxu = coordinates(m,x)
dy,yp,dim_sca_y,dyp,yv,dim_V,dyv = coordinates(n,y)
# 1D test case
print('x : ', x, '\n y : ', y)
print('xp : ',xp, '\n yp : ', yp)
print('xu : ',xu, '\n yv : ', yv)
```

```
x :  [0.  0.2 0.4 0.6 0.8 1. ]
 y :  [0. 1.]
xp :  [0.1 0.3 0.5 0.7 0.9]
 yp :  [0.5]
xu :  [0.2 0.4 0.6 0.8]
 yv :  []
```

In [7]:
```
# figure

plt.figure()
XCL, YCL = np.meshgrid(x, yp)
XU, YU = np.meshgrid(xu, yp)
XP, YP = np.meshgrid(xp, yp)

fig = plt.figure(figsize=(5,3))
plt.plot(XCL, YCL, 'r*');
plt.plot(XU, YU, 'go');
plt.plot(XP, YP, 'b*');

plt.xlabel("X")
plt.ylabel('Y')
plt.title( "Mesh grid: red: CL ; green: faces ; blue: scalar nodes " )
plt.grid(True)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



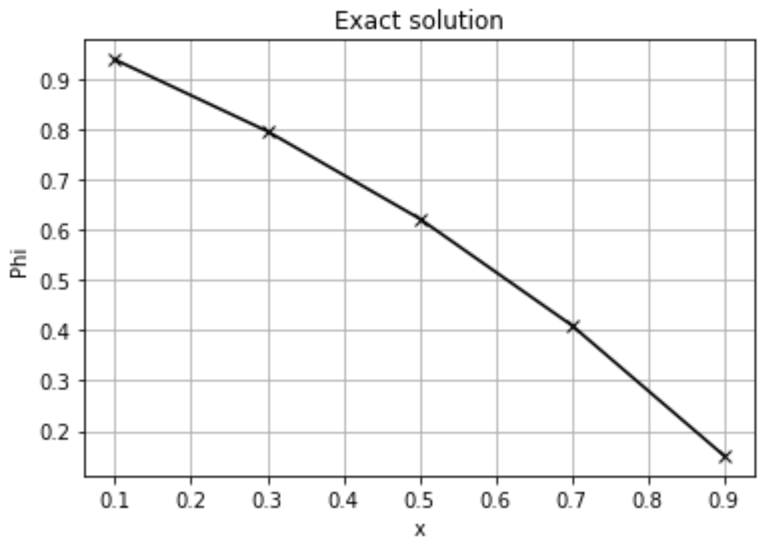
2. Plot and print the exact solution on the scalar mesh

In [8]:

```
def Cexact (x):
    """
    x       : position
    """
    f = C0+(CL-C0)*(np.exp(rhoval*Uval*x/gamma_x)-1)/(np.exp(rhoval*Uval*

    return f
```

In [9]:

```
sol_exact = Cexact(xp)

fig=plt.figure()
plt.plot(xp, sol_exact,'kx-')
plt.xlabel('x')
plt.ylabel('Phi')
plt.grid(True)
plt.savefig('Figures/Cexacte.png')
plt.title( "Exact solution" )

plt.show()

for j in range(0,np.size(sol_exact)):
        print('xp({0:3d}  ) =  {1:3.2f},   Cexact ({0:3d}  ) = {other:7.9
```




```
xp(  0  ) =  0.10,   Cexact (  0  ) = 0.938792975
xp(  1  ) =  0.30,   Cexact (  1  ) = 0.796390323
xp(  2  ) =  0.50,   Cexact (  2  ) = 0.622459331
```

```
xp(  3  ) =  0.70,   Cexact (  3  ) = 0.410019538
xp(  4  ) =  0.90,   Cexact (  4  ) = 0.150544988
```

3. Calculate the physical properties on the velocity mesh

In [10…

```
# physical properties estimated on the VF faces
Gamx_ew, rho_ew = prop_phys(dim_sca, dim_U, gamma_x, rhoval)
print(np.shape(Gamx_ew))
```

```
(4, 1)
```

4. Solve the diffusive problem

In [11…

```
# declarations

matA = np.eye( m )                       # the whole matrix including inner
BB = np.zeros( np.array ([m-2,m]) )      # diffusive part of matA
bcW = np.zeros( np.array ([1, m]) )      # BC row of matA
bcE = np.zeros( np.array ([1, m]) )      # BC row of matA

source = np.zeros(dim_sca)               # source vector
SbcW,  SbcE = 0., 0.                     # BC elements of Source
```

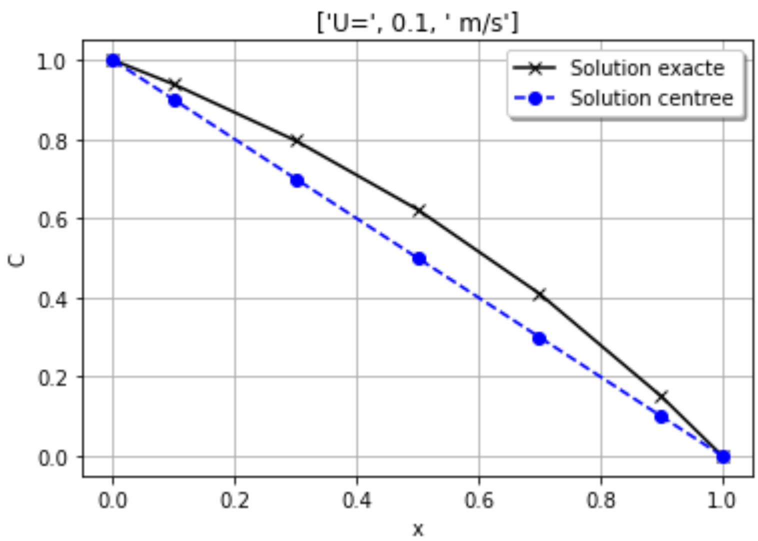
In [12…

```
### matrix
Div, Gra = gradiv (m,dxp) # divergence, gradient
mass = sp.diags([dy], [0], (m, m)).toarray() #dy - diagonal matrix

BB = mass[1:-1,1:-1] @ (Div@(Gamx_ew * Gra))
matA[1:-1,:]=BB

### boundary conditions
bcW[0,1] = matA[1,2] # a_E unchanged
bcE[0,-2] = matA[-2,-3]  # a_W unchanged
bcW[0,0] = matA[1,1] # a modifier par bc
bcE[0,-1] = matA[-2,-2] # a modifier par bc
bcW, bcE, SbcW, SbcE = bc_diff(bcW, bcE, SbcW, SbcE, m, x, xp, dxp,C0, Cl

# assembling
matA = np.concatenate((bcW[:], matA[1:-1,:], bcE[:]), axis=0)
source[0]= SbcW
source[-1]= SbcE
```

In [13…

```
### Resolution by a sparse solver: spsolve

# declaration of solution field
c_sol = np.ones(shape=m+2)
c_sol[0] = C0
c_sol[-1] = CL

# solver
CC = sp.csr_matrix(matA)
c_sol[1:-1] = spsolve(CC,source)
```

In [14…
```
# FIGURE #
xtot = np.ones(shape=m+2)
xtot[1:m+1] = xp
xtot[0] = xstart
xtot[-1] = xend
Cexact = C0+(CL-C0)*(np.exp(rhoval*Uval*xtot/gamma_x)-1)/(np.exp(rhoval*U

plt.figure()
plt.plot(xtot, Cexact,'kx-')
plt.plot(xtot, c_sol,'bo--')
plt.legend(('Solution exacte', 'Solution centree'), loc='best', shadow=Tr
plt.xlabel('x')
plt.ylabel('C')
plt.title(['U=', Uval, ' m/s'])
plt.grid(True)
plt.savefig('Figures/C_Diff_centred.png')
plt.show()

for j in range(0,np.size(Cexact)):
        print('xtot({0:3d}  ) =  {1:3.2f},   c_VF ({0:3d}  ) = {2:7.9f},
```




```
xtot(  0  ) =  0.00,   c_VF (  0  ) = 1.000000000,   Cexact (  0  ) = 1.0
00000000
xtot(  1  ) =  0.10,   c_VF (  1  ) = 0.900000000,   Cexact (  1  ) = 0.9
38792975
xtot(  2  ) =  0.30,   c_VF (  2  ) = 0.700000000,   Cexact (  2  ) = 0.7
96390323
xtot(  3  ) =  0.50,   c_VF (  3  ) = 0.500000000,   Cexact (  3  ) = 0.6
22459331
xtot(  4  ) =  0.70,   c_VF (  4  ) = 0.300000000,   Cexact (  4  ) = 0.4
10019538
xtot(  5  ) =  0.90,   c_VF (  5  ) = 0.100000000,   Cexact (  5  ) = 0.1
50544988
xtot(  6  ) =  1.00,   c_VF (  6  ) = 0.000000000,   Cexact (  6  ) = 0.0
00000000
```

5. Solve the convective-diffusive problem

In [15…
```
# declarations
# Ux : vector at the VF faces
Ux = Uval * np.ones (dim_U)
```

In [16…

```
### matrix
Int      = interp(m)  # centred interpolation
AA = mass[1:-1,1:-1] @ ( Div@(rho_ew*Ux*Int))
matA[1:-1,:]=matA[1:-1,:]-AA

### boundary conditions
bcW[0,1] = matA[1,2] # a_E unchanged
bcE[0,-2] = matA[-2,-3]  # a_W unchanged
bcW, bcE, SbcW, SbcE = bc_interp(bcW, bcE,SbcW, SbcE, U0, UL,C0,CL,rhoval

# assembling
matA = np.concatenate((bcW[:], matA[1:-1,:], bcE[:]), axis=0)
source[0]= SbcW
source[-1]= SbcE

print(matA)
```

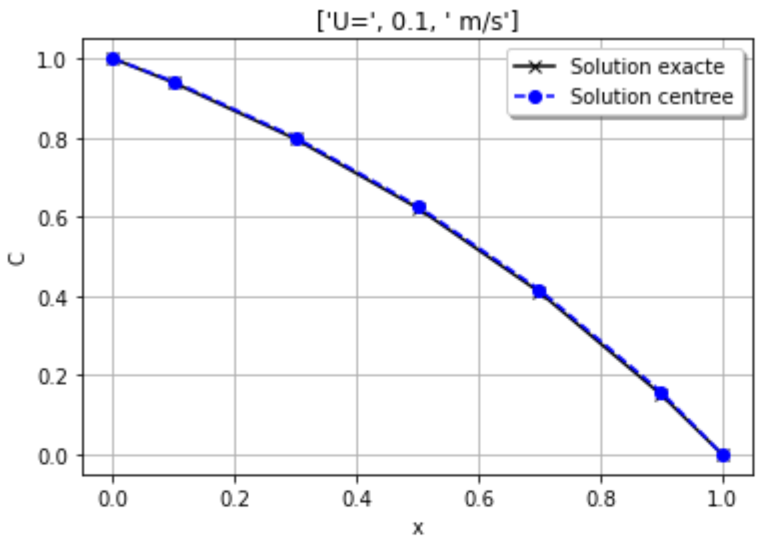
```
[[-1.55  0.45  0.    0.    0.  ]
 [ 0.55 -1.    0.45  0.    0.  ]
 [ 0.    0.55 -1.    0.45  0.  ]
 [ 0.    0.    0.55 -1.    0.45]
 [ 0.    0.    0.    0.55 -1.45]]
```

In [17…

```
### Resolution by a sparse solver: spsolve

# initialisation of solution field
c_sol = np.ones(shape=m+2)
c_sol[0] = C0
c_sol[-1] = CL

# solver
CC = sp.csr_matrix(matA)
c_sol[1:-1] = spsolve(CC,source)
```

In [18…

```
# FIGURE #
xtot = np.ones(shape=m+2)
xtot[1:m+1] = xp
xtot[0] = xstart
xtot[-1] = xend
Cexact = C0+(CL-C0)*(np.exp(rhoval*Uval*xtot/gamma_x)-1)/(np.exp(rhoval*U

plt.figure()
plt.plot(xtot, Cexact,'kx-')
plt.plot(xtot, c_sol,'bo--')
plt.legend(('Solution exacte', 'Solution centree'), loc='best', shadow=T
plt.xlabel('x')
plt.ylabel('C')
plt.title(['U=', Uval, ' m/s'])
plt.grid(True)
plt.savefig('Figures/C_CDiff_centred.png')
plt.show()

for j in range(0,np.size(Cexact)):
        print('xtot({0:3d}  ) =  {1:3.2f},   c_VF ({0:3d}  ) = {2:7.9f},
```




```
xtot(  0  ) =  0.00,   c_VF (  0  ) = 1.000000000,   Cexact (  0  ) = 1.0
00000000
xtot(  1  ) =  0.10,   c_VF (  1  ) = 0.942109959,   Cexact (  1  ) = 0.9
38792975
xtot(  2  ) =  0.30,   c_VF (  2  ) = 0.800600969,   Cexact (  2  ) = 0.7
96390323
xtot(  3  ) =  0.50,   c_VF (  3  ) = 0.627645536,   Cexact (  3  ) = 0.6
22459331
xtot(  4  ) =  0.70,   c_VF (  4  ) = 0.416255564,   Cexact (  4  ) = 0.4
10019538
xtot(  5  ) =  0.90,   c_VF (  5  ) = 0.157890041,   Cexact (  5  ) = 0.1
50544988
xtot(  6  ) =  1.00,   c_VF (  6  ) = 0.000000000,   Cexact (  6  ) = 0.0
00000000
```

6. The 2D diffusive problem

In [ ]:

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Feb  9 15:32:11 2021

@author: sergent
"""
import numpy as np
from scipy.special import erf
from math import pi

def reg_grid (n,xstart, xend):
    """
    s      : regular x distribution
    n      : cell number
    xstart : first point
    xend   : last point
    """
    dh=(xend-xstart)/(n)
    s =  np.array ([  j*dh for j in range(n+1) ])

    return s

def coordinates (m,x):
    """
    m       : cell number
    dx      : scalar cell size
    xp      : scalar node position
    dim_sca : number of scalar nodes
    dxp     : velocity cell size
    xu      : velocity node position
    dim_U   : number of velocity nodes
    dxu     : scalar cell size
    """
    dx= np.diff(x)

    # define scalar nodes inside the domain
    xp =  0.5 * (x[0:-1]+x[1:])
    dim_sca = np.array([ np.size(xp),1 ])
    dxp = np.diff(xp)

    # define velocity nodes inside the domain
    xu = x[1:-1]
    dim_U = np.array([ np.size(xu),1 ])
    dxu = np.diff(xu)

    return dx,xp,dim_sca,dxp,xu,dim_U,dxu

```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np

def prop_phys(dim_sca, dim_U, gamma_x, rhoval):
    """
    Gamx_ew : diffusion coeffient on the faces
    rho_ew  : density on the faces
    """
    # diffusion coeffient at the scalar nodes
    Gamx = gamma_x * np.ones (dim_sca)
    rho = rhoval * np.ones (dim_sca)

    # diffusion coeffient on the faces
    Gamx_ew = np.zeros (dim_U) # harmonic average
    rho_ew = np.zeros (dim_U)  # arithmetic average

    if gamma_x > 0:
        Gamx_ew = 2* (Gamx [0:-1,:]*Gamx [1:,:]) /   (Gamx [0:-1,:]+Gamx[1:,:])

    if rhoval > 0:
        rho_ew = 0.5* (rho [0:-1,:]+rho [1:,:])

    return Gamx_ew, rho_ew
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
"""
import numpy as np
import scipy.sparse as sp

#------------------------------------------------------------------
# Gradient and divergence operators
def gradiv (m, dxp):

    # declaration
    Div = np.zeros( np.array ([m-2,m-1]) )
    Gra = np.zeros( np.array ([m-1,m]) )

    # divergence
    Div = sp.diags([-1,1], [0,1], (m-2,m-1)).toarray()

    # gradient
    Gra = sp.diags([-1/dxp,1/dxp], [0,1], (m-1, m)).toarray()

    return Div, Gra
#------------------------------------------------------------------
# Interpolation centred scheme - face centred mesh
def interp(m):

   # declaration
   Int = np.zeros( np.array ([m-1,m]) )

   # interpolation
   Int = sp.diags([1./2,1./2], [0,1], (m-1, m-0)).toarray()

   return Int
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
# BOUNDARY CONDITIONS
"""

import numpy as np
import scipy.sparse as sp

#------------------------------------------------------------------------------
# diffusion for Phi at VF interface
def bc_diff(bcW, bcE, SbcW, SbcE, m, x, xp, dxp,C0, CL, gamma_x,Gamx_ew,mass):

    # WEST node
    bcW[0,0] = bcW[0,0] - mass[0,0] * gamma_x / (xp[0]-x[0]) + mass[0,0] * Gamx_ew[0] / dxp[0]
    SbcW = SbcW - mass[0,0] * gamma_x / (xp[0]-x[0])* C0

    # EST node
    bcE[0,-1] = bcE[0,-1] - mass[0,0] *gamma_x / (x[-1]-xp[-1]) + mass[0,0] *Gamx_ew[-1] / dxp[-1]
    SbcE = SbcE - mass[0,0] * gamma_x / (x[-1]-xp[-1]) * CL

    return bcW, bcE, SbcW, SbcE
#------------------------------------------------------------------------------
# centred interpolation for Phi at VF interface
def bc_interp(bcW, bcE,SbcW, SbcE, U0, UL,C0,CL,rhoval,rho_ew,Ux,mass):

    # WEST node
    bcW[0,0] = bcW[0,0] - mass[0,0] * (rho_ew*Ux)[0]/2.
    SbcW = SbcW - mass[0,0] * rhoval*U0 * C0

    # EST node
    bcE[0,-1] = bcE[0,-1]  + mass[0,0] * (rho_ew*Ux)[-1]/2.
    SbcE = SbcE + mass[0,0] * rhoval*UL *CL

    return bcW, bcE, SbcW, SbcE
#------------------------------------------------------------------------------
```