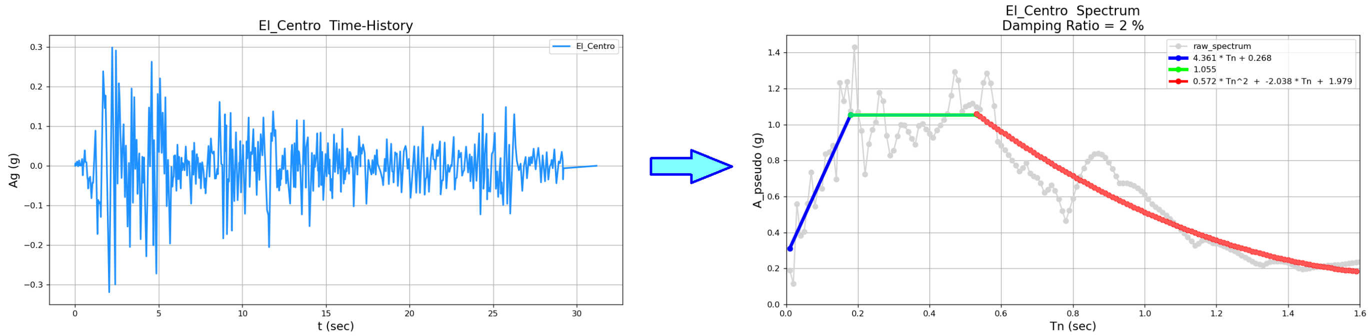
# *Generate_Spectrum*

### *A tool to generate "Spectrum" from "Time-History" of an EarthQuake*




In [1]:

```python
# (auto)
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
```

## reading the Time-History

In [2]:

```python
# (input) Time-History & dt & rD (damping-ratio)

Time_History = pd.read_csv(
    'https://raw.githubusercontent.com/AmirPeimon/repo1/main/El_Centro_TH.csv',header=

dt = 0.02  # (second)
rD = 0.02  # damping ratio
g  = 9800  # mm/sec2
Tn_Range = np.arange(0.05,1.65,0.05)

Earth_Quake_Name = 'El_Centro'

print( Time_History.head(3) )
```

```
   El_Centro
0    0.00000
1    0.00630
2    0.00364
```

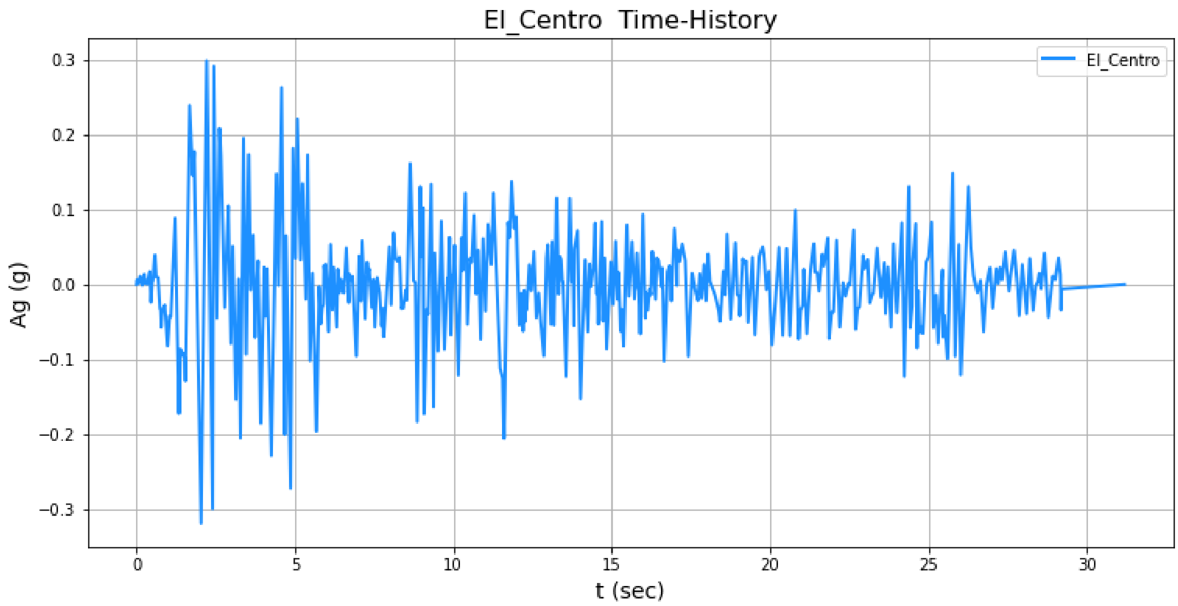
In [3]:

```python
# (auto)  Completing Data
Ag = g*Time_History.values
t  = np.array( Time_History.index*dt, 'float64' )
```

## Visualizing the Ground Acceleration Time-History

In [4]:

```python
# (auto) plot
fig,ax = plt.subplots( figsize=(12.5,6) )
### Time-History of Ground acceleration
ax.plot(  t, Ag/g, '-', color='dodgerblue', linewidth=2 )
# Decoration
plt.title(Earth_Quake_Name+'  Time-History',fontsize=16)
plt.xlabel('t (sec)',fontsize=14)
plt.ylabel('Ag (g)',fontsize=14)
plt.legend([ Earth_Quake_Name ])
plt.grid('on')
plt.savefig(Earth_Quake_Name+'_TH.png', dpi=120)
plt.show()
```




## Solving The Differential Equation of Motion

## u" + 2.rD.wn.u' + wn^2.u = -Ag

- u:displacement v:velocity a:acceleration
- y = [u, v]
- du/dt = v
- dv/dt = a = -2.rD.wn.u' - wn^2.u - Ag

In [5]:

```
# define a function that solves the Earthquake Differential Equation of Motion:
#
#                       u" + 2.rD.wn.u' + wn**2.u = -Ag
#
# for known:
#     damping ratio, rD
#     natural period, Tn
#     ground-acceleration time-history, Ag(t)
#
# and returns a dataframe ( uva ) containing:
#    displacements, u(t)
#    velocity, v(t)
#    total acceleration,    a_total(t)    =  (-wn**2)*u(t) + (-2*rD*wn)*v(t)
#    pseudo acceleration,   a_pseudo(t)   =  (-wn**2)*u(t)

def find_uva(rD,Tn,Ag):
    ...
    return uva
```

## Generating and Saving The Raw Spectrum

In [7]:

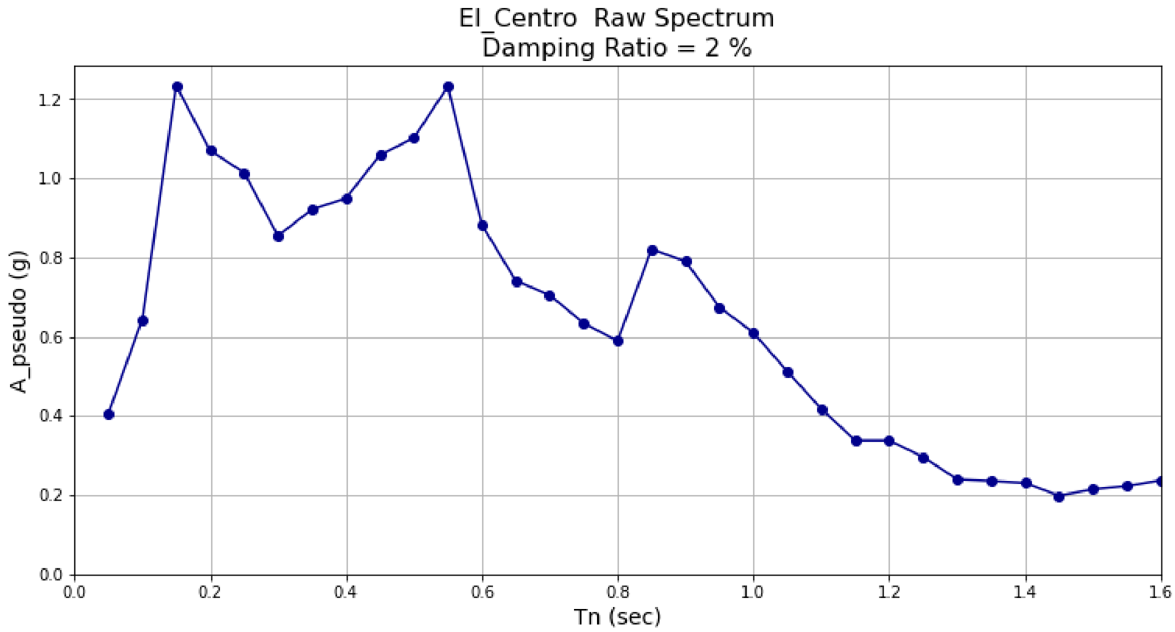
```
# (auto) define a function called Make_Spectrum
# that solves equation of motion for a range of natural periods,
# and returns their corresponding pseudu accelerations

def Make_Spectrum( rD, Tn_Range, Ag ):
    ...
    return Spectrum_Data

# Making Raw Spectrum
Spectrum_Data = Make_Spectrum( rD, Tn_Range, Ag )
# Checking
print( '\n\n', Spectrum_Data.head(3) )
```

```
 [-4.39724310e-02 -1.93089061e+00  1.03565240e+00  7.22564825e-01]
 ...
 [-5.93921249e+01 -2.64125305e+01  9.80227391e+02  9.75944686e+02]
 [-5.97239535e+01 -6.76278090e+00  9.82493932e+02  9.81397369e+02]
 [-5.96628821e+01  1.28559757e+01  9.78309275e+02  9.80393829e+02]]



 1.6
 [[ 0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00]
 [-1.23287412e-02 -1.23159532e+00  3.83583225e-01  1.90124686e-01]
 [-4.39820273e-02 -1.93173934e+00  9.81695047e-01  6.78258141e-01]
 ...
 [-5.37947812e+01 -1.81008707e+02  8.58015921e+02  8.29583140e+02]
 [-5.72397846e+01 -1.63323789e+02  9.08364263e+02  8.82709422e+02]
 [-6.03215164e+01 -1.44700708e+02  9.52963133e+02  9.30233599e+02]]
   Tn (sec)   A_ps (g)   A_total (g)    D (mm)   V (mm/sec)
0      0.05   0.404675     0.403531   0.251138    15.721895

1      0.10   0.643196     0.638001   1.596650    76.125158
2      0.15   1.234028     1.221401   6.892453   258.409157
```

## Visualizing Raw Spectrum

In [9]:

```
...
```




## Defining 3 Major Parts of Spectrum:

- Part 1: Displacement-Ralated: a_pseudo = a + b.Tn
- Part 2: Velocity-Related: a_pseudo = Mean
- Part 3: Acceleration-Related: a_pseudo = A.Tn^2 + B.Tn + C

In [10]:

```
# (input)
# Defining preliminary range of each Part
# overlaps are considered to synchronize final pieces

Part_1_Init, Part_1_End = 0.01, 0.22
Part_2_Init, Part_2_End = 0.15, 0.58
Part_3_Init, Part_3_End = 0.47, 1.60
```

In [11]:

```
# (auto)
# Part 1:   Displacement-Related-Part
# A straight line (regression-line) will represent this part:  a_pseudo=a+b*Tn
#
# define a function called  find_linear_reg_line
# to find a & b of regression-line  a_seudo=a+b*Tn  for Part_1

def find_linear_reg_line( Init, End, Spectrum_Data ):
    ...
    return [ a,b ]

[a,b] = find_linear_reg_line( Part_1_Init, Part_1_End, Spectrum_Data )
print(a,b)
```

```
0.19128336543309077 5.1734884922307245
```

In [12]:

```
# (auto)
# Part 2:   Velocity-Related-Part
# average (Mean) of all values will represent this part:  a_pseudo=Mean
#
# define a function called  find_mean
# that finds the mean of Part_2:  a_pseudo=mean

def find_mean( Init, End, Spectrum_Data ):
    ...
    return Mean

Mean = find_mean( Part_2_Init, Part_2_End, Spectrum_Data )
print(Mean)
```

```
1.0483509826733834
```

In [13]:

```
# (auto)
# Part 3:   Acceleration-Related Part
# a second-order parabola will represent this part:  a_pseudo = A*Tn^2 + B*Tn + C

# define a function called  find_quad_reg_line  that finds A,B,C

def find_quad_reg_line( Init, End, Spectrum_Data ):
    ...
    return x

[A,B,C] = find_quad_reg_line( Part_3_Init, Part_3_End, Spectrum_Data )
print(A,B,C)
```

```
0.5415535067299542 -1.9623165332471026 1.9367699372367655
```

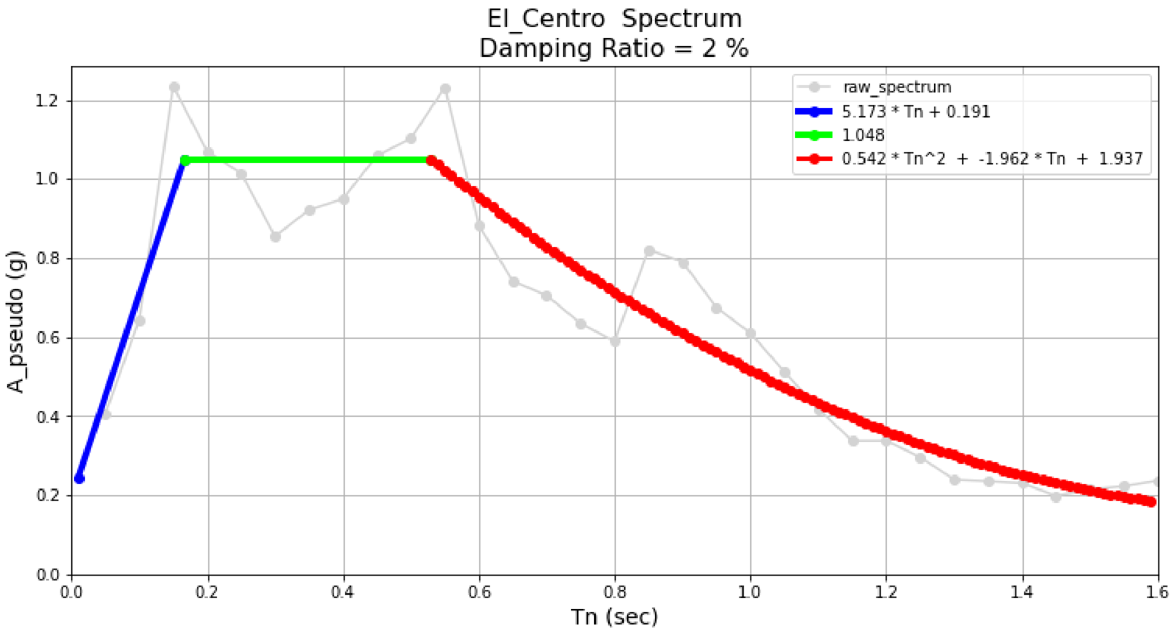
In [14]:

```
# (auto) Finding Intersection of 3 Parts
...
```

## Visualizing Spectrum

In [16]:

```
# (auto) plot
...
```




## making & Saving Final Spectrum DataFrame

In [22]:

```
# (input) Making The Final Spectrum DataFrame
...
print( Final_Spectrum )
```

```
        From Tn (sec)  To Tn (sec)                               a_pseudo (g)
Part_1          0.010        0.166                       5.173 * Tn  +  0.191
Part_2          0.166        0.530                                      1.048
Part_3          0.530        1.600  0.542 * Tn^2  +  -1.962 * Tn  +  1.937
```