Finite Difference Method applied to DML simulation. Part 2b.

ChristianV (Homeswinghome@diyAudio)

04 Feb. 2023

Abstract

Simulation of Distributed Mode Loudspeaker (DML) thanks to the Finite Difference Method (FDM)

Contents

Disclaimer : this paper is written in the context of DIY DML building. This document is not written in the context of any academic or scientific work. Its content is reviewed only by the feedback it can get while posting it in audio DIY forum like diyAudio.

The pdf format of the paper is directly extracted from a python script See Github py2pdf for more information about this method.

1 Introduction

The part 2a is the application of part 1 in a Python script up to the calculation of the system matrix.

In this part 2b, the focus is on the modes (frequency and space shape) and the comparison of the results of the script with some plate situations simulated with Elmer or having exact solutions.

Elmer is a free multi-plateform FEM solver. More details about the simulation files used below.

The tentative to use formulas to extend the simply supported plate case comes from [1], is applied in the function modes() below and compare in the results to the elmer output.

2 Eigenfrequencies

In part 1, the partial derevative equation (PDE) of the vibration of a plate was shown.

For an isotropic homogeneous material, the governing equation is :

$$D(\frac{\partial^4 w}{\partial x^4} + 2\frac{\partial^4 w}{\partial x^2 \partial y^2} + \frac{\partial^4 w}{\partial y^4}) = -q(x,y,t) - \mu\frac{\partial^2 w}{\partial t^2} \tag{1}$$

The PDE in the case of an orthotropic material was also shown (refer to part 1).

The calculation of the eigenfrequencies is a specific case where this equation is solved.

Looking for the natural resonance frequencies, the external load is assumed to be null : $q(x,y,t) = 0$.

The second assumption is to separate the plate deviation w in a product of 2 functions :

$$w(x,y,t) = W(x,y)cos(\omega t)$$

w being in the time a cosine function, its second derivative over the time is also a cosine so the time “disappears” from the equation which becomes :

$$D(\frac{\partial^4 W}{\partial x^4} + 2\frac{\partial^4 W}{\partial x^2 \partial y^2} + \frac{\partial^4 W}{\partial y^4}) = \mu\omega^2 W \tag{2}$$

Which is with the finite difference method (see also part 1) FDM in the form :

$$M_{sys} W = \mu\omega^2 W \tag{3}$$

The modes are the eigenfrequencies of the M_{sys} and the mode shapes are the eigenvectors

This web page explains the concept of eigenvalue and of eigenvector; how numpy can extract them.

3 The testing plate

See figure 1 for the plate representation in the FDM script.

4 The script results

```
The plate is Lx = 0.4 m by Ly = 0.6 m
The plate thichness is h = 0.015 m
The mesh is Nx = 41 cells by Ny = 61 cells
with a grid dx = 10 mm by dy = 10  mm
Young modulus E = 10 MPa, Density rho = 25.0 kg/m³
Bending stiffness B = 3.09 Nm, Areal density µ = 0.375 kg/m²
Case 0
Boundary conditions ['C [north]', 'C [west]', 'C [south]', 'C [east]']
Boundary coeff. North= 1.0  West= 1.0  South= 1.0  East= 1.0
        m         n     elmer        FDM       err%   formula      err%
        1         1        76         76          0        76         0
        1         2       118        118          0       118         0
        2         1       186        187          0       189         1
        1         3       188        189          0       189         0
        2         2       225        226          0       229         1
        1         4       285        286          0       288         1
        2         3       290        292          0       296         2
        3         1       353        354          0       359         1
```

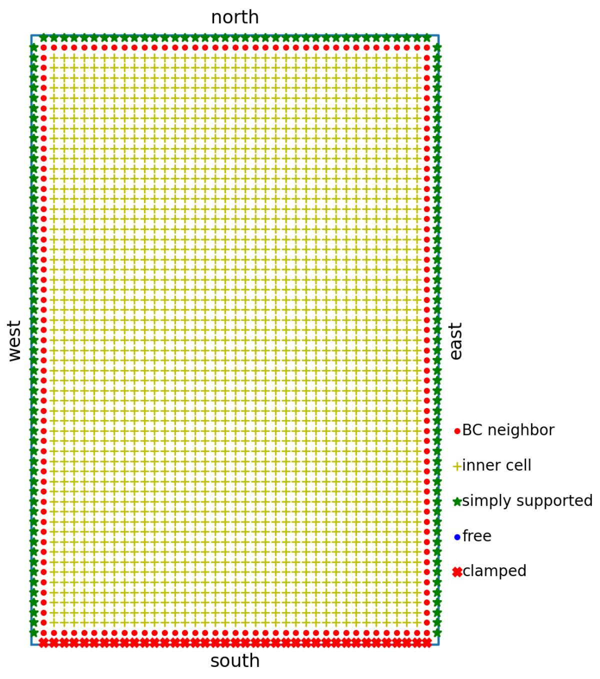


Figure 1: Testing plate

```
          2          4        383        385          0        390          1
          3          2        389        392          0        397          2
          1          5        407        408          0        412          1
          3          3        452        456          0        462          2
          2          5        502        504          0        510          1
          3          4        542        546          0        555          2
          1          6        554        554          0        562          1
          4          1        573        574          0        585          2
          4          2        609        612          0        622          2
          2          6        647        649          0        657          1
          3          5        658        662          0        673          2
          4          3        670        675          0        687          2
          1          7        725        724          0        738          1
          4          4        757        763          0        778          2
          3          6        799        804          0        818          2
          2          7        816        817          0        831          1
          5          1        848        847          0        868          2
Mean error 0.36 %
Case 1
Boundary conditions ['S [north]', 'S [west]', 'S [south]', 'S [east]']
Boundary coeff. North= -1.0  West= -1.0  South= -1.0  East= -1.0
          m          n      elmer        FDM       err%    formula       err%
          1          1         40         40          0         40          0
          1          2         77         78          1         78          1
          2          1        124        125          0        125          0
          1          3        140        140          0        140          0
          2          2        161        162          0        162          0
          2          3        223        225          0        225          0
          1          4        227        227          0        228          0
          3          1        265        265          0        266          0
          3          2        302        302          0        303          0
          2          4        310        312          0        313          0
          1          5        340        339          0        341          0
          3          3        363        364          0        366          0
          2          5        423        423          0        425          0
```

```
         3          4        449        452          0        454          1
         4          1        463        459          0        463          0
         1          6        477        475          0        479          0
         4          2        499        497          0        501          0
         4          6        560        559          0        901         60
         4          6        560        559          0        901         60
         3          5        561        563          0        566          0
         1          7        640        635          0        641          0
         4          4        645        646          0        651          0
         3          6        699        699          0        704          0
         5          1        717        708         -1        717          0
         2          7        723        719          0        726          0
Mean error 0.03 %
Case 2
Boundary conditions ['S [north]', 'S [west]', 'C [south]', 'C [east]']
Boundary coeff. North= -1.0  West= -1.0  South= 1.0  East= 1.0
         m          n      elmer        FDM       err%    formula       err%
         1          1         56         56          0         57          1
         1          2         96         96          0         97          1
         2          1        154        154          0        155          0
         1          3        163        163          0        164          0
         2          2        191        192          0        194          1
         1          4        255        255          0        257          0
         2          3        255        257          0        259          1
         3          1        308        308          0        311          0
         3          2        344        345          0        348          1
         2          4        345        347          0        350          1
         1          5        373        372          0        376          0
         3          3        406        409          0        413          1
         2          5        461        463          0        467          1
         3          4        494        497          0        503          1
         1          6        515        514          0        520          0
         4          1        517        515          0        522          0
         4          2        553        553          0        560          1
         2          6        603        603          0        609          0
         3          5        608        612          0        618          1
         4          3        614        616          0        623          1
         1          7        682        679          0        689          1
         4          4        700        703          0        713          1
         3          6        748        750          0        760          1
         2          7        769        767          0        777          1
         5          1        781        776          0        790          1
Mean error 0.13 %
Case 3
Boundary conditions ['S [north]', 'C [west]', 'S [south]', 'C [east]']
Boundary coeff. North= -1.0  West= 1.0  South= -1.0  East= 1.0
         m          n      elmer        FDM       err%    formula       err%
         1          1         71         71          0         48        -32
         1          2         99        100          1        100          1
         1          3        155        155          0        177         14
         2          1        183        184          0        129        -29
         2          2        213        214          0        176        -17
         1          4        239        239          0        278         16
         2          3        266        268          0        253         -4
         1          5        346        348          0        404         16
         2          4        349        348          0        355          1
         3          1        350        352          0        269        -23
         3          2        381        383          0        313        -17
         3          3        434        438          0        386        -11
         2          5        452        454          0        482          6
         1          6        485        482          0        555         14
```

```
        3         4        512        517          0        487         -4
        4         1        571        573          0        466        -18
        2         6        585        585          0        634          8
        4         2        603        605          0        508        -15
        3         5        617        620          0        615          0
        1         7        646        641          0        731         13
        4         3        656        661          0        579        -11
        4         4        734        740          0        678         -7
        2         7        744        741          0        810          8
        3         6        747        749          0        767          2
        1         8        832        823         -1        932         12
Mean error 0.24 %
Case 4
Boundary conditions ['S [north]', 'S [west]', 'C [south]', 'S [east]']
Boundary coeff. North= -1.0  West= -1.0  South= 1.0  East= -1.0
        m         n      elmer        FDM       err%    formula       err%
        1         1         44         44          0         53         20
        1         2         88         88          0         86         -2
        2         1        126        127          0        153         21
        1         3        157        157          0        146         -7
        2         2        168        169          0        187         11
        2         3        236        238          0        246          4
        1         4        250        251          0        233         -6
        3         1        267        266          0        309         15
        3         2        307        307          0        344         12
        2         4        330        331          0        329          0
        1         5        369        369          0        345         -6
        3         3        373        374          0        403          8
        2         5        448        449          0        439         -2
        4         1        464        460          0        521         12
        3         4        465        467          0        486          4
        4         2        503        501          0        556         10
        1         6        512        511          0        482         -5
        4         3        567        567          0        616          8
        3         5        583        585          0        595          2
        2         6        591        592          0        575         -2
        4         4        658        659          0        699          6
        1         7        679        676          0        645         -5
        5         1        718        708         -1        789          9
        3         6        726        727          0        729          0
        5         2        756        748         -1        825          9
Mean error 0.0 %
```

5 Computational efficiency

The time to get the result depend of course of the computer. For the one used for this test, the results from Elmer were obtains much more faster than with this FDM script which let suppose there are important possibilities of improvements but which need skills in coding.

The possibilities offered by the FDM script like adding lumped elements remain.

The execution time increases as the cube of the number of cells.

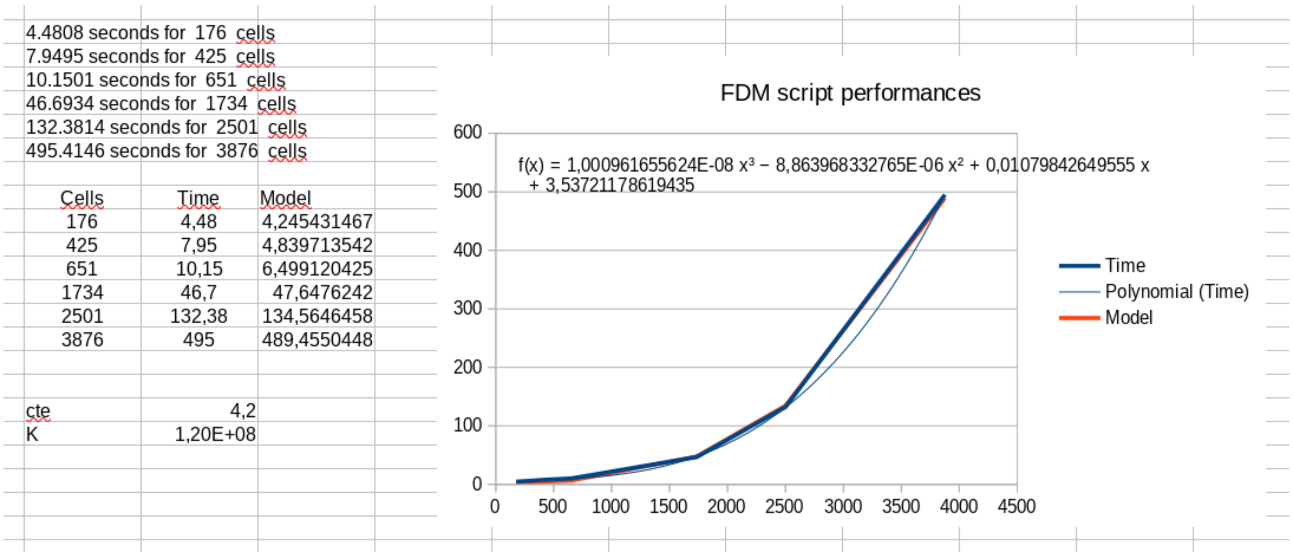
Possible heuristic : $t_{exec} = cte + N_{cells}^3/K$

See figure 2

6 FDM performances

With a 20mm grid, the precision for the 5 test cases are : -1.7, -.9, -1.2, -1.5, -1% over the 16th first modes.

With a 10mm grid, the precision for the 5 test cases are : .36, .21, .18, .36, .13% over the 16th first modes.


4.4808 seconds for 176 cells
7.9495 seconds for 425 cells
10.1501 seconds for 651 cells
46.6934 seconds for 1734 cells
132.3814 seconds for 2501 cells
495.4146 seconds for 3876 cells

Cells	Time	Model
176	4,48	4,245431467
425	7,95	4,839713542
651	10,15	6,499120425
1734	46,7	47,6476242
2501	132,38	134,5646458
3876	495	489,4550448

cte	4,2
K	1,20E+08

Figure 2: FDM performance

The paper shows a 10mm grid over the 25th first modes.

The formula tested gives good results for SSSS and CCCC but not so good in the mix of conditions which was not expected reading the original paper where the limitations are given when the free edge condition is used. As the elmer simulation is working in parallel of the FDM, there is less interest in a reference from a formula. The possible explanation of the deviation even if it is a bug is kept out of the scope of this test run.

7 Mode shapes

The FDM script extracts the mode shape. Below is an example presented in a colorized 2D. Other views are possible with matplotlib.

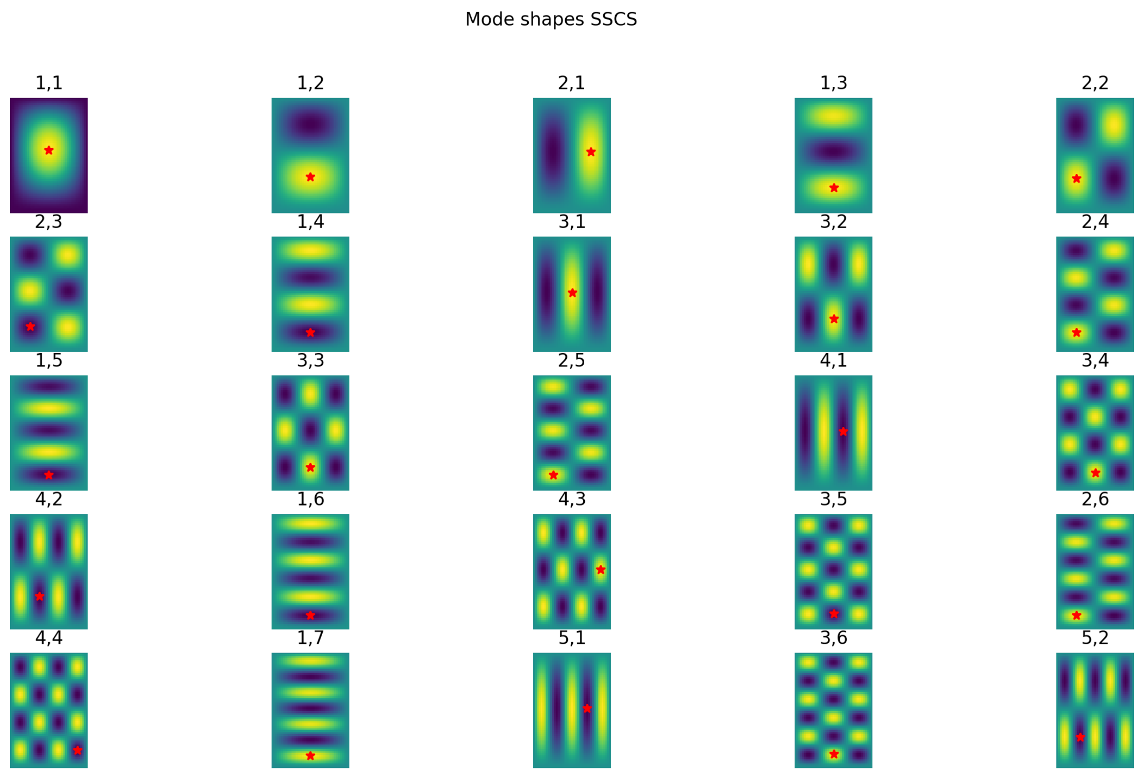


Figure 3: FDM performance

See figure 3

8 Elmer

8.1 Geometry (rectangular_plate.geo)

```
// Inputs
Lx = 0.4; //m
Ly = 0.6; //m
gridsize = Lx / 20;

// All numbering counterclockwise from bottom-left corner
Point(1) = {-Lx/2, -Ly/2, 0, gridsize};
Point(2) = {Lx/2, -Ly/2, 0, gridsize};
Point(3) = {Lx/2, Ly/2, 0, gridsize};
Point(4) = {-Lx/2, Ly/2, 0, gridsize};
Line(1) = {1, 2};                // bottom line
Line(2) = {2, 3};                // right line
Line(3) = {3, 4};                // top line
Line(4) = {4, 1};                // left line
Line Loop(5) = {1, 2, 3, 4};
// the order of lines in Line Loop is used again in surfaceVector[]
Plane Surface(6) = {5};
```

The rectangular_plate.msh is obtain from the gmsh application.

8.2 Mesh

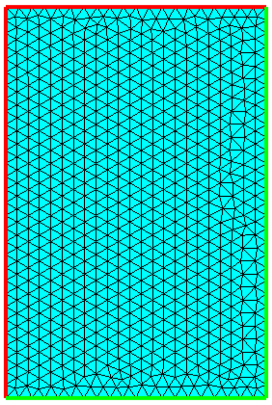


Figure 4: plate mesh

See figure 4

8.3 Solver (case.sif)

```
Header
  CHECK KEYWORDS Warn
  Mesh DB "." "."
  Include Path ""
  Results Directory ""
End

Simulation
  Max Output Level = 5
  Coordinate System = Cartesian
  Coordinate Mapping(3) = 1 2 3
  Simulation Type = Steady state
  Steady State Max Iterations = 1
  Output Intervals = 1
  Timestepping Method = BDF
  BDF Order = 1
  Solver Input File = case.sif
  Post File = case.vtu
End

Constants
  Gravity(4) = 0 -1 0 9.82
  Stefan Boltzmann = 5.67e-08
  Permittivity of Vacuum = 8.8542e-12
  Boltzmann Constant = 1.3807e-23
  Unit Charge = 1.602e-19
End

Body 1
  Target Bodies(1) = 1
  Name = "Body 1"
  Equation = 1
  Material = 1
End

Solver 1
  Equation = Elastic Plates
  Variable = -dofs 3 Deflection
  Eigen System Values = 25
  Eigen Analysis = True
  Eigen System Select = Smallest magnitude
  Procedure = "Smitc" "SmitcSolver"
  Exec Solver = Always
  Stabilize = True
  Bubbles = False
  Lumped Mass Matrix = False
  Optimize Bandwidth = True
  Steady State Convergence Tolerance = 1.0e-5
  Nonlinear System Convergence Tolerance = 1.0e-7
  Nonlinear System Max Iterations = 1
  Nonlinear System Newton After Iterations = 3
  Nonlinear System Newton After Tolerance = 1.0e-3
  Nonlinear System Relaxation Factor = 1
  Linear System Solver = Direct
  Linear System Direct Method = Umfpack
End

Equation 1
  Name = "Plate Equation"
  Active Solvers(1) = 1
End

Material 1
  Name = "EPS"
  Porosity Model = Always saturated
  Tension = 0.0
  Youngs modulus = 10e6
  Thickness = 0.015
  Poisson ratio = 0.
  Density = 25
End

Boundary Condition 1
  Target Boundaries(2) = 3 4
  Name = "fixed"
  Deflection 2 = 0
  Deflection 3 = 0
  Deflection 1 = 0
End

Boundary Condition 2
  Target Boundaries(2) = 1 2
  Name = "Simply supported"
  Deflection 1 = 0
End
```

The Elmer results are stored in the elmer() function below to be displayed along with the script results.

9 Python script code

9.1 Import modules

```python
import os # for py2pdf
import subprocess # for py2pdf with bash with alias
import libpy2pdf as p2p
import matplotlib.pyplot as plt
import numpy as np
from scipy import linalg
# from scipy.sparse.linalg import eigs
import time # for script time performance
import psutil # for script memory alloc check
```

9.2 Function mode()

```python
def modes(B, aa, bb, BB, mumu, mm, nn): # called to check FDM results
    # return the mode frequency of
    # a plate size aa in x by bb in y, material stiffness BB, surface density mumu
    # for the mode mm, nn
    # under the boundary conditions B, B being something like ['SS', 'SS', 'C', 'SS']
    # extension of the Hazell Mitchell approach in R Masso thesis
    k1 = 1/np.pi/2/aa**2*(BB/mumu)**.5 # to transform dimension less lambda in frequency
    r = mm*bb/nn/aa # modal parameter
    dm = 1/(1/r**2 + 2) + 1/60/r # edge correction in m, x direction
    dn = 1/(r**2 + 2) + r/60 # edge correction in n, y direction
    km = 0 # edge coeff in m, x, set to 0 by default (SS condition)
    kn = 0 # edge coeff in n, y, set to 0 by default (SS condition)
    # add 1/2 each time an edge is clamped, either in m or n
    # corrections in one direction m or n : SS => 0, CS or SC => 1/2, CC => 1
    if B[0]=='C': # North
        km = km + 0.5
    if B[1]=='C': # West
        kn = kn + 0.5
    if B[2]=='C': # South
        km = km + 0.5
    if B[3]=='C': # East
        kn = kn + 0.5
    # corrected modes
    mc = mm + km*dm
    nc = nn + kn*dn
    lbda = np.pi**2*(mc**2 + nc**2*(aa/bb)**2) # dimensionless modal frequency parameter
    ff = k1*lbda # frequency Hz to return
    return ff
```

9.3 Functions elmer()

```
def elmer(Bname, rank, rankmax):
    if rank > rankmax:
        rank = rankmax
    # SSSS
    if Bname == 'SSSS':
        mode = np.array([40.4, 77.6, 124.8, 140.1, 161.5, 223.4, 227.7, 265.8, 302.1,
                          310.6, 340.4, 363.3, 423.1, 449.9, 463.5, 477.9, 499.5, 560.1,
                          560.7, 561.8, 640.4, 645.9, 699., 717.8, 723.3])[rank]
    # SSCC
    if Bname == 'SSCC':
        mode = np.array([56.6, 96.4, 154.3, 163.1, 191.8, 255.3, 255.8, 308., 344.4,
                          345.9, 373., 406.6, 461.8, 494.6, 515.4, 517.2, 553., 603.1,
                          608.7, 614.1, 682.3, 700.4, 748.4, 769.4, 781.9])[rank]
    # SCSC
    if Bname == 'SCSC':
        mode = np.array([71., 99.5, 155.4, 183.7, 213.3, 239., 266.8, 346.5, 349.1,
                          350.7, 381.3, 434.8, 452.9, 485., 512.9, 571.9, 585.8, 603.,
                          617., 646.3, 656.9, 734.4, 744.8, 747.2, 832.6])[rank]
    # SSCS
    if Bname == 'SSCS':
        mode = np.array([44.2, 88.1, 126.8, 157.1, 168.5, 236.5, 250.9, 267.1, 307.,
                          330., 369.3, 373.3, 448.6, 464.4, 465.5, 503.1, 512.3, 567.8,
                          583.4, 591.8, 658.5, 679.7, 718.5, 726.2, 756.5])[rank]
    # CCCC
    if Bname == 'CCCC':
        mode = np.array([76.6, 118.2, 186.9, 188.4, 225.2, 285.1, 290.8, 353., 383.4,
                          389.8, 407.1, 452.8, 502.5, 542., 554., 573.6, 609.4, 647.,
                          658., 670.9, 725., 757.8, 799.7, 816.6, 848.2])[rank]
    return mode
```

9.4 Other functions

```
def fillcelltype(mat): # Solver : fill a 2D matrix according to the cell type
    mat[0,:] = 1 # cell type boundary
    mat[-1,:] = 1
    mat[:,0] = 1
    mat[:,-1] = 1
    mat[1,2:-2] = 2 # North, cells before boundary
    mat[-2,2:-2] = 4 #South, cells before boundary
    mat[2:-2,1] = 3 # West, cells before boundary
    mat[2:-2,-2] = 5 # East, cells before boundary
    mat[1,1] = 32 # cell type inner corner
    mat[1,-2] = 25
    mat[-2,1] = 34
    mat[-2,-2] = 45
    return mat

def index(kk,jj,n): # Solver : from 2D plate index to 1D system matrix index
    ii = kk + n*jj
    return ii

def SSSS(aa, bb, BB, mumu, mm, nn): # not called in the script
    # return the mode frequency of
    # a plate size aa in x by bb in y, material stiffness BB, surface density mumu
    # for the mode mm, nn
    # under the boundary conditions ['SS', 'SS', 'SS', 'SS'] simply supported plate
    k1 = np.pi/2/aa**2*(BB/mumu)**.5
    k2 = (aa/bb)**2
    ff = k1*(mm**2 + k2*nn**2)
    return ff

def CCCC(aa, bb, BB, mumu, mm, nn): # not called in the script
    # return the mode frequency of
    # a plate size aa in x by bb in y, material stiffness BB, surface density mumu
    # for the mode mm, nn
    # under the boundary conditions ['C', 'C', 'C', 'C'] clamped plate
    k1 = np.pi/2*(BB/mumu)**.5
    dm = 1/((nn*aa/mm/bb)**2 + 2)
    dn = 1/((mm*bb/nn/nn)**2 + 2)
    ff = k1*(((mm+dm)/aa)**2 + ((nn+dn)/bb)**2)
    return ff

def bound(B): # return the mark style of the edge points according to the BC for plotting
    bstyle = ['g*', 'g*', 'g*', 'g*'] # by default all the edges are simply supported (green)
    for ii in range (0, 4):
        if B[ii]=='C': # clamped edge (red)
            bstyle[ii] = 'rX'
        if B[ii]=='F': # free edge (blue)
            bstyle[ii] = 'b.'
    return bstyle

def boundname(B):
    Bname = B[0] + B[1] + B[2] + B[3]
    return Bname

def boundcoeff(B): # return the value of the stencil AxD according to the BC
    bcoeff = np.ones([4]) # by default all the edges are clamped
    # bcoeff = [-1, -1, -1, -1] # by default all the edges are simply supported
    for ii in range (0, 4):
        if B[ii]=='S': # simply supported
            bcoeff[ii] = -1
        if B[ii]=='F': # free edge
            bcoeff[ii] = 0
    return bcoeff

def plot_plate(n1, n2, bstyle, btext, file2save): # plot the plate
    m = 0.2
    fig = plt.figure(figsize=(15, 8)) # create the figure
    plt.plot([-m,-m, n1+m-1, n1+m-1, -m], [-m, n2+m-1, n2+m-1, -m, -m]) # plate limit
    for i in range(1, n1-1):
        plt.plot(i, 0, bstyle[2])
        plt.plot(i, n2-1, bstyle[0])
    for j in range(1, n2-1):
        plt.plot(0, j, bstyle[1])
        plt.plot(n1-1, j, bstyle[3])
    for i in range(2, n1-2): # inner cells
        for j in range(2, n2-2):
            plt.plot(i, j, '+', color='y')
    for i in range(1, n1-1):
        plt.plot(i, 1, '.',color='r')
        plt.plot(i, n2-2, '.',color='r')
    for j in range(1, n2-1):
        plt.plot(1, j, '.',color='r')
        plt.plot(n1-2, j, '.',color='r')
    plt.text(0.5*(n1-1), n2, btext[0], ha='center', va = 'bottom', size='large')
    plt.text(-1, 0.5*(n2-1), btext[1], rotation='vertical', va = 'center', ha = 'right', size='large')
    plt.text(0.5*(n1-1), -1, btext[2], ha = 'center', va = 'top', size='large')
    plt.text(n1, 0.5*(n2-1), btext[3], rotation='vertical', va = 'center', ha = 'left', size='large')
    # legend
    lstep = n2//10 + 1
    plt.plot(n1+1, lstep, 'rX'); plt.text(n1+1.5, lstep, "clamped", va = 'center')
    plt.plot(n1+1, 1.5*lstep, 'b.'); plt.text(n1+1.5, 1.5*lstep, "free", va = 'center')
    plt.plot(n1+1, 2*lstep, 'g*'); plt.text(n1+1.5, 2*lstep, "simply supported", va = 'center')
    plt.plot(n1+1, 2.5*lstep, 'y+'); plt.text(n1+1.5, 2.5*lstep, "inner cell", va = 'center')
    plt.plot(n1+1, 3*lstep, 'r.'); plt.text(n1+1.5, 3*lstep, "BC neighbor", va = 'center')
    plt.axis('equal')
    plt.axis('off')
    fig.savefig(file2save, bbox_inches="tight", dpi = 200) # this is the key line to store the matplotl
    # plt.title('DML FDM : the plat', fontsize=18) # title
    plt.show()

def memory_usage():
    current_process = psutil.Process()
    memory = current_process.memory_info().rss
    amemory = str(int(memory / (1024 * 1024)))+ "MB"
    return amemory

def freemem():
    stats = psutil.virtual_memory()  # returns a named tuple
    available = int(getattr(stats, 'available')/1024/1024)
    total =  int(getattr(stats, 'total')/1024/1024)
    afreemem = str(available) + "MB free on total " + str(total) + "MB"
    return afreemem

def signchange(a3):
    s3= np.sign(a3)
    #print(s3)
    #s3[s3==0] = -1     # replace zeros with -1
    zero_crossings3 = np.diff(s3)
    zero_crossings3 = np.abs(zero_crossings3)
    #print(zero_crossings3)
    count = np.count_nonzero(zero_crossings3 == 2)+1
    #print(count)
    return count
```

9.5 Prepare py2pdf

```
# lines to be included for py2pdf export
scriptname = os.path.basename(__file__).split('.')[0] # get this script file name without extension
py2pdfdir = p2p.newoutputdir(scriptname)
logfile = py2pdfdir + "/DML_FDM3b.txt" # define the logfile
p2p.clearlog(logfile) # clear the logfile (in case script is ran several times)

print(memory_usage(), "Mb used after import")
print(freemem())
```

9.6 Parameters

```
# Dimensions (in m otherwise mentionned)
Lxmm = 400; Lymm= 600 # use locally mm to avoid rounding problems
h = 0.015 # plate thickness

# Boundary conditions C = clamped, SS = simply supported, F = free (F not available for now)
Direction = ['north', 'west', 'south', 'east']
Bound_case = [['C', 'C', 'C', 'C'],
              ['S', 'S', 'S', 'S'],
              ['S', 'S', 'C', 'C'],
              ['S', 'C', 'S', 'C'],
              ['S', 'S', 'C', 'S']]
```

```
# Material
# do not change the values (used in Elmer for comparison)
Ex = 10e6; Ey = 10e6 # Young modulus x and y direction in Pa
rho = 25. # density in kg/m^3
nu = 0.3 # Poisson ration

B = Ex*h**3/12/(1-nu**2) # bending stiffness
mu = rho*h # areal density
# Mesh
dxmm = 10; dymm = dxmm
Nx = Lxmm//dxmm + 1; Ny = Lymm//dymm + 1 # number of points in the grid
Lx = Lxmm/1000; Ly = Lymm/1000
dx = dxmm/1000; dy= dymm/1000
# Expected results
m = 5; n = m # m by n modes are displayed
case2print = 4 # plot case2print

# Summary
p2p.print_twice(logfile, "The plate is Lx =", Lx,"m by Ly =", Ly, "m")
p2p.print_twice(logfile, "The plate thichness is h =", h,"m")
p2p.print_twice(logfile, "The mesh is Nx =", Nx,"cells by Ny =", Ny, "cells")
p2p.print_twice(logfile, "with a grid dx =", dxmm,"mm by dy =", dymm, " mm")
p2p.print_twice(logfile, "Young modulus E =", int(Ex/1e6), "MPa, Density rho =", rho, "kg/m³")
p2p.print_twice(logfile, "Bending stiffness B =", int(B*100)/100, "Nm, Areal density µ =", mu, "kg/m²")
```

9.7 System matrix filling process

```
tic = time.perf_counter() # optional, start of a timer for time performance check
# Fill the cell type matrix (inner, boundary...)
celltype = np.zeros([Ny, Nx])
celltype = fillcelltype(celltype)

# Fill the system matrix components
Jxrange = range(0,Nx); Kyrange = range(0, Ny)
Imat = np.zeros([Ny,Nx]) # for test
Ax = np.zeros([Ny*Nx,Ny*Nx]) # x direction
AxW = np.zeros([Ny*Nx,Ny*Nx]) # x direction for clamped or simply supported
AxE = np.zeros([Ny*Nx,Ny*Nx]) # x direction for clamped or simply supported
Axy = np.zeros([Ny*Nx,Ny*Nx]) # xy direction
Ay = np.zeros([Ny*Nx,Ny*Nx]) # y direction
AyN = np.zeros([Ny*Nx,Ny*Nx]) # y direction for clamped or simply supported
AyS = np.zeros([Ny*Nx,Ny*Nx]) # y direction for clamped or simply supported
BM = np.zeros([Ny*Nx,Ny*Nx]) # Boundaries
SysMat = np.zeros([Ny*Nx,Ny*Nx]) # System Matrix

for j in Jxrange:
    for k in Kyrange:
        i = k + Ny*j
        Imat[k,j]=i
        cell = celltype[k,j]
        if cell!=1:
            Axy[index(k,j,Ny),index(k,j,Ny)] = 4
            Axy[index(k,j,Ny),index(k-1,j-1,Ny)] = 1
            Axy[index(k,j,Ny),index(k+1,j+1,Ny)] = 1
            Axy[index(k,j,Ny),index(k-1,j+1,Ny)] = 1
            Axy[index(k,j,Ny),index(k+1,j-1,Ny)] = 1
            Axy[index(k,j,Ny),index(k+1,j,Ny)] = -2
            Axy[index(k,j,Ny),index(k-1,j,Ny)] = -2
            Axy[index(k,j,Ny),index(k,j+1,Ny)] = -2
            Axy[index(k,j,Ny),index(k,j-1,Ny)] = -2
        if cell==0:
            Ax[index(k,j,Ny),index(k,j,Ny)] = 6
            Ax[index(k,j,Ny),index(k,j-1,Ny)] = -4
            Ax[index(k,j,Ny),index(k,j+1,Ny)] = -4
            Ax[index(k,j,Ny),index(k,j-2,Ny)] = 1
            Ax[index(k,j,Ny),index(k,j+2,Ny)] = 1
            Ay[index(k,j,Ny),index(k,j,Ny)] = 6
            Ay[index(k,j,Ny),index(k-1,j,Ny)] = -4
            Ay[index(k,j,Ny),index(k+1,j,Ny)] = -4
            Ay[index(k,j,Ny),index(k-2,j,Ny)] = 1
            Ay[index(k,j,Ny),index(k+2,j,Ny)] = 1
        if cell==2 or cell==4:
            Ax[index(k,j,Ny),index(k,j,Ny)] = 6
            Ax[index(k,j,Ny),index(k,j-1,Ny)] = -4
            Ax[index(k,j,Ny),index(k,j+1,Ny)] = -4
            Ax[index(k,j,Ny),index(k,j-2,Ny)] = 1
            Ax[index(k,j,Ny),index(k,j+2,Ny)] = 1
        if cell==3 or cell==5:
            Ay[index(k,j,Ny),index(k,j,Ny)] = 6
            Ay[index(k,j,Ny),index(k-1,j,Ny)] = -4
            Ay[index(k,j,Ny),index(k+1,j,Ny)] = -4
            Ay[index(k,j,Ny),index(k-2,j,Ny)] = 1
            Ay[index(k,j,Ny),index(k+2,j,Ny)] = 1
        if cell==3 or cell==32 or cell==34: #West
            Ax[index(k,j,Ny),index(k,j,Ny)] = 6
            Ax[index(k,j,Ny),index(k,j-1,Ny)] = -4
            Ax[index(k,j,Ny),index(k,j+1,Ny)] = -4
            AxW[index(k,j,Ny),index(k,j,Ny)] = 1
            Ax[index(k,j,Ny),index(k,j+2,Ny)] = 1
        if cell==5 or cell==25 or cell==45: #East
            Ax[index(k,j,Ny),index(k,j,Ny)] = 6
            Ax[index(k,j,Ny),index(k,j-1,Ny)] = -4
            Ax[index(k,j,Ny),index(k,j+1,Ny)] = -4
            Ax[index(k,j,Ny),index(k,j-2,Ny)] = 1
            AxE[index(k,j,Ny),index(k,j,Ny)] = 1
        if cell==2 or cell==32 or cell==25: #North
            Ay[index(k,j,Ny),index(k,j,Ny)] = 6
            Ay[index(k,j,Ny),index(k-1,j,Ny)] = -4
            Ay[index(k,j,Ny),index(k+1,j,Ny)] = -4
            AyN[index(k,j,Ny),index(k,j,Ny)] = 1
            Ay[index(k,j,Ny),index(k+2,j,Ny)] = 1
        if cell==4 or cell==34 or cell==45: #South
            Ay[index(k,j,Ny),index(k,j,Ny)] = 6
            Ay[index(k,j,Ny),index(k-1,j,Ny)] = -4
            Ay[index(k,j,Ny),index(k+1,j,Ny)] = -4
            AyS[index(k,j,Ny),index(k,j,Ny)] = 1
            Ay[index(k,j,Ny),index(k-2,j,Ny)] = 1
        if cell==1:
            BM[index(k,j,Ny),index(k,j,Ny)] = 1
```

9.8 Solver

```
# Case = 4
for Case in range(5):
    Bound_cond = Bound_case[Case]
    BC = boundcoeff(Bound_cond)
    BCname = boundname(Bound_cond)
    plotboundcond =['','','','']
    for i in range(4):
        plotboundcond[i] = Bound_cond[i] + " [" + Direction[i] + "]"
```

~~~
p2p.print_twice(logfile, "Case", Case)
p2p.print_twice(logfile, "Boundary conditions", plotboundcond)
p2p.print_twice(logfile, "Boundary coeff. North=", BC[0], " West=", BC[1], " South=", BC[2], " East
# Show the plate
boundline = bound(Bound_cond) # set the line style for plotting
if Case==case2print:
    plot_plate(Nx, Ny, boundline, Direction, py2pdfdir + "/plate.png") # show the mesh

# Here is the solver
# Compute the system matrix according to the boundary conditions
SysMat = Ax + BC[0]*AyN + BC[1]*AxW + BC[2]*AyS + BC[3]*AxE + Ay + 2*Axy
SysMat = (B*SysMat)/mu/dx**4 + BM

# Compute the eigenfrequencies and the mode shapes
vals,vecs = linalg.eig(SysMat)

toc = time.perf_counter() # to evaluate time performance of the algorithm
# show some performance info
print(memory_usage(), "used")
print(freemem())
print(f"Excecution time = {toc - tic:0.4f} seconds for ", Nx*Ny," cells")

# Clean the results by removing 1 values and complex from the eigenvalues
vals = vals[vals != 1.+0j] # remove the value 1.+0j
vals = np.real(vals)
modefreq = np.zeros([len(vals), 2]) # prepare an array for the eigenfrequencies
modefreq[:,1] = vals**.5/2/np.pi # extract the frequencies
modefreq[:,0] = np.arange(0, len(vals), 1) # give an index
modefreq = modefreq[modefreq[:,1].argsort()] # sort
modefreq = modefreq.astype(int) # keep interger format to reduce the number of digit
```
## Script output printing
```python
results = np.zeros([m*n,8]) # prepare array result
results[:,3] = modefreq[:m*n,1] # eigenfrequency in col 3
results[:,7] = modefreq[:m*n,0] # mode shape number in col 7

for i in range(m*n):
    #results[i,6] = np.mean(vecs[:, int(results[i,7])]) # mode shape mean value
    results[i,2] = elmer(BCname, i, m*n-1) # elmer result in col 2

drivingpoint = np.zeros([Ny, Nx])
if Case == case2print:
    fig = plt.figure(figsize=(15, 8)) # create the figure

for i in range(m*n):
    modeshape = np.real(np.transpose(vecs[:, int(results[i,7])].reshape((Nx,Ny))))
    # modeshape is the 2D matrix showing the plate deviation in the plate coordinates
    ind = np.unravel_index(np.argmax(np.absolute(modeshape), axis=None), modeshape.shape)
    # ind returns the coordinates of the maximum sees in modeshape
    diry = modeshape[:,ind[1]]
    # diry is the 1D slice of modeshape in y direction at the maximum
    dirx = modeshape[ind[0],:]
    # dirx is the 1D slice of modeshape in x direction at the maximum
    # the mode numbers are obtains by checking how lany times dirx, diry signs change
    mx = signchange(dirx)
    ny = signchange(diry)
    results[i,0] = mx
    results[i,1] = ny
    results[i,5] = modes(Bound_cond, Lx, Ly, B, mu, mx, ny) # formula result in 5
    if results[i,7]!=0: # test of driving point similar to Zenker's paper
~~~

```
            mode = np.absolute(modeshape)
            drivingpoint = drivingpoint + (mode/np.max(mode))**0.3
            if i > 0:
                drivingpoint = drivingpoint/np.max(drivingpoint)
        if Case == case2print:
            plt.subplot(m,n,i+1)
            plt.imshow(modeshape)
            plt.plot(ind[1],ind[0], 'r*')
            # plt.title(str(int(results[i,0]))+","+str(int(results[i,1])))
            plt.title(str(mx)+","+str(ny))
            plt.axis('equal')
            plt.axis('off')

    # results[:,6] = 100*results[:,6]/np.max(abs(results[:,6]))
    # results[:,6] = results[:,6].astype(int)
    error = 1000*(results[:,3].astype(int)/results[:,2].astype(int)-1)
    results[:,4] = error.astype(int)/10
    mean_error = int(100*np.average(results[:,4]))/100
    results[:,4] = results[:,4].astype(int)

    error = 1000*(results[:,5].astype(int)/results[:,2].astype(int)-1)
    results[:,6] = error.astype(int)/10
    results[:,6] = results[:,6].astype(int)

    #print(results)
    spacing = 10
    legend = ["m", "n", "elmer", "FDM", "err%", "formula", "err%"]
    aa = ""
    for j in range(7):
        bb = legend[j]
        bb = (spacing - len(bb)-1)*" " + bb + " "
        aa = aa + bb
    p2p.print_twice(logfile, aa)
    for i in range(m*n):
        aa=""
        for j in range(7):
            bb = str(int(results[i, j]))
            bb = (spacing - len(bb)-1)*" " + bb + " "
            aa = aa + bb
        p2p.print_twice(logfile, aa)
    p2p.print_twice(logfile, "Mean error", mean_error, "%")

    if Case == case2print:
        plt.suptitle("Mode shapes " + Bound_cond[0] + Bound_cond[1] + Bound_cond[2] + Bound_cond[3])
        plt.show()
        fig.savefig(py2pdfdir + "/modeshape.png", bbox_inches="tight", dpi = 200) # this is the key lin
'''
ind = np.unravel_index(np.argmax(np.absolute(drivingpoint), axis=None), modeshape.shape)
plt.imshow(drivingpoint)
plt.plot(ind[1],ind[0], 'r*')
plt.show()
'''
```

9.9 py2pdf

```
print("start py2pdf")
cmd = "py2pdf " + scriptname # alias
subprocess.call(['/bin/bash', '-i', '-c', cmd]) # to launch the bash file (alias)
```

References

[1] R. Maso, "Application of boundary edge effect corrections to the vibration of beams and rectangular plates." PhD thesis, Carleton University, 1993. Available: https://curve.carleton.ca/system/files/etd/10817845-5151-47ac-9066-6fc2e360a6f2/etd_pdf/832572c0d80b3466b8f401a713c81266/maso-applicationofboundaryedgeeffectcorrections.pdf

[2] A. W. Leissa, *Vibration of plates*, vol. 160. Scientific and Technical Information Division, National Aeronautics and . . . , 1969. Available: https://ntrs.nasa.gov/api/citations/19700009156/downloads/19700009156.pdf