MoSV-1D
(MOmentum conservation Saint-Venant 1 Dimension)

Manual Book

MoSV 1D

By

© Telkom University

Version 1.0

By	:	Telkom University
Author	:	1. Hendro Pratama Saragih 2. Dr. Didit Adytia 3. Dede Tarwidi, M.Sc.
Address	:	Jl. Telekomunikasi No.1, Terusan Buah Batu, Kec. Dayeuhkolot Bandung, Jawa Barat 40257, Indonesia
Alamat e-mail	:	adytia@telkomuniversity.ac.id
Laman website	:	https://telkomuniversity.ac.id/

Telkom
University

Contents

Figure Contents

1. Introduction

This document is a guide to using and running MoSV 1D (Momentum Conservation Saint-Venant 1 Dimension) software. MoSV 1D software is based on the One Dimensional Shallow Water Equation (SWE 1D) model which is implemented numerically using the Finite Volume method with a staggered-grid scheme. The SWE equation and its numerical implementation, have been published in the following Journals and Proceedings:

1. Adytia, D. (2019). Momentum Conservative Scheme for Simulating Wave Runup and Underwater Landslide. Indonesian Journal on Computing (Indo-JC), 4(1), 29. https://doi.org/10.21108/indojc.2019.4.1.250

2. Adytia, D., Husrin, S., & Latifah, A. L. (2019). Dissipation of Solitary Wave Due To Mangrove Forest: A Numerical Study by Using Non-Dispersive Wave Model. ILMU KELAUTAN: Indonesian Journal of Marine Sciences, 24(1), 41. https://doi.org/10.14710/ik.ijms.24.1.41-50

3. Alfikri, M. Z., Adytia, D., & Subasita, N. (2019). Shock capturing staggered grid scheme for simulating dam-break flow and runup. Journal of Physics: Conference Series, 1192(1). https://doi.org/10.1088/1742-6596/1192/1/012041

2. Requirements

The requirements needed to run MoSV 1D are as follows :

2.1. System Requirements

2.1.1. Windows 10 All Version

Windows 10 is the operating system needed to run this application. This operating system is used because it is considered compatible on every user's device when using this application.

2.2. Program Requirements

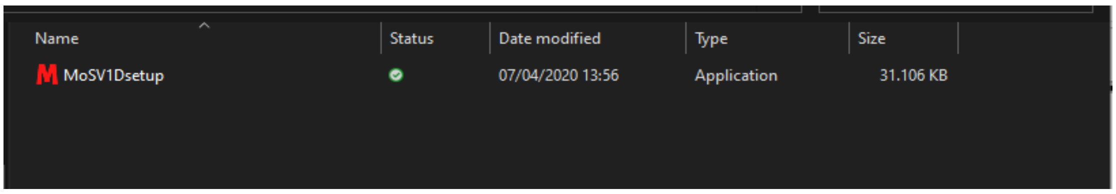
2.2.1. MoSV 1D Setup

MoSV Setup.exe is a program that needs to be installed to be able to run the MoSV 1D application. This application program is in the form of setup and is the main file for installation..

3. Installing

Before using the MoSV 1D application, we must first install the application. Following are the steps to install the MoSV 1D application :

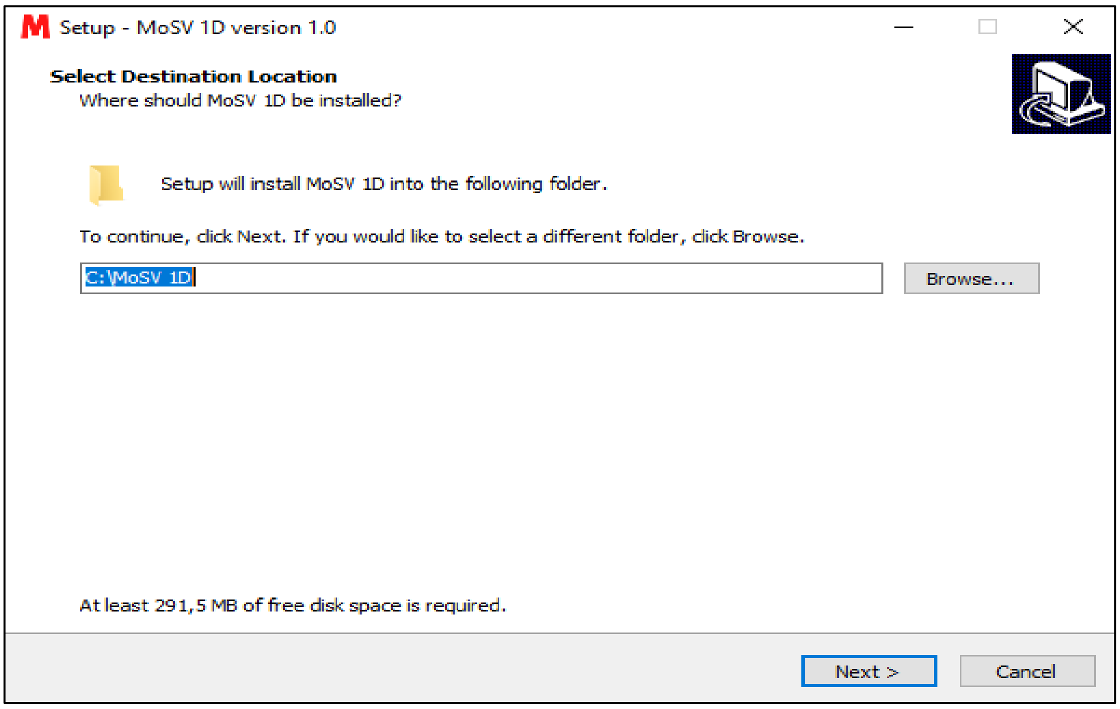
3.1. First double-click the MoSV 1D setup file and press **Run Administrator**.



Name	Status	Date modified	Type	Size
MoSV1Dsetup	✓	07/04/2020 13:56	Application	31.106 KB

Illustration 1 Setup MoSV 1D

3.2. Secondly click **Next**, the installation has been saved default in storage C.



Setup - MoSV 1D version 1.0

Select Destination Location

Where should MoSV 1D be installed?

Setup will install MoSV 1D into the following folder.

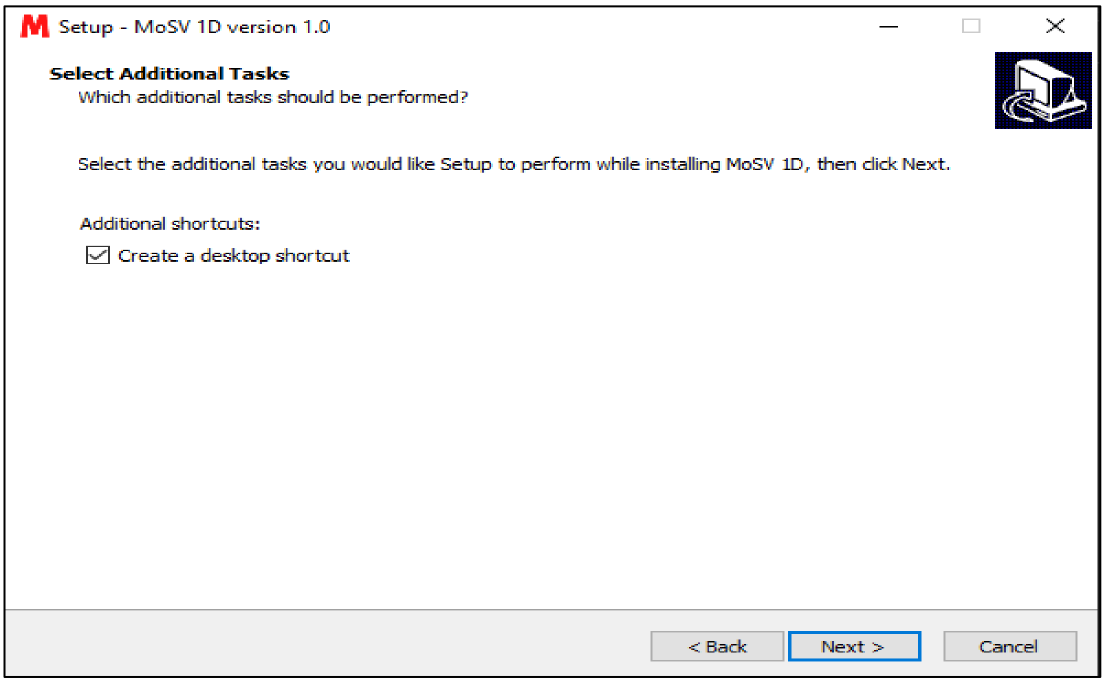
To continue, click Next. If you would like to select a different folder, click Browse.

C:\MoSV 1D

At least 291,5 MB of free disk space is required.

Illustration 2 Installation 1

3.3. Third click **Next** and check additional shortcuts to create applications on the your desktop.



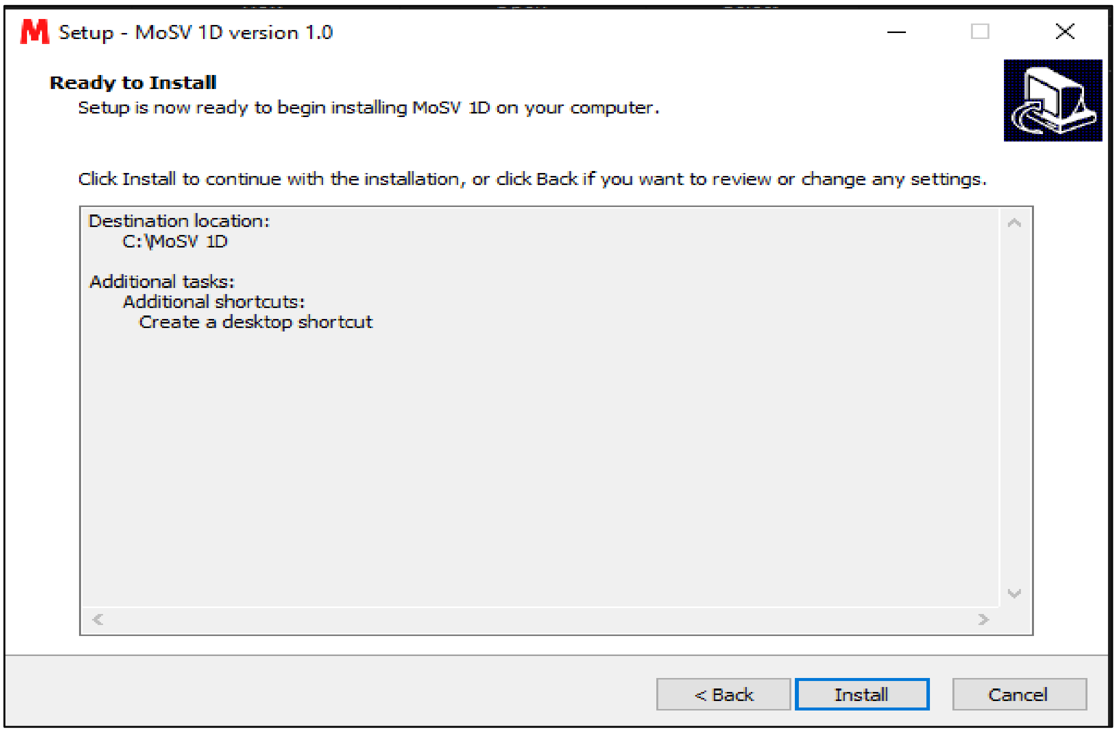
Select Additional Tasks
Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing MoSV 1D, then click Next.

Additional shortcuts:
☑ Create a desktop shortcut

Illustration 3 Installation 2

3.4. Fourth, press **Install** to install the MoSV 1D application.



Ready to Install
Setup is now ready to begin installing MoSV 1D on your computer.

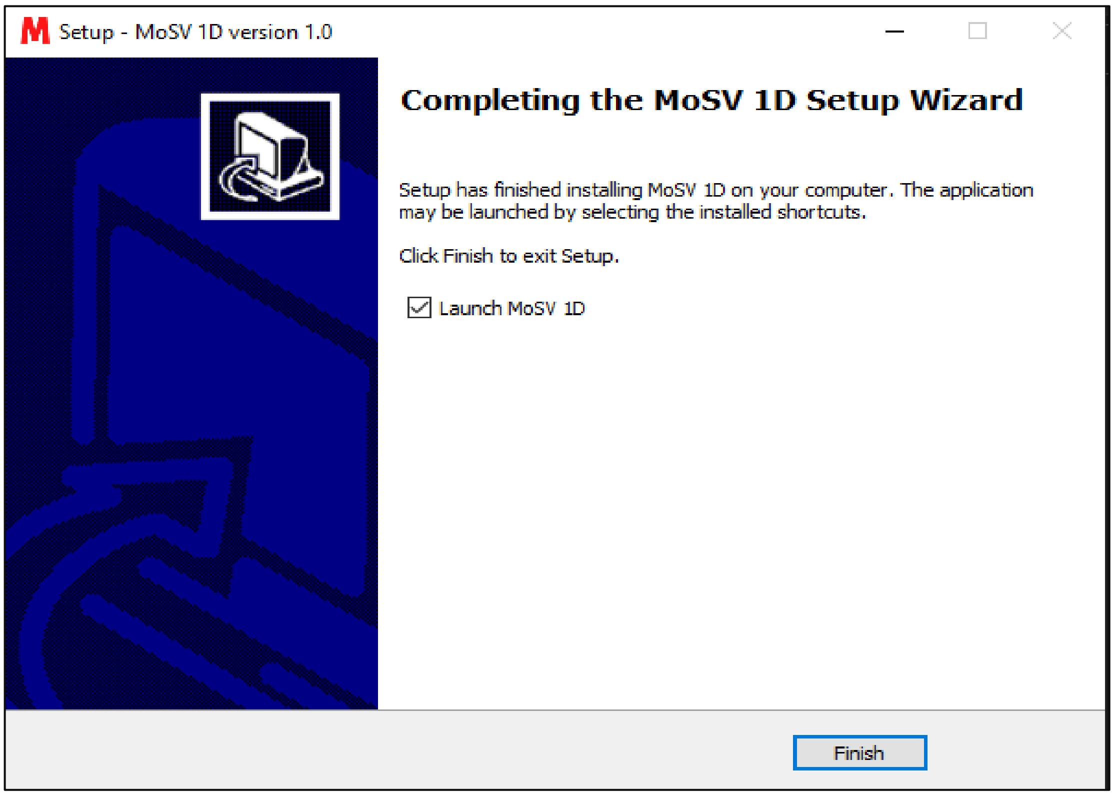
Click Install to continue with the installation, or click Back if you want to review or change any settings.

```
Destination location:
    C:\MoSV 1D

Additional tasks:
    Additional shortcuts:
      Create a desktop shortcut
```

Illustration 4 Installation 3

3.5. Finally press **Finish** when the installation process is complete.



Setup - MoSV 1D version 1.0

Completing the MoSV 1D Setup Wizard

Setup has finished installing MoSV 1D on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

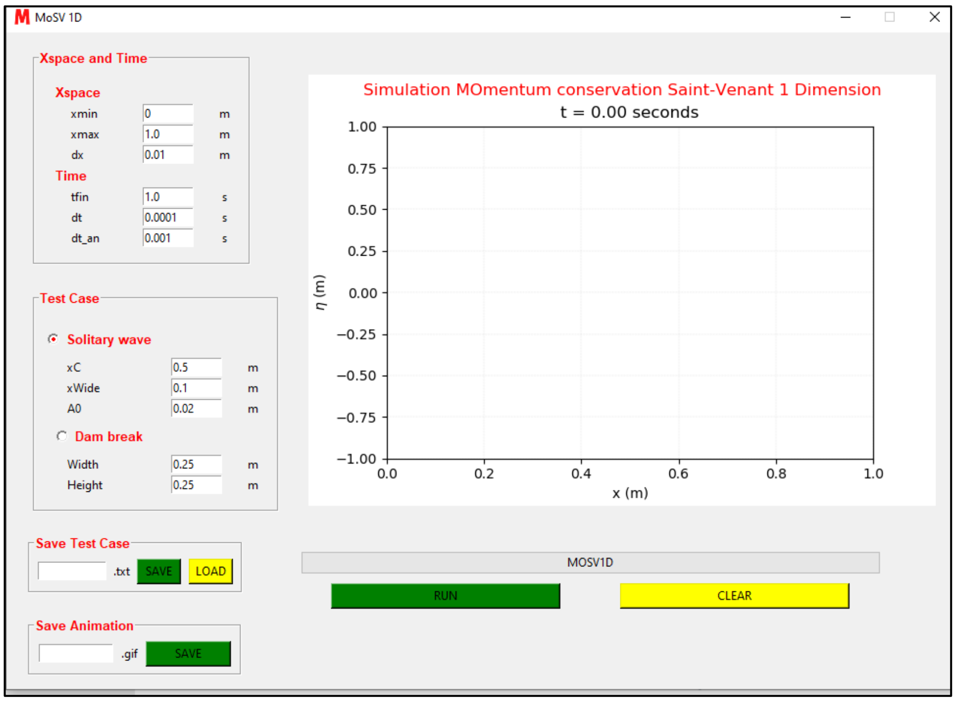
☑ Launch MoSV 1D

Illustration 5 Installation 4

4. Graphic User Interface (GUI)

The following are the parts of the MoSV 1D GUI (Graphical User Interface) :

4.1. Main GUI



MoSV 1D

Xspace and Time

Xspace

xmin	0	m
xmax	1.0	m
dx	0.01	m

Time

tfin	1.0	s
dt	0.0001	s
dt_an	0.001	s

Test Case

● Solitary wave

xC	0.5	m
xWide	0.1	m
A0	0.02	m

○ Dam break

Width	0.25	m
Height	0.25	m

Save Test Case

.txt SAVE LOAD

Save Animation

.gif SAVE

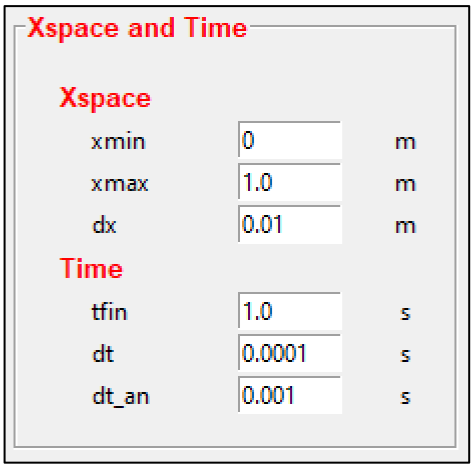
Simulation MOmentum conservation Saint-Venant 1 Dimension

t = 0.00 seconds

Illustration 6 Homepage

The picture above is the main display of the MoSV 1D application, there are 2 labels for input parameter namely Xspace & Time and Test Case. Then have several input entries and buttons.

4.2. Xspace and Time



Xspace and Time

Xspace

xmin	0	m
xmax	1.0	m
dx	0.01	m

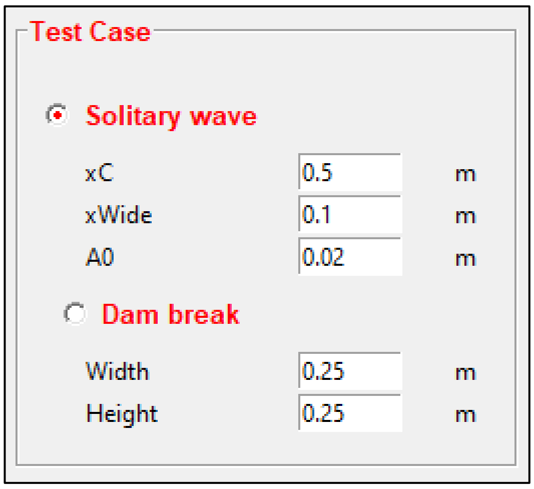
Time

tfin	1.0	s
dt	0.0001	s
dt_an	0.001	s

Illustration 7 Xspace and Time

Xspace and Time have 2 labels namely Xspace and Time, each of which has a different entry. Fill in all data in the provided entry fields using integer or float format. The Xspace section in illustration 7 contains three entry fields that must be filled. Xmin and Xmax in meters are the position of Xawal and Xakhir. For dx entries, it is a value that has a meter.

4.3. Test Case



Test Case

[✓ Solitary wave]

xC	0.5	m
xWide	0.1	m
A0	0.02	m

Dam break

Width	0.25	m
Height	0.25	m

Illustration 8 Test Case

The Test Case section in illustration 8, has 2 choices namely Solitary Wave and Dam Break. In the Solitary Wave option there is A0 in meters which means amplitude, xC in meters means position from the peak of Soliton and xWide is width in meters. Then the second option Dam Break is Width which is the width has meters and Height is height has units of meters

4.4. Save Test Case

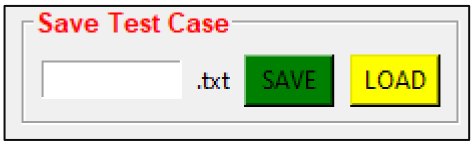


Illustration 9 Save Test Case

The Save Test Case section in illustration 9 has a function to save and load the parameter file that was filled in earlier. Before saving parameters, the user must fill in the file name. The file that is saved and loaded is in the .txt extension.

4.5. Save Animation

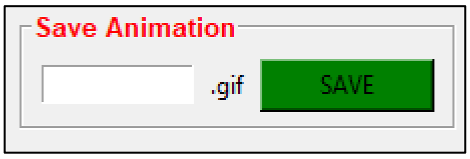


Illustration 10 Save Animation

The Save Animation section in illustration 10 has a function for saving files in the form of animation. In storing the animation file has an extension that is gif

4.6. Button

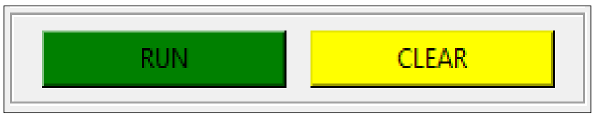


Illustration 11 Button

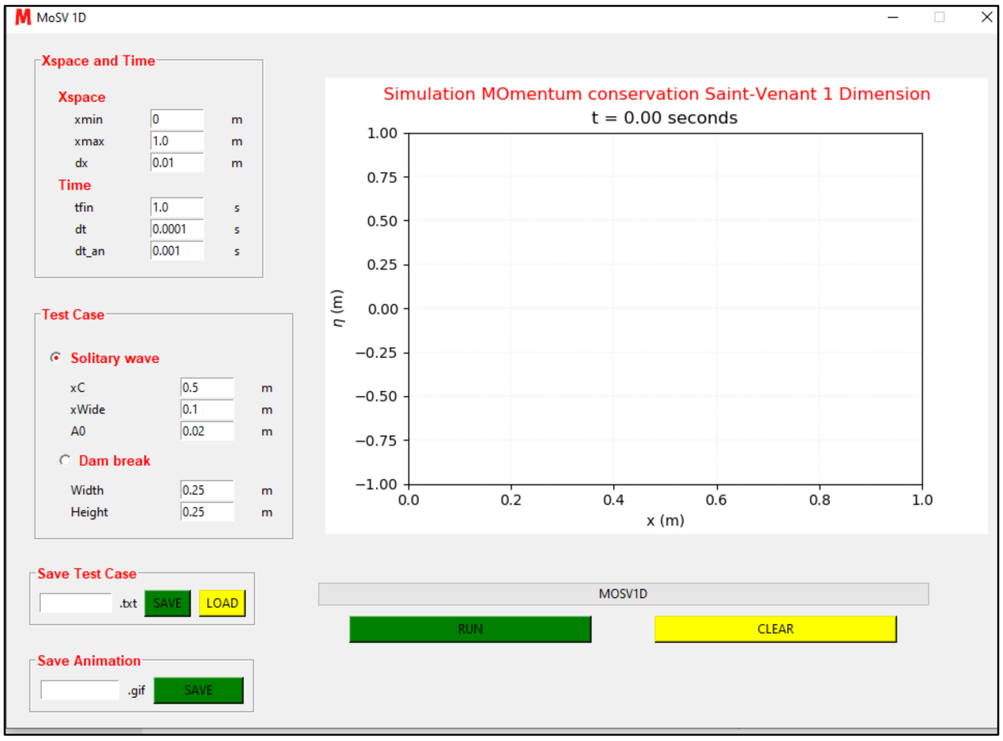
In the illustration section 11 has 2 buttons namely the RUN button and the CLEAR button. Run button functions to run the animation in the application based on parameters that have been filled. The CLEAR key functions to delete entries in the parameters, wave, bathymetry and time sections, which aim to speed up work in repeating new test cases and animations.

5. Operating

Here the example by running the software. There are 2 conditions to use this software : Running Animation and Save Test Case & Animation

5.1. Running Animation

5.1.1. Filled Data



Xspace and Time

Xspace

xmin	0	m
xmax	1.0	m
dx	0.01	m

Time

tfin	1.0	s
dt	0.0001	s
dt_an	0.001	s

Test Case

[✓ Solitary wave]

xC	0.5	m
xWide	0.1	m
A0	0.02	m

Dam break

Width	0.25	m
Height	0.25	m

Save Test Case

.txt SAVE LOAD

Save Animation

.gif SAVE

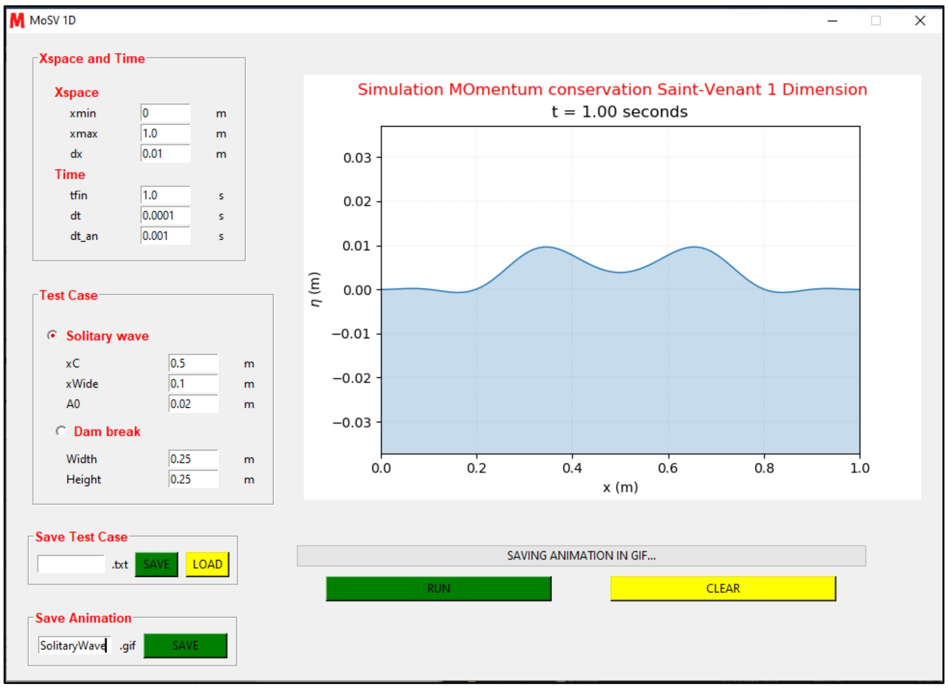
Illustration 12 Filled Data

First we fill all the data entries in the domain, wave, bathymetry, time and dam break sections. Then when all the data has been filled, press the green RUN button. Then wait for the animated image to appear on the blank canvas. The results of the animation of each parameter, will display the animation form as illustrated 16 (Result).

5.1.2. Result

The following is the result of each parameter, will display the results of the animation as below :

5.1.2.1 Result Solitary Wave



Xspace and Time

Xspace

xmin 0 m
xmax 1.0 m
dx 0.01 m

Time

tfin 1.0 s
dt 0.0001 s
dt_an 0.001 s

Test Case

(●) Solitary wave

xC 0.5 m
xWide 0.1 m
A0 0.02 m

( ) Dam break

Width 0.25 m
Height 0.25 m

Save Test Case

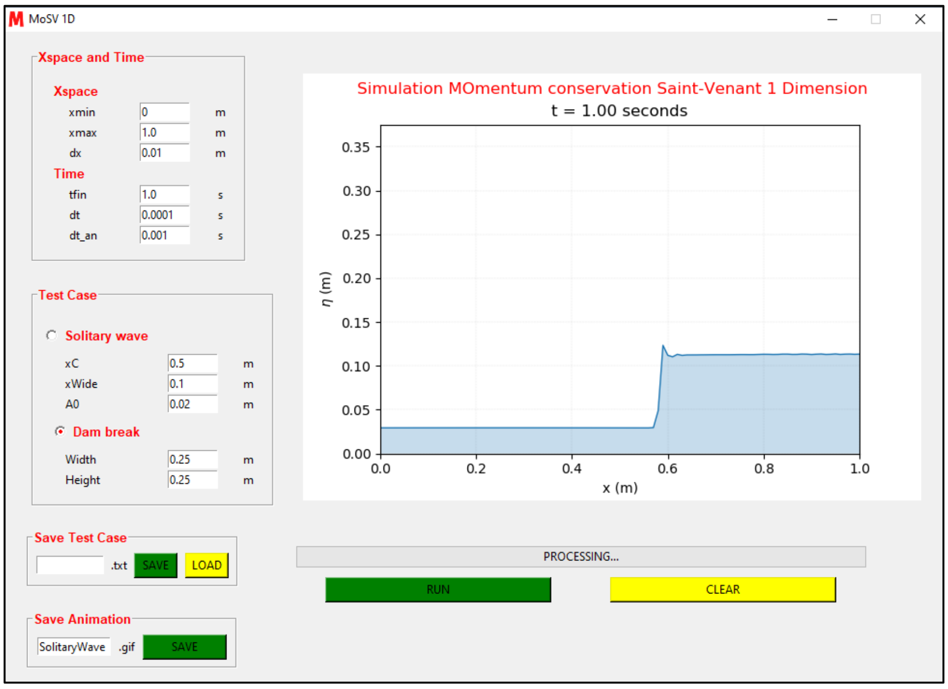
.txt

Save Animation

SolitaryWave .gif

Illustration 13 Result Solitary Wave

5.1.2.2 Result Dam Break



Xspace and Time

Xspace

xmin 0 m
xmax 1.0 m
dx 0.01 m

Time

tfin 1.0 s
dt 0.0001 s
dt_an 0.001 s

Test Case

( ) Solitary wave

xC 0.5 m
xWide 0.1 m
A0 0.02 m

(●) Dam break

Width 0.25 m
Height 0.25 m

Save Test Case

.txt

Save Animation

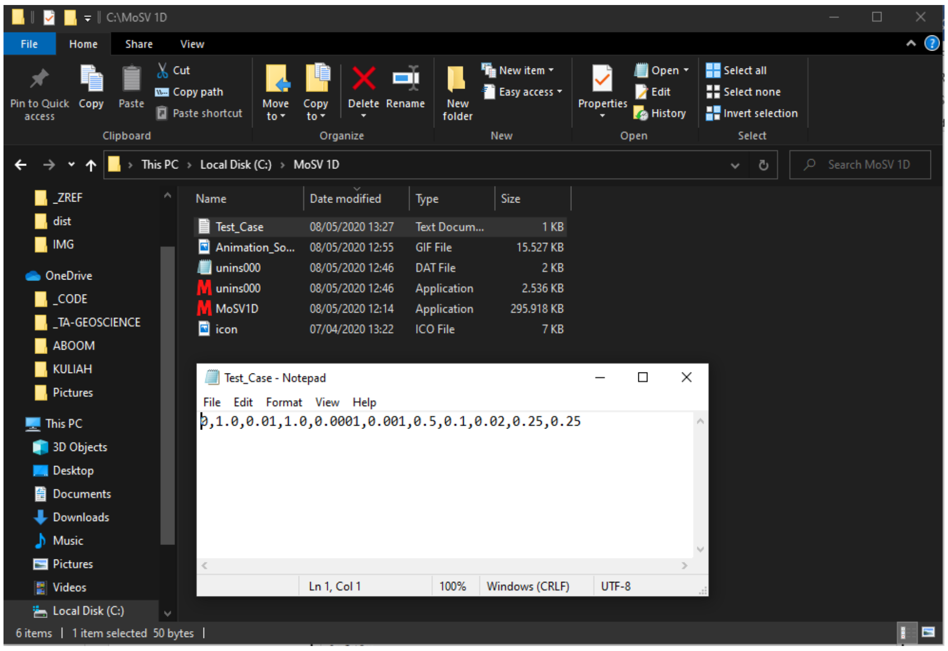
SolitaryWave .gif

Illustration 14 Result Dam Break

5.2. Additional Features

5.2.1. Save Test Case

Save all 1D MoSV parameters in the form of a text (.txt) file by giving the file name first. The following is an example of a test case file with the name TestCase_1 that has been saved.



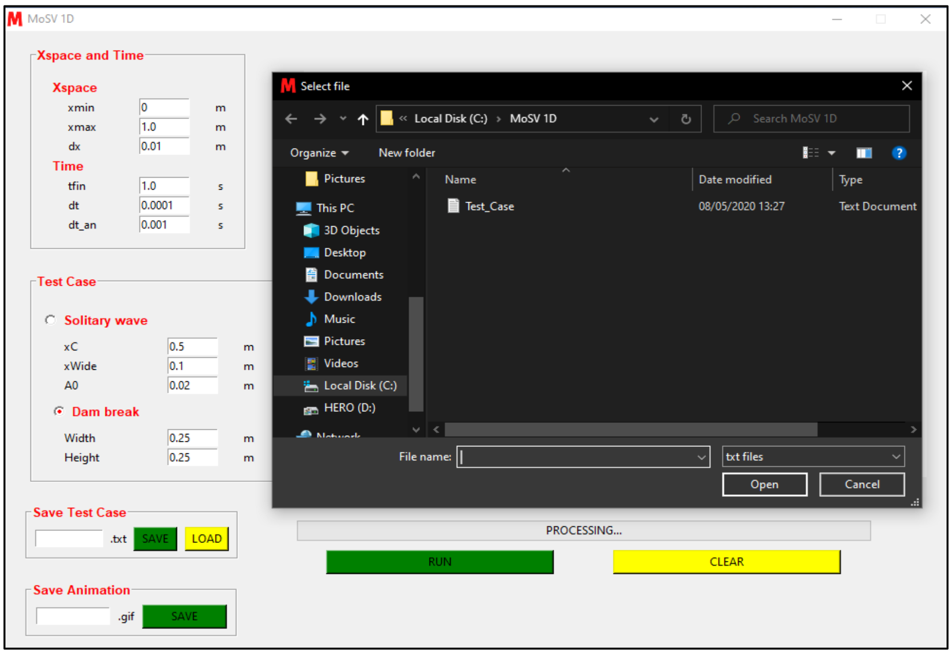
Name	Date modified	Type	Size
Test_Case	08/05/2020 13:27	Text Docum...	1 KB
Animation_So...	08/05/2020 12:55	GIF File	15.527 KB
unins000	08/05/2020 12:46	DAT File	2 KB
unins000	08/05/2020 12:46	Application	2.536 KB
MoSV1D	08/05/2020 12:14	Application	295.918 KB
icon	07/04/2020 13:22	ICO File	7 KB

```
0,1.0,0.01,1.0,0.0001,0.001,0.5,0.1,0.02,0.25,0.25
```

Illustration 15 Save & View Test Case

5.2.2. Load Test Case

Load the test case file that has been previously stored and then retrieve all the data contained in the file to be filled in to the GUI parameter entry and then it will be a requirement to run the animation.



xmin	0	m
xmax	1.0	m
dx	0.01	m
tfin	1.0	s
dt	0.0001	s
dt_an	0.001	s

Solitary wave

xC	0.5	m
xWide	0.1	m
A0	0.02	m

[✓ Dam break]

Width	0.25	m
Height	0.25	m

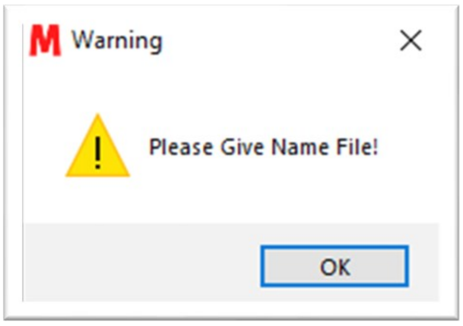
Name	Date modified	Type
Test_Case	08/05/2020 13:27	Text Document

File name: | txt files

Illustration 16 Load Test Case

5.2.3. Save Animation

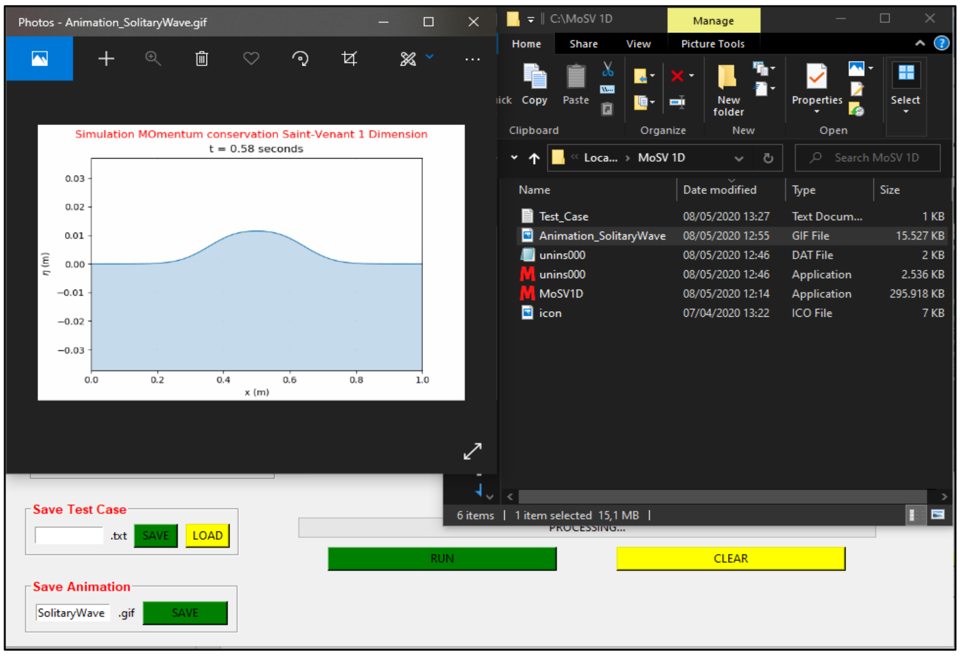
Save the results of the MoSV 1D animation in the form of a gif (.gif) file by giving the file name first. When pressing the save button before giving a name a pop-up will appear as follows :


Warning

Please Give Name File!

Illustration 17 Pop Up Warning

The following is an example of an animation file with the name Animation_1 that has been saved.



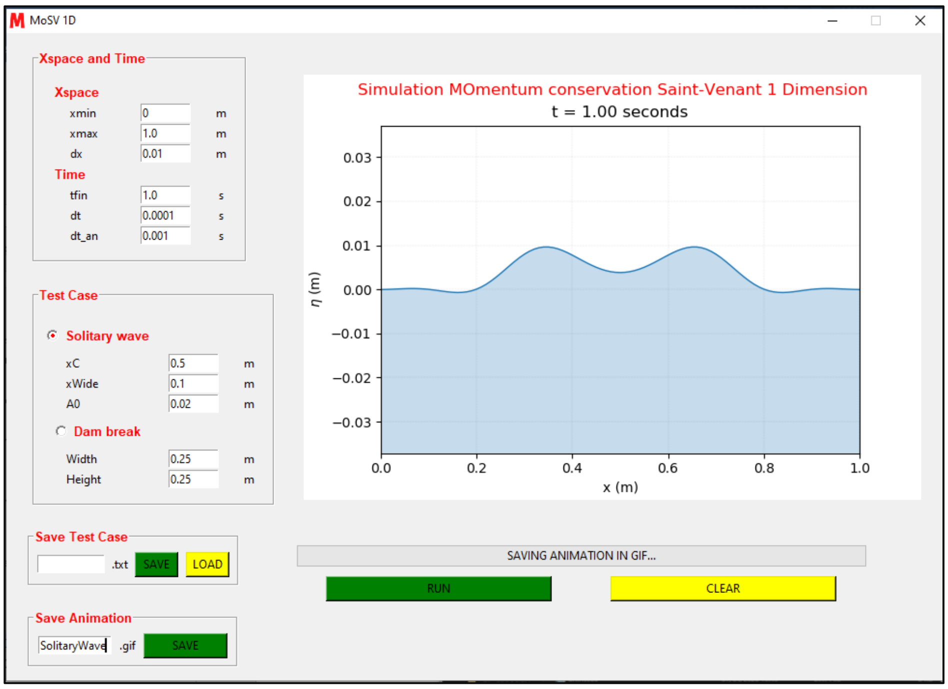
Name	Date modified	Type	Size
Test_Case	08/05/2020 13:27	Text Docum...	1 KB
Animation_SolitaryWave	08/05/2020 12:55	GIF File	15.527 KB
unins000	08/05/2020 12:46	DAT File	2 KB
unins000	08/05/2020 12:46	Application	2.536 KB
MoSV1D	08/05/2020 12:14	Application	295.918 KB
icon	07/04/2020 13:22	ICO File	7 KB

6 items | 1 item selected 15,1 MB |

Illustration 18 Save & View Animation

5.2.4. Run Animation

Running animation with a state where all parameters have been filled and pressing the RUN button so that the animation will appear in the form of canvas in the GUI.



MoSV 1D

Xspace and Time

Xspace

xmin	0	m
xmax	1.0	m
dx	0.01	m

Time

tfin	1.0	s
dt	0.0001	s
dt_an	0.001	s

Test Case

[✓ Solitary wave]

xC	0.5	m
xWide	0.1	m
A0	0.02	m

Dam break

Width	0.25	m
Height	0.25	m

Save Test Case: .txt SAVE LOAD

Save Animation: SolitaryWave .gif SAVE

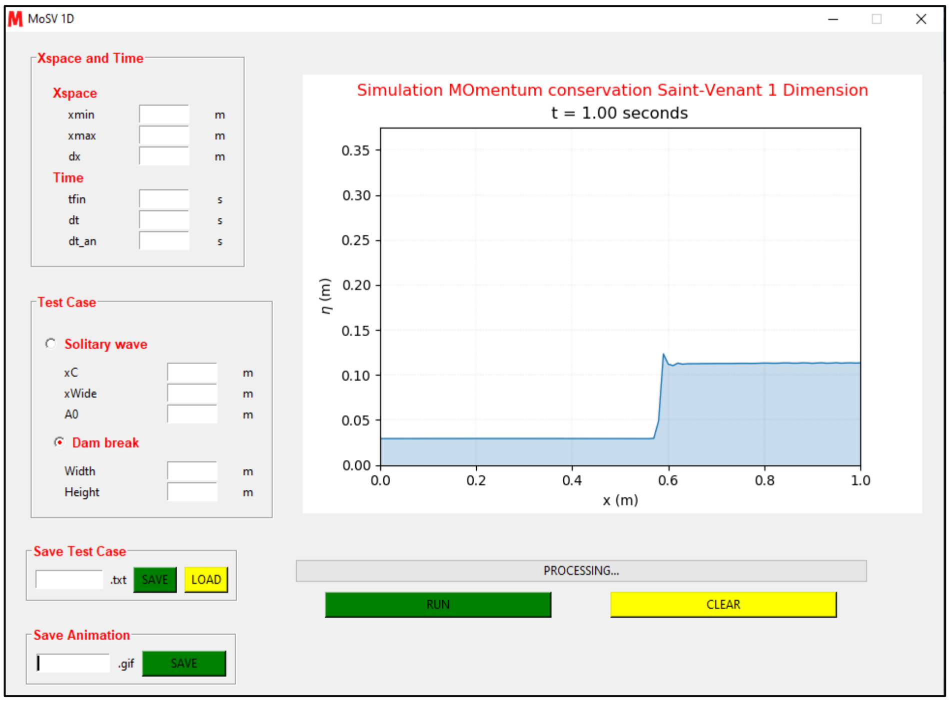
SAVING ANIMATION IN GIF...

RUN CLEAR

Illustration 19 Run Animation

5.2.5. Clear Input Parameter

Remove the contents of all entry parameters in the Xspace, Time, Solitary Wave and dam break except. This section aims to make it easier for us to replace all parameters at once. Here is the result of clear animation:



MoSV 1D

Xspace and Time

Xspace

xmin		m
xmax		m
dx		m

Time

tfin		s
dt		s
dt_an		s

Test Case

Solitary wave

xC		m
xWide		m
A0		m

[✓ Dam break]

Width		m
Height		m

Save Test Case: .txt SAVE LOAD

Save Animation: .gif SAVE

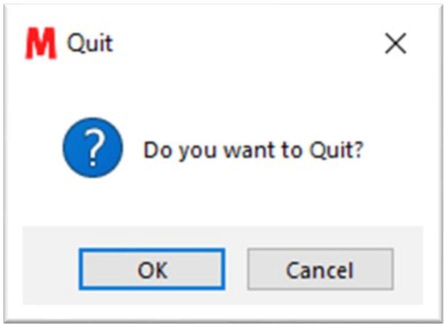
PROCESSING...

RUN CLEAR

Illustration 20 Clear Input Parameter

5.2.6. Close GUI

When pressing the close button on the top right, a Quit pop-up will appear. If the user asks OK then the application will exit, while pressing cancel returns the application. Here is a pop-up when pressing the close button on the application.



Quit

Do you want to Quit?

Illustration 21 Pop Up Quit

6. Test Result

6.1. Solitary Wave

The following is the result of Solitary Wave based on input parameters as shown below :

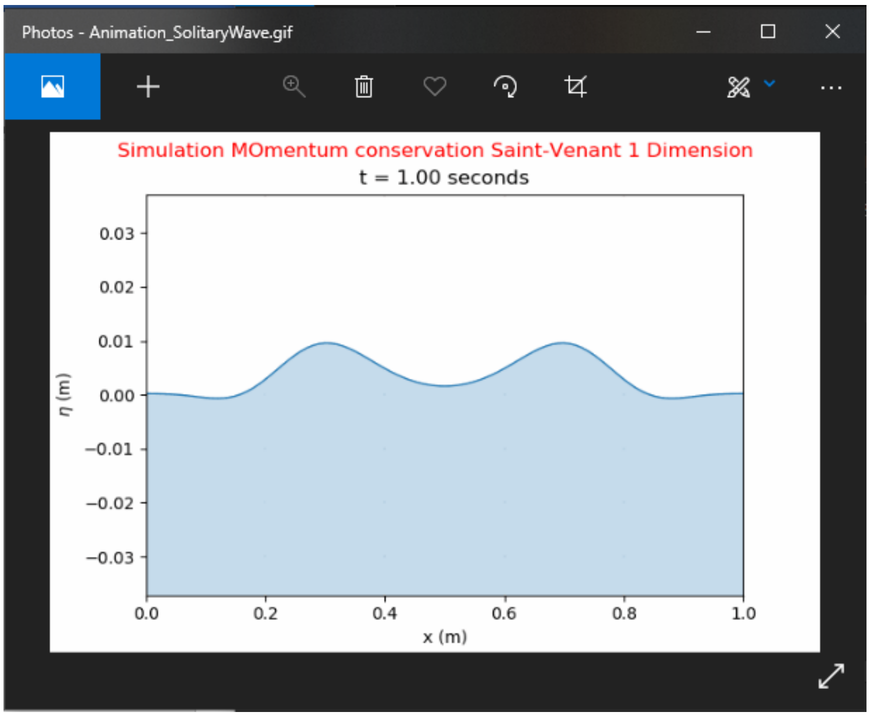


Illustration 22 Solitary Wave

6.2. Dam Break

The following is the result of Dam Break based on input parameters as shown below :

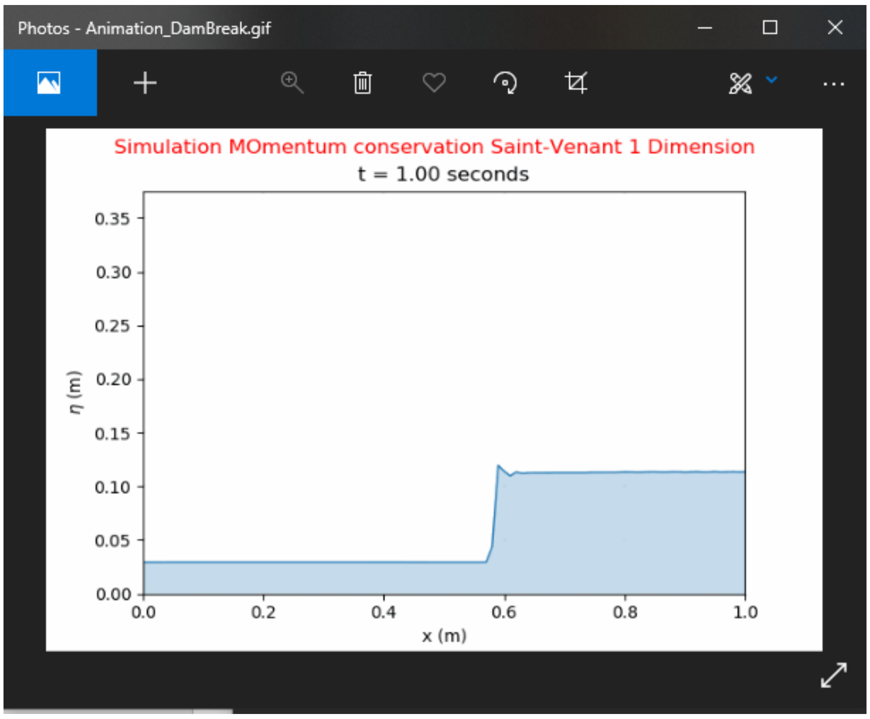


Illustration 23 Dam Break

7. Source Code

```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import tkinter as tk
from tkinter import *
from tkinter import ttk
import numpy as np
import matplotlib
matplotlib.use("TKAgg")
from tkinter import messagebox
from tkinter.filedialog import askopenfile
from tkinter.filedialog import askopenfilename
from tkinter.filedialog import asksaveasfilename
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import math as ma

import time

root = tk.Tk()
root.geometry("970x670")
root.resizable(0,0)

# ============================#
#                EXTERNAL FUNCTION              #
# ============================#

def initial_Condition(x, Nx, A0, xC, xwide, dam_height, dam_width, IC_option) :
        y  = [0]*(Nx+1)
        if IC_option == 'dam_break' :
                for i in range(0, Nx+1):
                        if x[i] <= dam_width:
                                y[i] = dam_height
                        else :
                                y[i] = 0.
        elif IC_option == 'solitary_wave' :
                y = A0*np.exp((-((x-xC)/xwide)*((x-xC)/xwide)))

        return  np.asarray(y)

def bathymetry(x, Nx, IC_option) :

        y  = [0]*(Nx+1)
        if IC_option == 'dam_break' :
                for i in range(0, Nx+1):
                        y[i] = 0.         # flat bottom

        elif IC_option == 'solitary_wave' :
                hshall = 0.02
                hC = 0.75
                hwide = 0.1
                y = 20 - hshall*np.exp((-((x-hC)/hwide)*((x-hC)/hwide)))

        return np.asarray(y)

def call_SWE(xleft, xright, Nx, dx, dt, dt_anim, tfin, A0, xC, xwide, dam_height, dam_width,
        IC_option) :
        # 0. INPUT PARAMETERS
        # wave initial condition
        x          = [0]*(Nx+1)
        x          = np.linspace(xleft, xright, Nx + 1)
        eta_list    = list()
        time_list  = list()

        # parameters
        g                  = 9.81
        eps_thin           = 0.0001
```

```
# 1. PREPARATION
h               = [0]*(Nx+1)         # bottom / bathymetry / topography
H               = [0]*(Nx+1)         # total depth in FULL grid at time n
H_new           = [0]*(Nx+1)         # total depth in FULL grid at time n+1
H_star          = [0]*(Nx+2)         # total depth in 'half-grid'
H_bar           = [0]*(Nx+2)         # averaged total depth at HALF grid

u               = [0]*(Nx+2)         # hor velocity in HALF grid at t=n+1
u_new           = [0]*(Nx+2)         # hor velocity in HALF grid at t=n+1
u_star          = [0]*(Nx+1)         # velocity in full grid
u_du            = [0]*(Nx+2)         # advection term in half-grid

q               = [0]*(Nx+2)         # approx. for h*u in HALF grid
q_bar           = [0]*(Nx+1)         # average of q1 in FULL grid
eta             = [0]*(Nx+1)         # wave elevation in FULL grid at time n
eta_new         = [0]*(Nx+1)         # wave elevation in FULL grid at time n+1
Ht         = [0]*(Nx+1)


# 2. INITIAL CONDITION & BATHYMETRY
eta = initial_Condition(x, Nx, A0, xC, xwide, dam_height, dam_width, IC_option)
# 2.1. Initial condition
h = bathymetry(x, Nx, IC_option) # 2.2. Bathymetry/bottom

t = 0
step = 0

H = h + eta  # 2.3. total depth in full grid

# MAIN CALCULATION; TIME STEPPING
max_iteration = ma.ceil(tfin/dt)
every = ma.ceil(dt_anim/dt)

# ----------- TREATMENT FOR RUNUP ---------------------------------------
eta = np.maximum(eta, -h)     # treatment for runup
H = eta + h                          # Water thickness
Ht = H[:]
ithin = np.where(H<eps_thin) # find indices for dry area (runup/overtopping)
Ht[ithin] = eps_thin       # thin layer is only used when solving

while (t <= tfin) :
        t = t + dt
        step += 1

        # UPDATE VARIABLES
```

```
H = eta + h                          # Water thickness
Ht = H[:]
ithin = np.where(H<eps_thin) # find indices for dry area
Ht[ithin] = eps_thin       # thin layer is only used when solving

# 1. CONTINUITY EQ.
# 1.1. Prepare for UPWIND of q=h*u
for j in range(1, Nx+1) :
        if (u[j] >= 0) :
                H_star[j]    = H[j-1]
        else :
                H_star[j]    = H[j]
        q[j] = H_star[j]*u[j]
# Neumann condition
H_star[0]      = H_star[1]
H_star[Nx+1]    = H_star[Nx]
# no-flux through boundary, or zero velocity
q[0]    = 0
q[Nx+1] = 0

for j in range(0, Nx+1) :
        eta_new[j] = eta[j] - (dt/dx)*(q[j+1]-q[j])
        H[j] = eta_new[j] + h[j]
        # Calculate q-bar (FLUX) for calculating (u-start upwind condition)
        # CALCULATE q-bar
        q_bar[j] = 0.5*(q[j] + q[j+1])
        # UPWIND for u-star
        if(q_bar[j] >= 0) :
                u_star[j]=u[j]
        else :
                u_star[j]=u[j+1]
# UPDATE H_new
H_new = h + eta_new # eta at time n+1  % UPDATED total depth
# 2. MOMENTUM EQ.
# 2.1. Horizontal Momentum
for j in range(1, Nx+1) :
        H_bar[j] = 0.5*(H[j] + H[j-1])
        if(H_bar[j] >= eps_thin) :  #for runup
                # Advection term - wet dry procedure using momentum
                  conservative scheme
                # Layer-1 (upper), HALF grid
                u_du[j] = (1./(H_bar[j]*dx))*((q_bar[j]*u_star[j] - q_bar[j-
                      1]*u_star[j-1]) - (u[j]*(q_bar[j] - q_bar[j-1])))
                # Calc full eq. of 2nd dynamic eq. (Hydrostatic)
                u_new[j] = u[j] - (dt/dx*g*( eta_new[j] - eta_new[j-1] )) –
                      (dt*u_du[j]) #Layer-1
```

```
                        else : # if the total depth is not positive (no water) --> no velocity
                                u_new[j] = 0
                # hardwall boundary conditions
                u_new[0] = 0
                u_new[Nx+1] = 0

                # updating values of H, eta, u
                eta   = eta_new
                H     = H_new
                u     = u_new


                if (step % every == 0) :
                        eta_list.append(eta_new[:])
                        time_list.append(t)

        return (eta_list, time_list)


# ==============================#
#                 EXTERNAL FUNCTION               #
# ==============================#

def on_closing():
  if messagebox.askokcancel("Quit", "Do you want to Quit?"):
    root.destroy()

class Window(tk.Frame):

        def __init__(self, master):
                tk.Frame.__init__(self, master)
                self.master = master
                self.init_window()
                self.running = False
                self.ani = None

        def start(self):
                # clear list
                self.eta_list.clear()
                self.time_list.clear()

                self.Axleft=float(self.xleft.get())
                self.Axright=float(self.xright.get())
                self.Adx=float(self.dx.get())
                self.ANx= ma.ceil((self.Axright - self.Axleft)/self.Adx)
self.Atfin=float(self.tfin.get())
self.Adam_width=float(self.dam_width.get())
self.Adam_height=float(self.dam_height.get())

self.AA0=float(self.A0.get())
self.AxC=float(self.xC.get())
self.Axwide=float(self.xwide.get())

self.Adt=float(self.dt.get())
self.Adt_anim = float(self.dt_anim.get())



self.x  = [0]*(self.ANx+1)
self.x = np.linspace(self.Axleft, self.Axright, self.ANx + 1)
self.eta_list, self.time_list = call_SWE(self.Axleft, self.Axright, self.ANx, self.Adx,
        self.Adt, self.Adt_anim, self.Atfin, self.AA0, self.AxC, self.Axwide,
        self.Adam_height, self.Adam_width, self.IC_option)
self.eta_len = len(self.eta_list)
if self.IC_option == 'dam_break' :
        self.ymax = 1.5*np.amax(self.eta_list[1][:])
        self.ymin = 0.0
else :
        self.ymax = 2*np.amax(self.eta_list[1][:])
        self.ymin = -self.ymax

fig, (ax1) = plt.subplots(1, 1, figsize=(6.4, 4.4))
fig.suptitle('Simulation MOmentum conservation Saint-Venant 1 Dimension',
        color='r', fontsize=12)

def animate(i):
        ax1.cla()
        ax1.set_title("t = %3.2f seconds" %self.time_list[i])
        ax1.fill_between(self.x, self.ymin, self.eta_list[i][:], alpha=0.25)
        ax1.axis((self.Axleft, self.Axright,self.ymin, self.ymax))
        ax1.grid(True)
        ax1.grid(color='black', linestyle='-.', linewidth=0.2, alpha=0.2)
        ax1.set(xlabel='x (m)', ylabel=r'$\eta$' + ' (m)')
        ax1.plot(self.x, self.eta_list[i][:], lw=1)

self.canvas = FigureCanvasTkAgg(fig, master=self.frame8)
self.canvas.get_tk_widget().grid(column=0,row=0, padx=10)
self.ani = animation.FuncAnimation(fig, animate, np.arange(1, self.eta_len),
        interval=25, blit=False, repeat=False)
# self.running = True
# self.B5.config(text='RUN.')
```

```
            self.ani._start()

    def on_click(self):
            # if self.ani is None:
                    self.progressBar['maximum'] = 100
                    for i in range(101):
                            self.style.configure('text.Horizontal.TProgressbar',
                                    text='PROCESSING...')
                            time.sleep(0.01)
                            self.progressBar["value"] = i
                            self.progressBar.update()
                            self.progressBar["value"] = 0
                    return self.start()

    def clear_text(self):
            self.E0.delete(0, 'end')
            self.E1.delete(0, 'end')
            self.E2.delete(0, 'end')
            self.E3.delete(0, 'end')
            self.E4.delete(0, 'end')
            self.E5.delete(0, 'end')
            self.E6.delete(0, 'end')
            self.E7.delete(0, 'end')
            self.E8.delete(0, 'end')
            self.E9.delete(0, 'end')
            self.E10.delete(0, 'end')


    def validate_float(self, action, index, value_if_allowed,
      prior_value, text, validation_type, trigger_type, widget_name):
      # action=1 -> insert
      if(action=='1'):
        if text in value_if_allowed:
          try:
            float(value_if_allowed)
            return True
          except ValueError:
            return False
        else:
          return False
      else:
        return True

    def save(self):
      file = self.Ename.get()
      param0=self.E0.get()
```

```
        param1=self.E1.get()
        param2=self.E2.get()
        param3=self.E3.get()
        param4=self.E4.get()
        param5=self.E5.get()
        param6=self.E6.get()
        param7=self.E7.get()
        param8=self.E8.get()
        param9=self.E9.get()
        param10=self.E10.get()
        with open(file + '.txt', 'w') as file_object:
          file_object.write(param0)
          file_object.write(',')
          file_object.write(param1)
          file_object.write(',')
          file_object.write(param2)
          file_object.write(',')
          file_object.write(param3)
          file_object.write(',')
          file_object.write(param4)
          file_object.write(',')
          file_object.write(param5)
          file_object.write(',')
          file_object.write(param6)
          file_object.write(',')
          file_object.write(param7)
          file_object.write(',')
          file_object.write(param8)
          file_object.write(',')
          file_object.write(param9)
          file_object.write(',')
          file_object.write(param10)

    def loadfile(self):
        filename = filedialog.askopenfilename(title = "Select file",filetypes = (("txt
                files","*.txt"),("all files","*.*")))
        print(filename)
        readfile = open(filename, "r")
        for line in readfile:
              Type = line.split(",")

              self.xleft.set(Type[0])
              self.xright.set(Type[1])
              self.dx.set(Type[2])
              self.tfin.set(Type[3])
              self.dt.set(Type[4])
```

```
            self.dt_anim.set(Type[5])
            self.xC.set(Type[6])
            self.xwide.set(Type[7])
            self.A0.set(Type[8])
            self.dam_width.set(Type[9])
            self.dam_height.set(Type[10])


    def save_animate_gif(self):

            file_name_gif = self.Aniname_gif.get()
            if len(self.Aniname_gif.get())==0:
                    messagebox.showwarning("Warning", "Run First & Give The File
                            Name!")
            else:
                    self.progressBar['maximum'] = 100
                    self.style.configure('text.Horizontal.TProgressbar', text='SAVING
                            ANIMATION IN GIF...')
                    for i in range(101):
                            time.sleep(0.005)
                            self.progressBar["value"] = i
                            self.progressBar.update()
                            self.progressBar["value"] = 0
                    animation_gif = self.ani.save(file_name_gif+'.gif',
                            writer='imagemagick', extra_args=['-vcodec', 'libx264'])
                    return animation_gif

    def ICselection(self):
            if self.varIC.get() == 1 :
                    self.IC_option = 'solitary_wave'
            else :
                    self.IC_option = 'dam_break'

    def init_window(self):

            # ================================#
            #                PARAMETER INPUT                #
            # ================================#

            self.IC_option = 'solitary_wave'
            self.eta_list = list()
            self.time_list = list()

            self.running = False
            self.ani     = None
```

```
self.xleft   = StringVar()
self.xright  = StringVar()
self.Nx      = StringVar()
self.dx      = StringVar()
self.tfin    = StringVar()
self.A0      = StringVar()
self.xC      = StringVar()
self.xwide   = StringVar()
self.dt      = StringVar()
self.dt_anim = StringVar()
self.hC      = StringVar()
self.h0      = StringVar()
self.hshall  = StringVar()
self.hwide   = StringVar()

self.dam_width  = StringVar()
self.dam_height = StringVar()

self.file_name    = StringVar()
self.ani_name_mp4 = StringVar()
self.ani_name_gif = StringVar()

self.master.title("MoSV 1D")
root.iconbitmap("icon.ico")
 vcmd = (self.register(self.validate_float),'%d', '%i', '%P', '%s', '%S', '%v', '%V',
            '%W')

# -------------------------------------------#
#         PARAMETER INPUT                     #
# -------------------------------------------#
tk.Label(self,text="Simulation of SWE 1D",pady=10, padx=50, font
          ="Helvetica 20").grid(column=0, row=0,columnspan=2)

border=Frame(root, pady=5 , padx=2)
border.grid(row=0,column=0,padx=15,pady=10,columnspan=2)

self.frame = LabelFrame(border, text = "Xspace and Time", font='Helvetica
           10 bold', fg='red', padx=8, pady=15)
self.frame.grid(row=0, column=0, padx=10,pady=2,sticky=W)


self.LX = Label(self.frame, text='Xspace', font='Helvetica 10 bold', fg='red')
self.LX.grid(row=0, column=0,columnspan=2, sticky=W, padx= 10)

self.L0 = Label(self.frame, text='xmin').grid(row=1, column=0, sticky=W,
           padx= 26)
```

```
self.E0 = tk.Entry(self.frame, textvariable=self.xleft, width=8,
            validatecommand = vcmd)
self.E0.grid(row=1, column=1, padx= 10)
self.E0.insert(0,'0')
self.S0 = Label(self.frame, text='m').grid(row=1, column=2,padx=8)

self.L1 = Label(self.frame, text='xmax').grid(row=2, column=0, sticky=W,
            padx= 26)
self.E1 = tk.Entry(self.frame, textvariable=self.xright, width=8,
            validatecommand = vcmd)
self.E1.grid(row=2, column=1, padx= 10)
self.E1.insert(0,'1.0')
self.S1 = Label(self.frame, text='m').grid(row=2, column=2,padx=8)

self.L2 = Label(self.frame, text='dx').grid(row=3, column=0, sticky=W,
            padx= 26)
self.E2 = tk.Entry(self.frame, textvariable=self.dx, width=8, validatecommand
            = vcmd)
self.E2.grid(row=3, column=1, padx= 10)
self.E2.insert(0,'0.01')
#self.E3.configure(state='readonly')
self.S2 = Label(self.frame, text='m').grid(row=3, column=2,padx=8)



# -------------------------------------------#
#                Time                        #
# -------------------------------------------#

self.LY = Label(self.frame, text='Time',font='Helvetica 10 bold', fg='red')
self.LY.grid(row=4, column=0, sticky=W, padx= 10,columnspan=2)

self.L3 = Label(self.frame, text='tfin').grid(row=5, column=0, sticky=W,
         padx= 26)
self.E3 = Entry(self.frame, textvariable=self.tfin, width=8, validatecommand
         = vcmd)
self.E3.grid(row=5, column=1, padx= 15)
self.E3.insert(0,'1.0')
self.S3= Label(self.frame, text='s').grid(row=5, column=2,padx=8)

self.L4 = Label(self.frame, text='dt').grid(row=6, column=0, sticky=W,
        padx= 26)
self.E4 = Entry(self.frame, textvariable=self.dt, width=8, validatecommand =
        vcmd)
self.E4.grid(row=6, column=1, padx= 15)
self.E4.insert(0,'0.0001')
#self.E12.configure(state='readonly')
```

```
self.S4 = Label(self.frame, text='s').grid(row=6, column=2,padx=6)

self.L5 = Label(self.frame, text='dt_an').grid(row=7, column=0, sticky=W,
        padx= 26)
self.E5 = Entry(self.frame, textvariable=self.dt_anim, width=8,
         validatecommand = vcmd)
self.E5.grid(row=7, column=1, padx= 15)
self.E5.insert(0,'0.001')
#self.E13.configure(state='readonly')
self.S5 = Label(self.frame, text='s').grid(row=7, column=2,padx=6)

self.frames = Frame(border, padx=8, pady=15)
self.frames.grid(row=1, column=0, padx=10,pady=10,sticky=W)

# --------------------------------------------#
#                Wave Initial condition          #
# --------------------------------------------#
self.frame1 = LabelFrame(border, text="Test Case", font='Helvetica 10 bold',
       fg='red', padx=8, pady=15)
self.frame1.grid(row=2, column=0, padx=10,pady=2,sticky=W)

self.varIC = IntVar()
self.varIC.set(1)

self.radiobtn1 = Radiobutton(self.frame1, text="Solitary wave",
       font='Helvetica 10 bold', fg='red', variable=self.varIC, value=1,
       command=self.ICselection)
self.radiobtn1.grid(row=0, column=0, padx=0, pady=5)

#self.L4 = Label(self.frame1, text='Solitary wave', font='Helvetica 10 bold',
        fg='red')
#self.L4.grid(row=2, column=0, columnspan=2, sticky=W, padx= 10)

self.L6 = Label(self.frame1, text='xC').grid(row=1, column=0, sticky=W,
        padx= 22)
self.E6 = tk.Entry(self.frame1, textvariable=self.xC, width=8,
        validatecommand = vcmd)
self.E6.grid(row=1, column=1, padx= 15)
self.E6.insert(0,'0.5')
self.S6 = Label(self.frame1, text='m').grid(row=1, column=2,padx=8)

self.L7 = Label(self.frame1, text='xWide').grid(row=2, column=0, sticky=W,
         padx= 22)
self.E7 = Entry(self.frame1, textvariable=self.xwide, width=8,
         validatecommand = vcmd)
self.E7.grid(row=2, column=1, padx= 15)
```

```
        self.E7.insert(0,'0.1')
        self.S7 = Label(self.frame1, text='m').grid(row=2, column=2,padx=8)

        self.L8 = Label(self.frame1, text='A0').grid(row=3, column=0, sticky=W,
            padx= 22)
        self.E8 = Entry(self.frame1, textvariable=self.A0, width=8, validatecommand
            = vcmd)
        self.E8.grid(row=3, column=1, padx= 15)
        self.E8.insert(0,'0.02')
        self.S8 = Label(self.frame1, text='m').grid(row=3, column=2,padx=8)

        # ------------------------------------------#
        #                   Dam Break               #
        # ------------------------------------------#

        self.radiobtn2 = Radiobutton(self.frame1, text="Dam break", font='Helvetica
           10 bold', fg='red', variable=self.varIC, value=2,
           command=self.ICselection, anchor=tk.W,
           justify=tk.LEFT)
        self.radiobtn2.grid(row=4, column=0, padx=0, pady=5)

        self.L9 = Label(self.frame1, text='Width').grid(row=5, column=0, sticky=W,
            padx= 22)
        self.E9 = Entry(self.frame1, textvariable=self.dam_width, width=8,
            validatecommand = vcmd)
        self.E9.grid(row=5, column=1, padx= 15)
        self.E9.insert(0,'0.25')
        self.S9= Label(self.frame1, text='m').grid(row=5, column=2,padx=8)

        self.L10 = Label(self.frame1, text='Height').grid(row=6, column=0,
            sticky=W, padx= 22)
        self.E10 = Entry(self.frame1, textvariable=self.dam_height, width=8,
            validatecommand = vcmd)
        self.E10.grid(row=6, column=1, padx= 15)
        self.E10.insert(0,'0.25')
        self.S10 = Label(self.frame1, text='m').grid(row=6, column=2,padx=8)

    #==========================================#
    #               SAVE & LOAD FILE           #
    #==========================================#

        self.frames1 = Frame(border, padx=8, pady=15)
        self.frames1.grid(row=4, column=0, padx=10,pady=10,sticky=W)

        self.frame5 = LabelFrame(border,text="Save Test Case",font='Helvetica 10
            bold', fg='red', padx=4,pady=7)
            self.frame5.grid(row=5, column=0,padx=5, columnspan=2, sticky=W)

            # self.Lname = Label(self.frame5, text='Test Case').grid(row=0, column=0,
                  padx= 4)
            self.Ename = Entry(self.frame5, textvariable=self.file_name, width=11)
            self.Ename.grid(row=1, column=0, padx= 4)
            self.Pname = Label(self.frame5, text='.txt').grid(row=1, column=1)
            self.B0 = Button(self.frame5,
                  text='SAVE',width=5,bg='green',command=self.save)
            self.B0.grid(row=1, padx=4,column=2)
            self.B1 = Button(self.frame5,
                  text='LOAD',width=5,bg='yellow',command=self.loadfile)
            self.B1.grid(row=1, padx=4,column=3)

    #========================== ==#
    #                 SAVE ANIMATION                  #
    #=========================== =#
            self.frames1 = Frame(border, padx=8, pady=15)
            self.frames1.grid(row=6, column=0, padx=10,pady=10,sticky=W)

            self.frame6 = LabelFrame(border,text="Save Animation", font='Helvetica 10
                 bold', fg='red', padx=4,pady=7)
            self.frame6.grid(row=7, column=0,padx=5, columnspan=2, sticky=W)

            self.Aniname_gif = Entry(self.frame6, width=12)
            self.Aniname_gif.grid(row=0, column=1, padx= 5)
            self.Pname_gif = Label(self.frame6, text='.gif').grid(row=0, column=2)
            self.B0 = Button(self.frame6,
                  text='SAVE',width=11,bg='green',command=self.save_animate_gif)
            self.B0.grid(row=0, padx=5,column=3)

            #================= =====#
            #                 BUTTON                 #
            #=============== ======#

            self.frame7 = Frame(border,padx=8,pady=6)
            self.frame7.grid(row=5, column=1, rowspan=2 ,padx=5,pady=2,sticky=W)

            self.style = ttk.Style(root)
            self.style.layout('text.Horizontal.TProgressbar',
                    [('Horizontal.Progressbar.trough',
                     {'children': [('Horizontal.Progressbar.pbar',
                              {'side': 'left', 'sticky': 'ns'})],
                      'sticky': 'nswe'}),
                     ('Horizontal.Progressbar.label', {'sticky': ''})])
```

```
                self.style.configure('text.Horizontal.TProgressbar', text='MOSV1D')
                self.progressBar = ttk.Progressbar(self.frame7, orient="horizontal",
                     length=590,mode="determinate", style='text.Horizontal.TProgressbar')
                self.progressBar.grid(column = 0, row = 0, pady=10, columnspan=2)

                self.B5 =
                Button(self.frame7,text='RUN',width=32,bg='green',command=self.on_click)
                self.B5.grid(row=1, padx=6,column=0)
                self.B1 =
                Button(self.frame7,text='CLEAR',width=32,bg='yellow',command=
                self.clear_text)
                self.B1.grid(row=1, padx=6,column=1)

        #====================== ====== #
        #              PLOT & ANIMATION           #
        #======================== ==#

                self.frame8 = Frame(border,padx=10,pady=10)
                self.frame8.grid(row=0, column=1, rowspan=5, padx=0,pady=0,sticky=W)

                self.Axleft=float(self.xleft.get())
                self.Axright=float(self.xright.get())
                self.Adx=float(self.dx.get())
                self.ANx= ma.ceil((self.Axright - self.Axleft)/self.Adx)

                self.x  = [0]*(self.ANx+1)
                self.x = np.linspace(self.Axleft, self.Axright, self.ANx + 1)

                fig, (ax1) = plt.subplots(1, 1, figsize=(6.4, 4.4))

                ax1.set_title("t = 0.00 seconds")
                ax1.axis((self.Axleft, self.Axright, -1, 1))
                ax1.grid(True)
                ax1.grid(color='black', linestyle='-.', linewidth=0.2, alpha=0.2)
                ax1.set(xlabel='x (m)', ylabel=r'$\eta$' + ' (m)')

                fig.suptitle('Simulation MOmentum conservation Saint-Venant 1 Dimension',
                       color='r', fontsize=12)

                self.canvas = FigureCanvasTkAgg(fig, master=self.frame8)
                self.canvas.get_tk_widget().grid(column=0,row=0, padx=10)

root.protocol("WM_DELETE_WINDOW", on_closing)
Window(root)
root.mainloop()
```