Name of the Program: Potential Flow Calculator
Author: Edith Villegas
Date: October 5th, 2018
Programming Language: Python
Format: Source code
File list: main.py (main program), defGeometrie.py (set of functions), geometrie.py (set of functions), integrafonct.py (set of functions), mathfonct.py (set of functions), systeme.py (set of functions), main_2.py (secondary program)

I. Functional Specification

The program allows the calculation of the potential flow of a fluid on four different geometries: a simple rectilinear canal, a canal with increasing width, a canal with a deviation, and a rectilinear canal with an obstacle. Calculations are done in 2 dimensions, and the system is considered invariant in the z direction. The viscosity of the fluid and the force exerted by the walls and obstacles on the fluid are not taken into account.

The program uses the following python libraries: numpy, matplotlib and scipy.

Input Variables

Variable	Description	Possible Values	Condition
v	initial speed [m/s]	Real number	
phiref	reference value for the potential ϕ at the exit [m^2/s]	Real number	
p0	initial pressure [kg/s^2]	Real number	
rho	fluid density [kg/m^2]	Positive real number	
h	step of the grid [m]	Positive real number	
ht	step in time [s]	Positive real number	
pointsx	size of the geometry grid (number of columns)	Integer (preferably between 5 and 100)	
pointsy	size of the geometry grid (number of lines)	Integer (preferably between 5 and 100)	
typegeometrie	geometry type	1: simple rectilinear canal, 2: canal with increasing width, 3: canal with a deviation, 4: rectilinear canal with an obstacle	
l0	Initial width of the canal	Integer (element number)	Geometry 2
lf	Final width of the canal	Integer (element number)	Geometry 2
coudex	Lenght of the deviator	Integer (element number)	Geometry 3
coudey	Height of the deviator	Integer (element number)	Geometry 3
x0	x position of the upper-left corner of the obstacle	Integer (element number)	Geometry 4
y0	y position of the upper-left corner of the obstacle	Integer (element number)	Geometry 4
xf	x position of the lower-right corner of the obstacle	Integer (element number)	Geometry 4
yf	y position of the lower-right	Integer (element number)	Geometry 4

	corner of the obstacle		

Output

Name	Format	Conditions
Potential Field ϕ	PDF (contour plot)	
Velocity Field	PDF (vector field)	
Current Lines	PDF (current lines)	
Pressure Field	PDF (filled contour plot)	
Speed profiles	PDF (plot)	Geometry 2, 3, 4
Pressure on the Obstacle	PDF (plot)	Geometry 3, 4
Force on the Obstacle	Numerical Value	Geometry 3, 4
Calculated Initial and Final Velocities	Numerical Values (for each entry and exit position in the grid)	

It is possible to do a comparison between two different integrations methods for the calculation of the lines of current: Euler and Runge-Kutta 4. The result will be two plots with both calculations, for a step in time of 0.1 s and 0.5 s. The secondary program main_2.py does this same comparison for two different obstacle shapes.

The geometrical parameters (like position of the obstacle), are specified in number of steps in the grid (pixels). Care has to be taken not to do enter parameters that wouldn’t make sense, like a deviator bigger than the canal itself, or an obstacle with its left corner to the right of the right corner.

II. Internal Functioning

defGeometrie

Set of functions to define the geometry matrix. The geometry of the system is represented as a matrix with only 0,1,2 and 3. 0 represents an obstacle or a wall, 2 represents a cell with fluid, 1 represents an entry cell and 3 an exit cell. Uses numpy.

canalh	canalh(pointsx, pointsy, border= True) Builds a matrix that represents a simple rectilinear canal **input:** pointsx: number of columns of the matrix (int) pointsy: number of lines of the matrix (int) border: optional boolean parameter, when “True” fills the first and last line with 0s (walls) **output** numpy array of size (pointsx,pointsy)
canalCoude	canalCoude(pointsx, pointsy, coudex, coudey) Builds a matrix that represents a canal with a deviator **input:** pointsx: number of columns of the matrix (int) pointsy: number of lines of the matrix (int) coudex: length of the deviator (in “pixels”), (int) coudey: height of the deviator (in “pixels”), (int)

	output numpy array of size (pointsx,pointsy)
canalObstacle	canalObstacle(pointsx, pointsy, x0, y0, xf, yf) Builds a matrix that represents a simple canal with an obstacle **input:** pointsx: number of columns of the matrix (int) pointsy: number of lines of the matrix (int) x0: position of the upper-left corner of the obstacle (column number in the matrix), (int) y0: position of the upper-left corner of the obstacle (line number in the matrix), (int) xf: position of the lower-right corner of the obstacle (column number in the matrix), (int) yf: position of the lower-right corner of the obstacle (line number in the matrix), (int) **output** numpy array of size (pointsx,pointsy)
canalElargi	canalElargi(pointsx, pointsy, l0, lf) Builds a matrix that represents a canal of varying width **input:** pointsx: number of columns of the matrix (int) pointsy: number of lines of the matrix (int) l0: initial width of the canal (in “pixels”), (int) lf: final width of the canal (in “pixels”), (int) **output** numpy array of size (pointsx,pointsy)
canalcircle	canalcircle(pointsx, pointsy, x0, y0, r) Builds a matrix that represents a canal with a circular obstacle **input:** pointsx: number of columns of the matrix (int) pointsy: number of lines of the matrix (int) x0: center of the circle (column number), (int) y0: center of the circle (line number), (int) r: radius of the circle (positive real number) **output** numpy array of size (pointsx,pointsy)
canaltriang	canaltriang(pointsx, pointsy, x0, y0, a) Builds a matrix that represents a canal with a triangular obstacle with soft corners **input** pointsx: number of columns of the matrix (int) pointsy: number of lines of the matrix (int)

	x0: center of the triangle (column number), (int) y0: center of the triangle (line number), (int) a: geometrical parameter that determines the size of the triangle (positive real number) **output** numpy array of size (pointsx,pointsy)

geometrie
Set of functions related to geometrical manipulations necessary for the program. Uses numpy.

numeroter	numeroter(matriceGeometrie) Returns a matrix with the fluid cells numbered (starting from one), while ignoring walls and obstacles. **input** matriceGeometrie: numpy array representing the geometry of the system **output** numpy array of the same size as the geometry matrix
numCases	numCases(matriceGeometrie) Returns the number of cells that contain fluid (not including the number of cells that represent walls and obstacles) **input** matriceGeometrie: numpy array representing the geometry of the system **output** numerical value (int)
tableaucoord	tableaucoord(matriceGeometrie, matriceNumerotee, nomCases) Returns a list with the coordinates of all the cells that contain fluid, ordered according to their corresponding number (number assigned by matriceNumerotee) **input** matriceGeometrie: numpy array representing the geometry of the system matriceNumerotee: numpy array representing the numbered matrix obtained from the geometry matrix by applying numeroter() nomCases: number of cells containing fluid (int) **output** list that contains pairs of coordinates [i,j] starting from one, the first element of the list is the number 0
voisinage	voisinage(matrice, i,j) Given a matrix and the coordinates of an element in the matrix, it returns a dictionary containing the numerical values of the neighboring elements, and counts the number of neighboring elements. **input** matrice: numpy array of 2 dimensions i: line number of the element to be considered

	j: column number of the element to be considered **output** returns the tuple voisins, numvoisins voisins: dictionary that contains the numerical values of the neighboring elements, those can be accessed with the following keys: “up”, “down”, “left” and “right” for each corresponding neighboring element numvoisins: numerical value of the number of neighbors
estBordh	estBordh(matrice, i, j, shape=None) Tells if the element (i,j) of a matrix is in a border (along the horizontal axis) **input** matrice: matrix to be considered (numpy array) i: line number of the element j: column number of the elements shape: optional parameter indicating the shape of the matrix (tuple: lines, columns) **output** returns a string that can be either “left” to indicate that the element is in the left border, “right” to indicate that it’s in the right border and “inside” to indicate that the element is not in a border.
estBordv	estBordv(matrice, i, j, shape=None) Tells if one element of a matrix is in a border (along the vertical axis) **input** matrice: matrix to be considered (numpy array) i: line number of the element j: column number of the elements shape: optional parameter indicating the shape of the matrix (tuple: lines, columns) **output** returns a string that can be either “up” to indicate that the element is in the upper border, “down” to indicate that it’s in the lower border and “inside” to indicate that the element is not in a border.
estBord	estBord(matrice, i, j) Tells if one element of a matrix is in a border (along both axis) **input** matrice: matrix to be considered (numpy array) i: line number of the element j: column number of the elements **output** returns a list of two strings [positionh, positionv], where positionh is the result of the function estBordh and positionv is the result of the function estBordv
estBordv_nan	estBordv_nan(matrice, i,j, shape=None)

	Tells if one element of a matrix is in a border, like estBordv, but it also considers a neighboring NaN element as a border **input** matrice: matrix to be considered (numpy array) i: line number of the element j: column number of the elements shape: optional parameter indicating the shape of the matrix (tuple: lines, columns) **output** returns a string that can be either “up” to indicate that the element is in the upper border, “down” to indicate that it’s in the lower border and “inside” to indicate that the element is not in a border.
estBordh_nan	estBordh_nan(matrice, i,j, shape=None) Tells if the element (i,j) of a matrix is in a border, like estBordh, but it also considers neighboring NaN elements as a border **input** matrice: matrix to be considered (numpy array) i: line number of the element j: column number of the elements shape: optional parameter indicating the shape of the matrix (tuple: lines, columns) **output** returns a string that can be either “left” to indicate that the element is in the left border, “right” to indicate that it’s in the right border and “inside” to indicate that the element is not in a border.

mathfonct
Set of functions to calculate the gradient using the finite differences method. Uses numpy and geometrie.

gradiently	gradiently(matrice, i,j, h=1, t='progressive') Returns the (i,j) element of the matrix representing the y component of the gradient of *matrice*, using the finite differences method. **input** matrice: matrix that we want to calculate the gradient of i: line number of the element that we want j: column number of the element that we want h: optional parameter that indicates the step of the grid t: optional parameter that indicates how the gradient will be calculated, possible values are “progressive”, “retrograde” and “centree” **output** numerical value of the (i,j) element of the y component of the gradient
gradientlx	gradientlx(matrice, i,j, h=1, t='progressive') Returns the (i,j) element of the matrix representing the x component of the gradient of *matrice*, using the finite differences method.

	input matrice: matrix that we want to calculate the gradient of i: line number of the element that we want j: column number of the element that we want h: optional parameter that indicates the step of the grid t: optional parameter that indicates how the gradient will be calculated, possible values are "progressive", "retrograde" and "centree" **output** numerical value of the (i,j) element of the x component of the gradient
gradienty	gradienty(matrice, h=1) Returns the y component of the gradient of *matrice*, using the finite differences method. **input** matrice: matrix that we want to calculate the gradient of h: optional parameter that indicates the step of the grid **output** numpy array of the same size that the original matrix, containing y component of the gradient
gradientx	gradientx(matrice, h=1) Returns the x component of the gradient of *matrice*, using the finite differences method. **input** matrice: matrix that we want to calculate the gradient of h: optional parameter that indicates the step of the grid **output** numpy array of the same size that the original matrix, containing x component of the gradient
gradienty_c	gradienty_c(matrice, h=1) Returns the y component of the gradient of *matrice*, using the finite differences method. This function works for matrices that include NaN values, and considers them as a boundary. **input** matrice: matrix that we want to calculate the gradient of h: optional parameter that indicates the step of the grid **output** numpy array of the same size that the original matrix, containing y component of the gradient
gradientx_c	gradientx_c(matrice, h=1) Returns the x component of the gradient of *matrice*, using the finite differences method. This function works for matrices that include NaN values, and considers them as a boundary. **input**

	matrice: matrix that we want to calculate the gradient of h: optional parameter that indicates the step of the grid **output** numpy array of the same size that the original matrix, containing x component of the gradient

integrafonct
Set of functions to integrate the velocities and find the lines of current. Uses numpy, scipy and geometrie.

nantozero	nantozero(matrice) Changes all the NaN values of a matrix to zero. **input** matrice: numpy array (matrix) **output** numpy array with no NaN values
interpolation	interpolation(vx,vy, kind='linear') Given a discrete vector field (components vx and vy) in the form of two matrices, it returns two functions that give the value of the function at any point in the domain. **input** vx: x component of the vector field vy: y component of the vector field kind: optional parameter that indicates the type of interpolation, can be “linear” or “cubic” **output** tuple of functions (fvx,fvy) that interpolate the original matrices they take as arguments the position x,y
trajectoirelim	trajectoirelim(x0, y0, fvx, fvy, lim, typelim, tcoord, maxit=2000, ht=1) Calculates the trajectory of a particle given its initial position (x0, y0) and its velocity vector field using Euler’s Method. **input** x0: initial position in x (column number) y0: initial position in y (line number) fvx: function to calculate the velocity in all points in the domain (x component) fvy: function to calculate the velocity in all points in the domain (y component) lim: maximum or minimum value that we want to calculate (the function stops when that value is reached) typelim: parameter that tells the function if the limit is a maximum or a minimun, and if its in the y or x axis. Can be “xmax”, “xmin”, “ymax”, “ymin”. tcoord: list of coordinates of all fluid cells in the geometry matrix, this helps the function stop the calculation when the particle hits a wall or obstacles maxit: maximum number of iterations the function will perform, optional ht: step in time for the calculation, in seconds, optional

	output tuple of lists x,y containing the x and y coordinates for each step
trajectoireRK4	trajectoireRK4(x0, y0, fvx, fvy, lim, typelim, tcoord, maxit=2000, ht=1) Calculates the trajectory of a particle given its initial position (x0, y0) and its velocity vector field using Runge-Kutta 4 method. **input** x0: initial position in x (column number) y0: initial position in y (line number) fvx: function to calculate the velocity in all points in the domain (x component) fvy: function to calculate the velocity in all points in the domain (y component) lim: maximum or minimum value that we want to calculate (the function stops when that value is reached) typelim: parameter that tells the function if the limit is a maximum or a minimun, and if its in the y or x axis. Can be “xmax”, “xmin”, “ymax”, “ymin”. tcoord: list of coordinates of all fluid cells in the geometry matrix, this helps the function stop the calculation when the particle hits a wall or obstacles maxit: maximum number of iterations the function will perform, optional ht: step in time for the calculation, in seconds, optional **output** tuple of lists x,y containing the x and y coordinates for each step
coord	coord(matriceGeometrie, tcoord, nomCases) Returns the coordinates of the entry cells and of the exit cells. **input** matriceGeometrie: the geometry matrix tcoord: list of coordinates of all fluid cells nomCases: number of fluid cells, not counting walls and obstacles **output** tuple of lists coordentree, coordsortie that contain the coordinates of the entry and exit cells, respectively
orientation	orientation(matriceGeometrie, coords) Returns the orientation and position of the entry and exit of the geometry matrix **input** matriceGeometrie: the geometry matrix coords: tuple of coordinates of the entry and exit cells, obtained with the coord function **output** tuple of strings orient_entree, orient_sortie that indicate the orientation of the entry and exit in the geometry matrix. Possible values are “vertical, left”, “vertical, right”, “horizontal, up” and “horizontal, down”

systeme

Set of functions to find the velocity potential field, the velocities and pressures from a given geometry matrix of the system and some given initial parameters. Uses numpy, mathfonct and geometrie.

The velocity potential ϕ can be calculated from the following equation:

$$\Delta \phi(x, y) = 0$$

To solve the equation numerically, the finite differences method is used. Border conditions are Neuman conditions in the entry cells and Dirichlet conditions in the exit cells. The equation is discretised and a matrix equation of the form Ax = B is found, where unknown x corresponds to the velocity potential field.

After calculating the velocity potential field, the velocity itself is calculated according to the following equation:

$$\vec{v} = -\nabla \phi$$

and the pressure:

$$P + 1/2 \rho v^2 = cte$$

sytemeB	systemeB(matriceGeometrie, nomCases, tcoord, h=1, v=1, phiref=1) Returns the matrix B for the matrix equation Ax= B, for a given geometry and initial conditions. **input** matriceGeometrie: Geometry matrix of the system nomCases: number of fluid cells tcoord: ordered list with the coordinates of all the numbered fluid cells h: optional parameter that indicates the grid step v: optional parameter that indicates the velocity of the fluid at the entry phiref: optional parameter that indicates the reference value for the potential ϕ at the exit **output** numpy array that represents matrix B in the linear matrix equation Ax = B, for the system
eqSortie	eqSortie(matriceGeometrie, nomCases, matriceNumerotee, i,j) Returns a line from the matrix A in the matrix equation Ax = B, corresponding to an equation for the Dirichlet condition. **input** matriceGeometrie: geometry matrix of the system nomCases: number of fluid cells matriceNumerotee: matrix with all fluid cells numbered i: coordinates of the exit cell that we are examining (line number) j: coordinates of the exit cell that we are examining (column number) **output** numpy array (1 dimensional)
eqEntree	eqEntree(matriceGeometrie, nomCases, matriceNumerotee, i,j) Returns a line from the matrix A in the matrix equation Ax = B, corresponding

	to an equation for the Neumann condition. **input** matriceGeometrie: geometry matrix of the system nomCases: number of fluid cells matriceNumerotee: matrix with all fluid cells numbered i: coordinates of the entry cell that we are examining (line number) j: coordinates of the entry cell that we are examining (column number) **output** numpy array (1 dimensional)
eqStandard	eqStandard(matriceGeometrie, nomCases, matriceNumerotee, i,j) Returns a line from the matrix A in the matrix equation Ax = B, corresponding to an equation that is found following the finite differences method for the element $\phi_{i,j}$ of the matrix. That is, the discretisation of $\frac{\partial \phi^2}{\partial x^2}+\frac{\partial \phi^2}{\partial y^2}=0$ for $\phi_{i,j}$. **input** matriceGeometrie: geometry matrix of the system nomCases: number of fluid cells matriceNumerotee: matrix with all fluid cells numbered i: coordinates of the exit cell that we are examining (line number) j: coordinates of the exit cell that we are examining (column number) **output** numpy array (1 dimensional)
systemeA	systemeA(matriceGeometrie, nomCases, matriceNumerotee, tcoord) Puts together the functions eqEntree(), eqSortie() and eqStandard to obtain the complete matrix A in the equation Ax=B. **input** matriceGeometrie: Geometry matrix of the system nomCases: number of fluid cells tcoord: ordered list with the coordinates of all the numbered fluid cells **output** numpy array that represents matrix A
PHI	PHI(matriceGeometrie, nomCases, matriceNumerotee, tcoord, h=1, v=1, phiref=1) Calculates the velocity potential matrix given the geometry of the system and the initial conditions of the problem, by solving the matrix equation Ax=B. **input** matriceGeometrie: Geometry matrix of the system nomCases: number of fluid cells tcoord: ordered list with the coordinates of all the numbered fluid cells h: optional parameter that indicates the grid step v: optional parameter that indicates the velocity of the fluid at the entry phiref: optional parameter that indicates the reference value for the potential ϕ at the exit **output**

	numpy array that represents matrix containing the velocity potential ϕ
c_nan	c_nan(matrice, matriceGeometrie) Takes one matrix and the geometry matrix of the system. Returns the initial matrix with NaN values where there are obstacles and walls. **input** matrice: a matrix matriceGeometrie: Geometry matrix of the system **output** numpy array (2D)
vitesse	vitesse(phi, h=1) Calculates the velocity from the velocity potential by applying the gradient, as in the equation: $$\vec{v} = -\nabla\phi$$ **input** matrice: a matrix matriceGeometrie: Geometry matrix of the system **output** tuple of 2 dimensional numpy arrays: VX, VY
pression	pression(VX, VY, matriceGeometrie, tcoord=None, p0=1, rho=1, v=1) Returns a matrix with the values of the pressure field, from the velocities. **input** VX: 2D numpy array containing the x component of the velocity VY: 2D numpy array containing the y component of the velocity matriceGeometrie: Geometry matrix of the system tcoord: ordered list with the coordinates of all the numbered fluid cells p0: pressure of the fluid at the entry cells rho: density of the fluid [kg/m^2] v: initial velocity of the fluid **output** numpy array (2D)
sol	sol(matriceGeometrie, h=1, v=1, phiref=1, p0=1, rho=1) For a given geometry of the system and given initial conditions, it returns the matrices for the potential field, the velocity vector field and the pressure field. **input** matriceGeometrie: geometry matrix of the system h: grid step v: initial velocity of the fluid phiref: parameter that indicates the reference value for the potential ϕ at the exit p0: pressure of the fluid at the entry cells rho: density of the fluid [kg/m^2] **output** tuple of 2D numpy arrays: phi, vx, vy, P

	phi is the velocity potential field, vx and vy are the components of the velocity vector field and P is the pressure vector field
vEntree	vEntree(VX,VY, matriceGeometrie, shape=None) Gives a list of the numerical values of the entry velocities. **input** VX: 2D numpy array representing the x component of the velocity VY: 2D numpy array representing the y component of the velocity matriceGeometrie: geometry matrix of the system shape: optional parameter representing the shape of the geometry matrix. If not given, it is calculated. **output** list of lists containing the coordinates for the entry cells and the corresponding velocities in the following format: (i,j), vx, vy
vSortie	vSortie(VX,VY, matriceGeometrie, shape=None) Gives a list of the numerical values of the exit velocities. **input** VX: 2D numpy array representing the x component of the velocity VY: 2D numpy array representing the y component of the velocity matriceGeometrie: geometry matrix of the system shape: optional parameter representing the shape of the geometry matrix. If not given, it is calculated. **output** list of lists containing the coordinates for the exit cells and the corresponding velocities in the following format: (i,j), vx, vy
calculForce	calculForce(P, numline, lmin, lmax, h=1, horizontal=True) Integrates the pressure over a line or column to calculate the force applied over that line or column (for example, the force applied to the obstacle). **input** P: 2D numpy array representing the pressure field numline: number of line or column lmin: inferior limit for the integration lmax: superior limit for the integration h: grid step horizontal: boolean parameter that indicates if we are integrating over a line (True) or over a column (False) **output** numerical value of the force

affichages

Set of functions to plot the results. Uses matplotlib, numpy, integrafonct and geometrie.

affGeometrie	affGeometrie(matriceGeometrie) Plots the geometry matrix **input** matriceGeometrie: geometry matrix (numpy array)

	output the representation of the geometry of the system is plotted on the screen
affPotentiel	affPotentiel(phi, matriceGeometrie, nom='') Generates a contour plot of the velocity potential ϕ, and saves it as a pdf. **input** phi: potential field matrix (numpy array) matriceGeometrie: geometry matrix nom: prefix to be added to the name of the resulting plot **output** contour plot in a pdf file
affVitesse	affVitesse(VX,VY, matriceGeometrie, numval = False, vin = None, vout = None, nom='', textrotation =0) Generates a vector field plot of the velocities and saves it as a pdf. It can also include the numerical values of the exit and entry velocities in the plot. **input** VX: matrix (numpy array) with the x component of the velocity VY: matrix (numpy array) with the y component of the velocity matriceGeometrie: geometry matrix numval: optional parameter, if "True", plots the initial and final velocities, the values have to be provided vin: initial velocities, list of values in format: [(i,j), vx, vy] vout: final velocities, list of values in format: [(i,j), vx, vy] nom: prefix to be added to the name of the resulting plot textrotation: optional parameter that changes the orientation of the labels for the exit velocity (90 makes it vertical) **output** vector field plot in a pdf file
affCourant	affCourant(VX, VY, matriceGeometrie, nom='') Generates a plot with the contour lines, using the function streamplot in matplotlib.pyplot. Saves the result in a pdf file. It only works for square matrices. **input** VX: matrix (numpy array) with the x component of the velocity VY: matrix (numpy array) with the y component of the velocity matriceGeometrie: geometry matrix nom: prefix to be added to the name of the resulting plot **output** plot of current lines in a pdf file
affPresion	affPression(P, matriceGeometrie, nom='') Generates a filled contour plot with the pressure field. Saves the result as a pdf. **input** P: pressure field matrix (numpy array) matriceGeometrie: geometry matrix

	nom: prefix to be added to the name of the resulting plot **output** filled contour plot in a pdf file
affCourant2	affCourant2(vx, vy, matriceGeometrie, ht=1, maxit=2000, nom='', methode='RK4') Plots the lines of current of the system, by integrating the velocities and finding the trajectories of a set of initial particles (each one beginning in an entry cell). It works for rectangular matrices. The method of integration can be chosen, available methods are Euler's method and Runge-Kutta 4. **input** VX: matrix (numpy array) with the x component of the velocity VY: matrix (numpy array) with the y component of the velocity matriceGeometrie: geometry matrix ht: step in time for the numerical integration, in seconds (positive real value) maxit: maximum number of iterations for the integration of each line nom: prefix to be added to the name of the resulting plot methode: integration method to be used, possible values are "RK4" (Runge-Kutta 4), "Euler" and "comparaison" (comparison of the two methods on the same plot). **output** plot of current lines in a pdf file
ProfilVitesse	ProfilVitesse(vx, vy, numline, horizontal=True, nom='') Generates a plot of the speed profile in a section of the geometry of the system. The result is saved as a pdf. **input** vx: matrix (numpy array) with the x component of the velocity vy: matrix (numpy array) with the y component of the velocity numline: number of line or column of the geometry matrix that will be shown in the speed profile horizontal: optional boolean parameter. If True, numline is a line in the matrix, if False, numline is a column number of the matrix. nom: prefix to be added to the name of the resulting plot **output** plot of the speed profiles as a pdf file
TracePression	TracePression(P, numline, lmin, lmax, horizontal=True, nom='') Generates a plot of the pressure in a section of the geometry of the system. The result is saved as a pdf. **input** P: pressure field matrix (numpy array) numline: number of line or column of the geometry matrix that will be shown in the speed profile lmin: minimum value for the position axis lmax: maximum value for the position axis horizontal: optional boolean parameter. If True, numline is a line in the matrix, and lmin, lmax are column numbers. If False, numline is a column number of the matrix, and lmin, lmax are line numbers.

	nom: prefix to be added to the name of the resulting plot **output** plot of the speed profiles as a pdf file

III. Program Reliability
All the functions included in the program were tested to ensure that they worked properly and behaved in the way they were expected to behave. The set of functions included in geometrie, defGeometrie, integrafonct, mathfonct were manually checked in a number of see if their output was correct. The results of the functions in affichages were visually checked. The results of the functions to numerically solve the problem (functions in systeme) were compared to a manually calculated result.

IV. Example

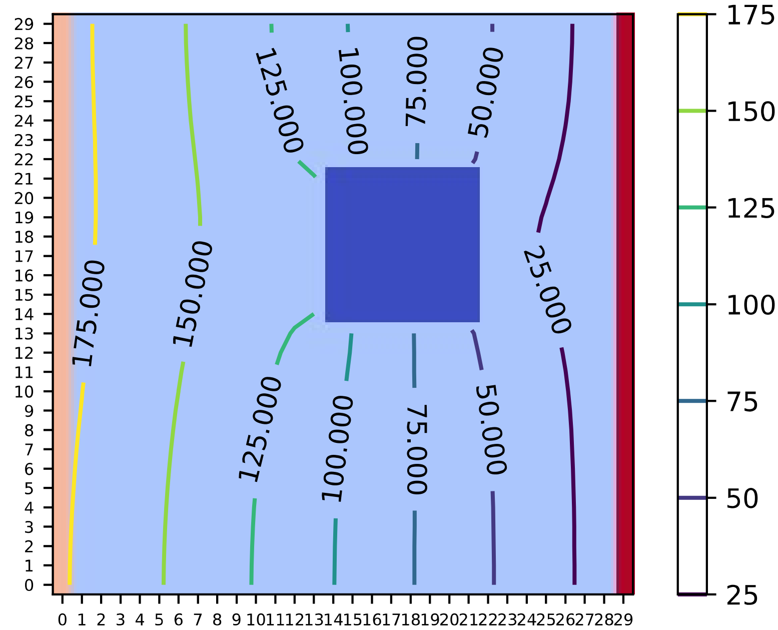
input
Initial velocity [m/s]: 5
Reference value for the potential: 10
Initial pressure value: 20
Fluid density: 1
Grid step: 1
Step in time: 0.1
Matrix size (x): 30
Matrix size (y): 30
Geometry: 4
Upper-left angle of the obstacle (x): 14
Upper-left angle of the obstacle (y): 14
Lower-right angle of the obstacle(x): 21
Lower-right angle of the obstacle (y): 21
Show influence of the integration method: yes

output
Vitesses Initiales: (x,y) vx, vy
[[(0, 0), 5.0, -0.07], [(1, 0), 5.0, -0.1], [(2, 0), 5.0, -0.17], [(3, 0), 5.0, -0.23], [(4, 0), 5.0, -0.29], [(5, 0), 5.0, -0.35], [(6, 0), 5.0, -0.4], [(7, 0), 5.0, -0.45], [(8, 0), 5.0, -0.49], [(9, 0), 5.0, -0.52], [(10, 0), 5.0, -0.53], [(11, 0), 5.0, -0.53], [(12, 0), 5.0, -0.52], [(13, 0), 5.0, -0.49], [(14, 0), 5.0, -0.44], [(15, 0), 5.0, -0.38], [(16, 0), 5.0, -0.31], [(17, 0), 5.0, -0.23], [(18, 0), 5.0, -0.15], [(19, 0), 5.0, -0.07], [(20, 0), 5.0, -0.0], [(21, 0), 5.0, 0.06], [(22, 0), 5.0, 0.1], [(23, 0), 5.0, 0.13], [(24, 0), 5.0, 0.14], [(25, 0), 5.0, 0.14], [(26, 0), 5.0, 0.12], [(27, 0), 5.0, 0.1], [(28, 0), 5.0, 0.06], [(29, 0), 5.0, 0.04]]

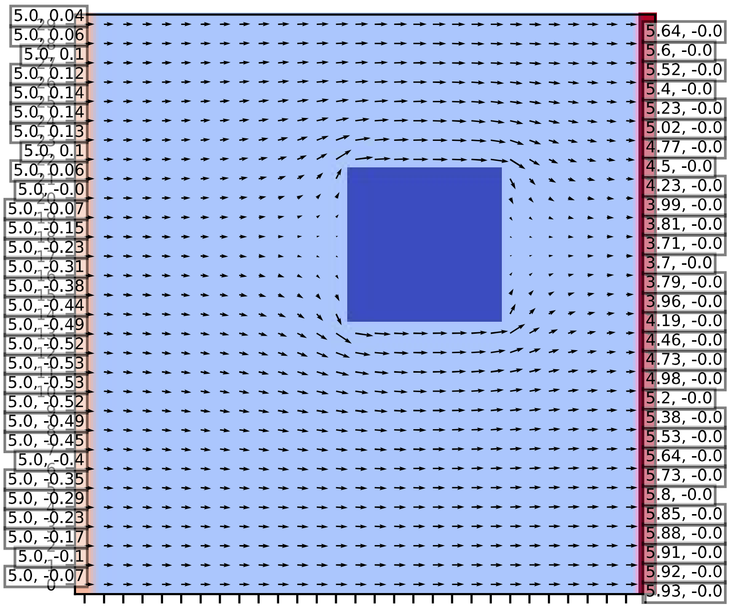
Vitesses Finales: (x,y), vx, vy
[[(0, 29), 5.93, -0.0], [(1, 29), 5.92, -0.0], [(2, 29), 5.91, -0.0], [(3, 29), 5.88, -0.0], [(4, 29), 5.85, -0.0], [(5, 29), 5.8, -0.0], [(6, 29), 5.73, -0.0], [(7, 29), 5.64, -0.0], [(8, 29), 5.53, -0.0], [(9, 29), 5.38, -0.0], [(10, 29), 5.2, -0.0], [(11, 29), 4.98, -0.0], [(12, 29), 4.73, -0.0], [(13, 29), 4.46, -0.0], [(14, 29), 4.19, -0.0], [(15, 29), 3.96, -0.0], [(16, 29), 3.79, -0.0], [(17, 29), 3.7, -0.0], [(18, 29), 3.71, -0.0], [(19, 29), 3.81, -0.0], [(20, 29), 3.99, -0.0], [(21, 29), 4.23, -0.0], [(22, 29), 4.5, -0.0], [(23, 29), 4.77, -0.0], [(24, 29), 5.02, -0.0], [(25, 29), 5.23, -0.0], [(26, 29), 5.4, -0.0], [(27, 29), 5.52, -0.0], [(28, 29), 5.6, -0.0], [(29, 29), 5.64, -0.0]]

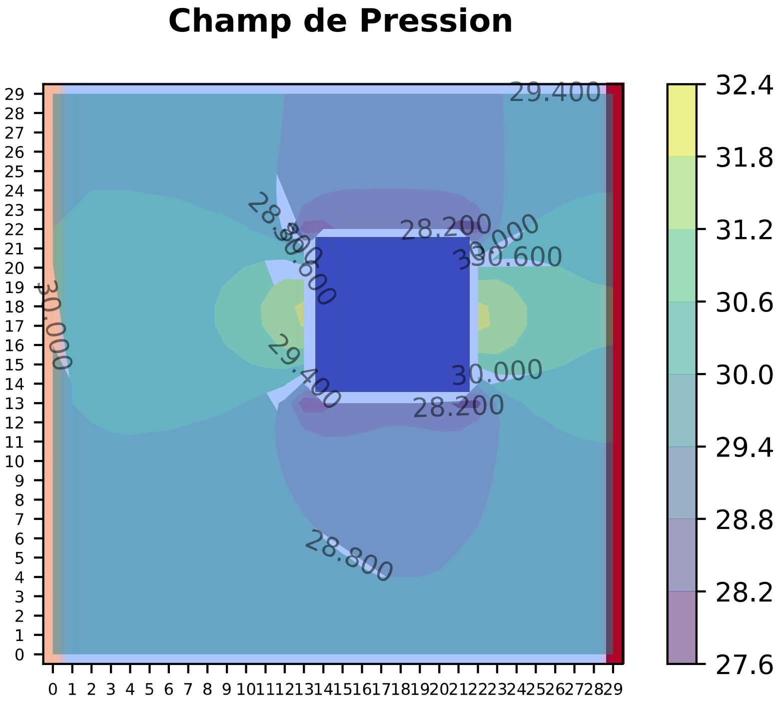
La force sur l'obstacle est de 217.26510802821863

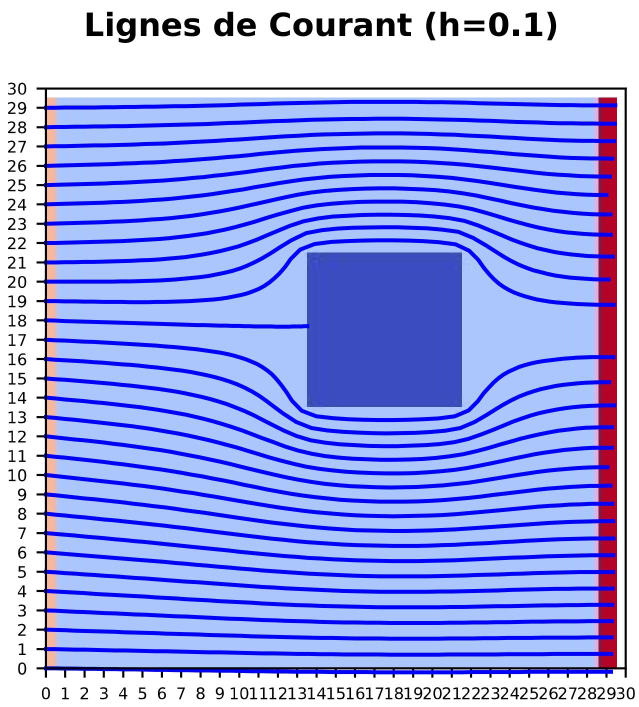
Potentiel de Vitesses ϕ

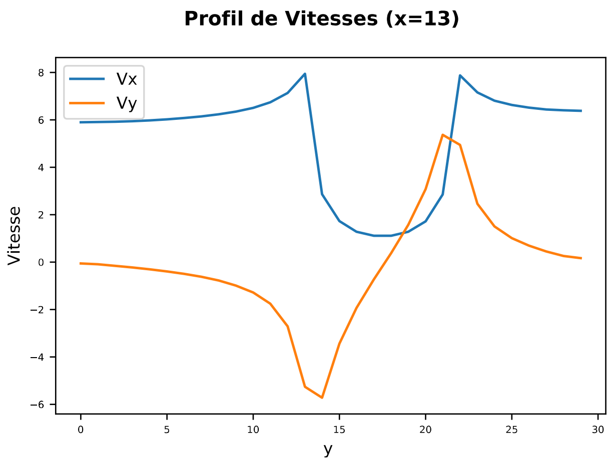


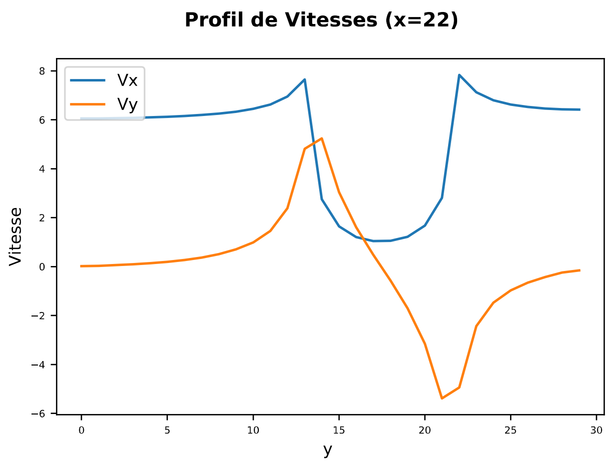
Champ de Vitesses

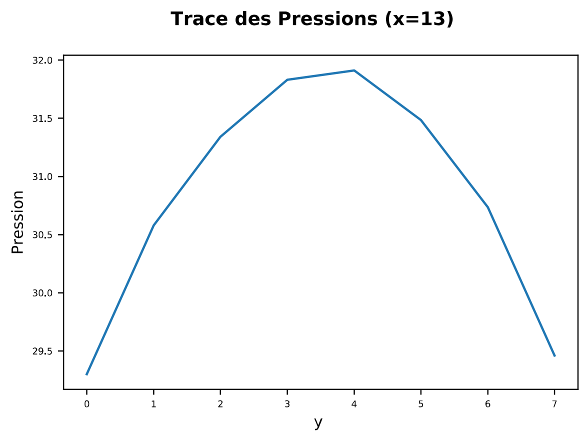


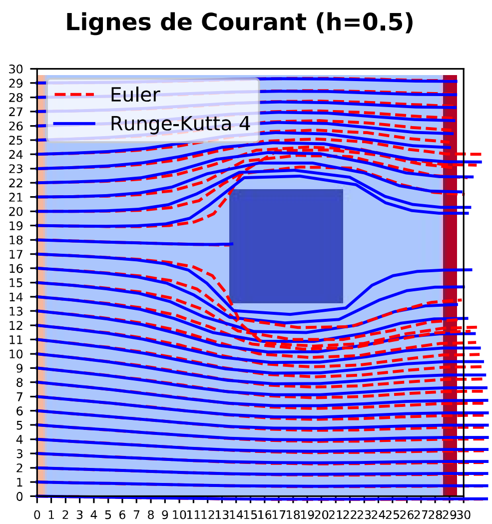

Champ de Pression
29.400
28.200
30.000
30.600
28.800
29.400
30.000
28.200
30.000
28.800
32.4
31.8
31.2
30.6
30.0
29.4
28.8
28.2
27.6

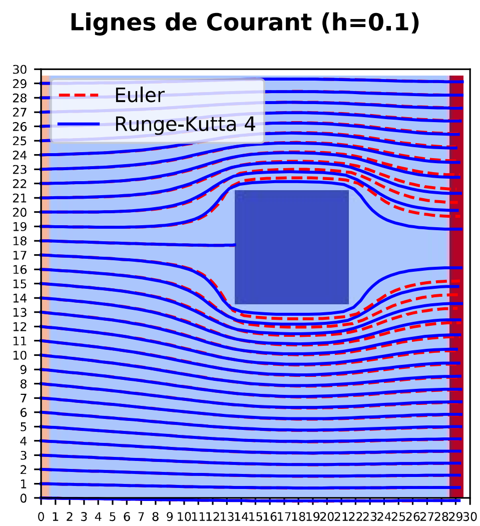

Lignes de Courant (h=0.1)


Profil de Vitesses (x=13)
Vx
Vy
Vitesse
y


Profil de Vitesses (x=22)
Vx
Vy
Vitesse
y


Trace des Pressions (x=13)
Pression
y


Lignes de Courant (h=0.5)
Euler
Runge-Kutta 4


Lignes de Courant (h=0.1)
Euler
Runge-Kutta 4