Documentation

James Lamb, Pawel Manikowski

17 November 2018

Q2. Strings handling

"Create a class called **StringContainer**, initialised it with a string containing only spaces and alpha-numerical characters. . . "

Firstly, the alpha-numerical characters needs to be defined:

"(. . .) alphanumeric characters are those comprised by the combined set of the 26 alphabetic characters, A to Z, and the 10 Arabic numerals, 0 to 9." (https://whatis.techtarget.com/definition/alphanumeric-alphameric)

Therefore, all other characters will be treated as exception. Assumption has been made that at least one space character and one alphanumeric character were used.

The following methods have been used:

- for the *split* method, we used *split()*
- for the *check_occurence(str)* and the *get_frequency(str)* method, we used *count(str)* function
- for the *replace(str,newstr)* method we used *replace(old,new)* function
- for the *mirror* method we used *[::-1]* function

A code `[all(x.isalnum() or x.isspace() for x in self.text)]` was copied from:
(https://stackoverflow.com/questions/29460405/checking-if-string-is-only-letters-and-spaces-python)

(a) Exception

Consider:

```
example2 = StringContainer("blah $ blah")
```

Since $ is neither an alpha-numerical character or space the exception is thrown:

```
raise Exception("String does not contain alphanumerical characters and spaces.")
Exception: String does not contain alphanumerical characters and spaces.
```

As expected.

(b) split() method

The following string has been used for all methods:

```
example = StringContainer("In the middle of the 1984 I was walking down the street")
```

Output of the `example.split()`:

```
The string:
 In the middle of the 1984 I was walking down the street
 contains the following words:

['In', 'the', 'middle', 'of', 'the', '1984', 'I', 'was', 'walking', 'down', 'the', 'street']
```

(c) check_occurence(str) method

Check if the `example` string contains word "down".

Output of the `example.check_occurance("down")`:

```
Checking occurances for the word:  down
Out[9]: True
```

Output of the `example.check_occurance("up")`:

```
Checking occurances for the word:  up
False
Out[13]: False
```

(d) get frequency(str) method

Returns the number of occurence of the "the" string in the whole message.

Output of the `example.get_frequancy("the")`:

```
Number of occurances of the word:  the  : 3
```

As expected.

(e) replace(str, newstr) method

The string "the" was replaced with the string "blah" in this example:

Output of the `example.replace("the","blah")`:

```
Old string:  In the middle of the 1984 I was walking down the street
the  will be replaced with  blah
New string:  In blah middle of blah 1984 I was walking down blah street
```

(f) merge(obj) method

For this method operator + was used.

Output of the `example.merge("in the valley....")`:

```
In the middle of the 1984 I was walking down the street in the valley....
```

(g) mirror method

For this method $[:: -1]$ function was used.

Output of the `example.mirror()`:

```
 The mirror of the:
 In the middle of the 1984 I was walking down the street
 is:
teerts eht nwod gniklaw saw I 4891 eht fo elddim eht nI
```

Q3. Guitar string simulation

(a) Function that returns the analytical solution

The following equation needs to be solved for the position x of the string, for time t and for $n_{\max}$:

$$y(x,t) = \lim_{n_{\max}\to\infty} \sum_{n=1}^{n_{\max}} A_n \sin(k_n x)\cos(\omega_n t)$$

Where:

$$A_n = \frac{2hL^2}{\pi^2 n^2 d(L-d)\sin(\frac{n\pi d}{L})}$$

Therefore the final function for the analytical solution is the following:

$$y(x,t) = \sum_{n=1}^{n_{\max}} \frac{2hL^2 \sin(k_n x)\cos(\omega_n t)}{\pi^2 n^2 d(L-d)\sin(\frac{n\pi d}{L})} \tag{1}$$

Where:

- L - length of the string (1 m)
- h - height of the string (0.005 m)
- d - $L/10$

The function (1) has been implemented in the following way:

```
def yy(x,t,n):
    if x > L or t < 0:
        raise Exception("x must be less or equal L (in our case 1) and t must be more or equal to 0")
    yxt = 0.0
    for i in range(1,n):
        yxt = yxt + (2*h*L**2*math.sin((i*math.pi*d)/L)*math.sin((i*math.pi*x)/L)
        *math.cos((c*i*math.pi*t)/L))/(math.pi**2*i**2*d*(L-d))
    return yxt
```

Initial condition, at time $t = 0$:

$$f(x) = \begin{cases} \frac{hx}{d} & 0 \leqslant x \leqslant d \\ \frac{(L-x)h}{L-d} & d < x \leqslant L \end{cases} \tag{2}$$

Implementation of the (2):

```
def init(x):
    initx=0.0
    if x > 0 and x <= d:
        initx = h*x/d
    if x > d  and x <= L:
        initx = ((L-x)*h)/(L-d)
    return initx
```

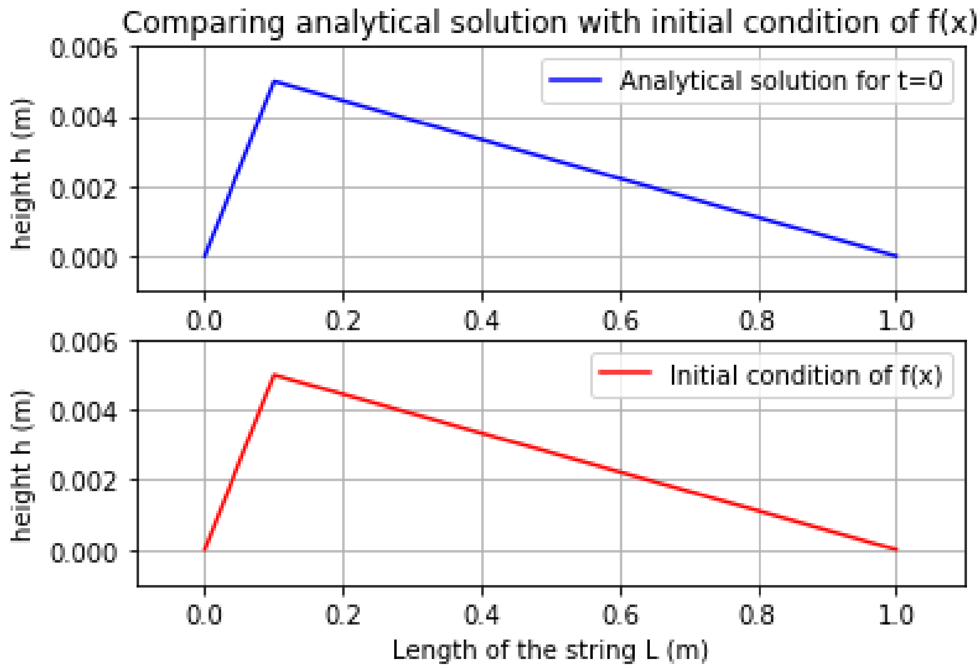
The following results have been obtained for time $t = 0$, $n = 1000$ and the following values of x:

- $x = 0.1$: analytical solution: 0.0004999999534605353, initial condition: 0.0005
- $x = 0.03$: analytical solution: 0.0014999998347647126, initial condition: 0.0014999999999999998
- $x = 0.077$: analytical solution: 0.0038500021149159317, initial condition: 0.0038499999999999997

It can be seen that the above values are almost identical therefore it can be conclued that function **yy(x,t,n)** returns expected values.

(b) Plot of initial condition and analytical solution

Functions (1) and (2) have been plotted for $t = 0$ and $n = 1000$:



It can be seen that these two plots match which confirms our previous observations.

(c) Function that implements the algorithm

Algorithm 1: Algorithm to solve the equation of motion of a one-dimensional string

Solver step $(y, i, \Delta x, \Delta t, Nx, c)$;
Input : an array y, i the index of the current timeslice, dx the spatial step size, dt the temporal step size, Nx the number steps in the spatial direction, c the parameter of the wave equation.
Output: a new vector u describing the position of the string at time $\Delta t(i+1)$

Set $\alpha = c * \Delta t / \Delta x$
Initialise u to a vector of the appropriate length containing zeros.
if $\alpha > 1$ **then**
    α should be smaller than 1 ! Exitraxiing !
else
    for *all positions* j *except for* $j = 0$*and* $j = Nx + 1$ *(the boundaries are fixed).* **do**
        if $i = 0$ **then**
            $u[j] = -y[i,j] + 2(1-\alpha^2)y[i,j] + \alpha^2\,(y[i,j+1] + y[i,j-1])$
        else
            $u[j] = -y[i-1,j] + 2(1-\alpha^2)y[i,j] + \alpha^2\,(y[i,j+1] + y[i,j-1])$
        end
    end
end
return u

Where $c = 2Lf$, since $L = 1$ and $f = 440$ then $c = 880$. To ease the analysis of the convergence of the **Algoritm 1** $\Delta t = \frac{\Delta x}{c}$, Δx must be equal to $\frac{L}{Nx}$ and since $\alpha = c\frac{\Delta t}{\Delta x}$ then $\alpha = 1$.

The implementation of the **Algorihtm 1**:

```
def Solver_steper(y,i,dx,dt,Nx,c):
    alpha = c*dt/dx
    u = np.zeros(Nx)
    if alpha > 1:
        raise Exception("alpha (c*dt/dx) must be less than 1")
    else:
        for j in range(1,Nx):
            if i == 0:
                u[j] = -y[i,j]+2*(1-alpha**2)*y[i,j]+alpha**2*(y[i,j+1]+y[i,j-1])
            else:
                u[j] =-y[i-1,j]+2*(1-alpha**2)*y[i,j]+alpha**2*(y[i,j+1]+y[i,j-1])
    return u
```

(d) Prediction of the motion of the string and plot of the residual

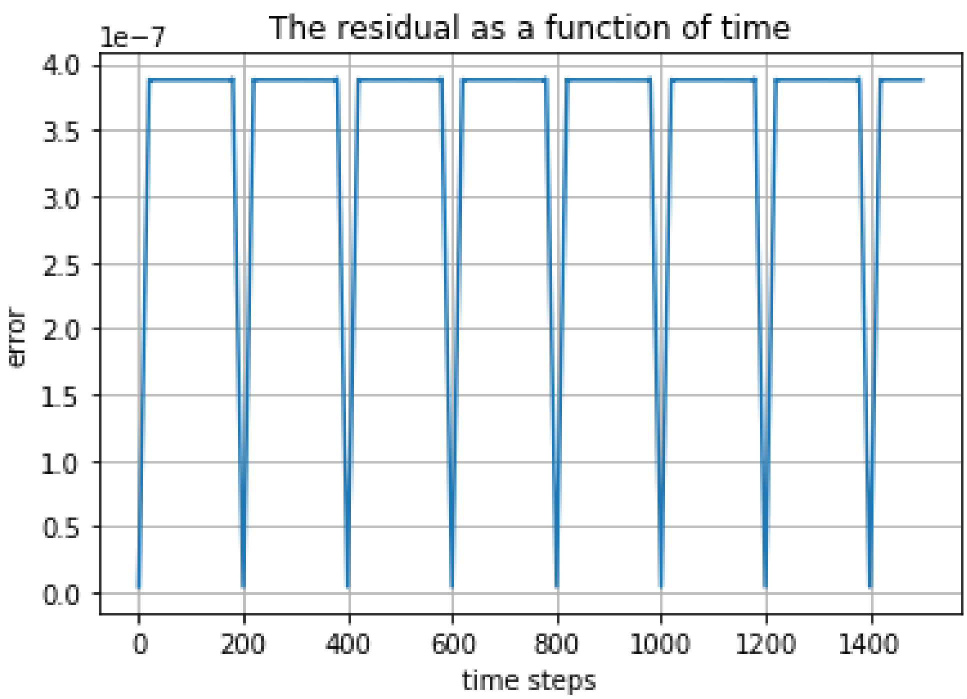
The **Algorithm 1** was used for calculation of 1500 time steps and retured the matrix:

```
[[ 0.00000000e+00  2.50000000e-04  5.00000000e-04 ...  5.55555556e-05
   2.77777778e-05  0.00000000e+00]
 [ 0.00000000e+00  2.50000000e-04  5.00000000e-04 ...  5.55555556e-05
   2.77777778e-05  0.00000000e+00]
 [ 0.00000000e+00  2.50000000e-04  5.00000000e-04 ...  5.55555556e-05
   2.77777778e-05  0.00000000e+00]
 ...
 [ 0.00000000e+00 -2.77777778e-05 -5.55555556e-05 ...  5.55555556e-05
   2.77777778e-05  0.00000000e+00]
 [ 0.00000000e+00 -2.77777778e-05 -5.55555556e-05 ...  5.55555556e-05
   2.77777778e-05  0.00000000e+00]
 [ 0.00000000e+00 -2.77777778e-05 -5.55555556e-05 ...  5.55555556e-05
   2.77777778e-05  0.00000000e+00]]
```

The residual, which is a standard measure of the difference between the numerical and the analytical solution is defined as:

$$r(i\Delta t) = r_i = \sum_j = (y(i,j) - y(i,j)_{\text{analytical}})^2 \tag{3}$$

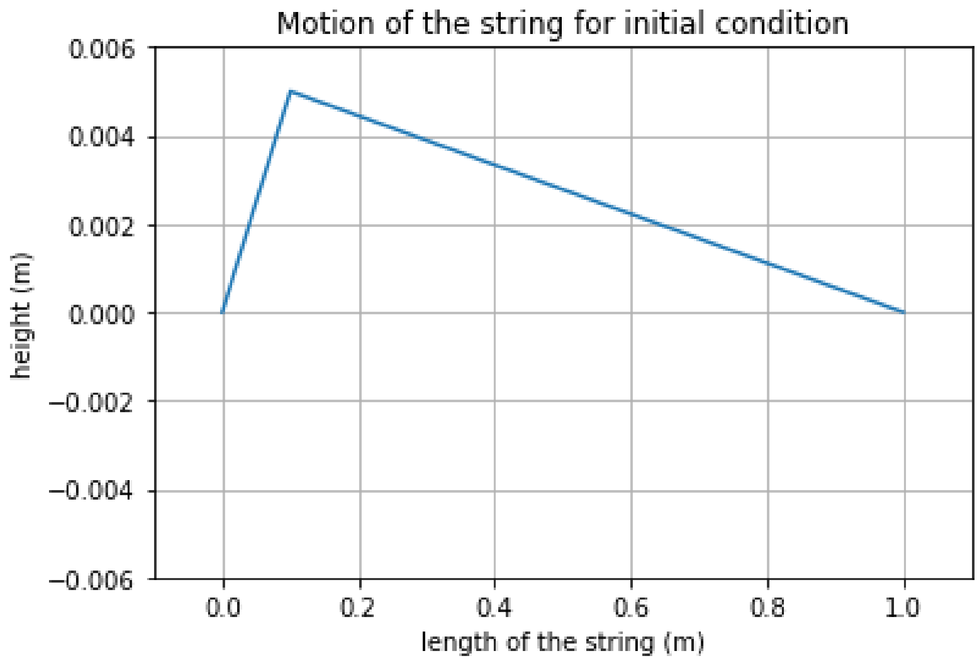
Plot of the residual as a function of time (in our case 1500 time steps):



It can be noticed that the errors are not bigger than 4×10^{-7} which means that the **Algoritm 1** provides a very good approximation.

(e) Animation illustrating the motion of the string

First animation, **init_cond.mp4**, was produced for the initial condition (2). First frame of the animation:



Second animation, **stat_wave.mp4**, was produced for the initial condition:

$$f_n(x) = A \sin\left(\frac{xn\pi}{L}\right)$$

Where $A = 0.005$ and $n = 3$. First frame of the animation: