Programming deficiencies at the start of the master

Joost Ellerbroek

Saullo G. P. Castro

TU Delft

å

Python in the BSc at Aerospace Engineering

TU Delft

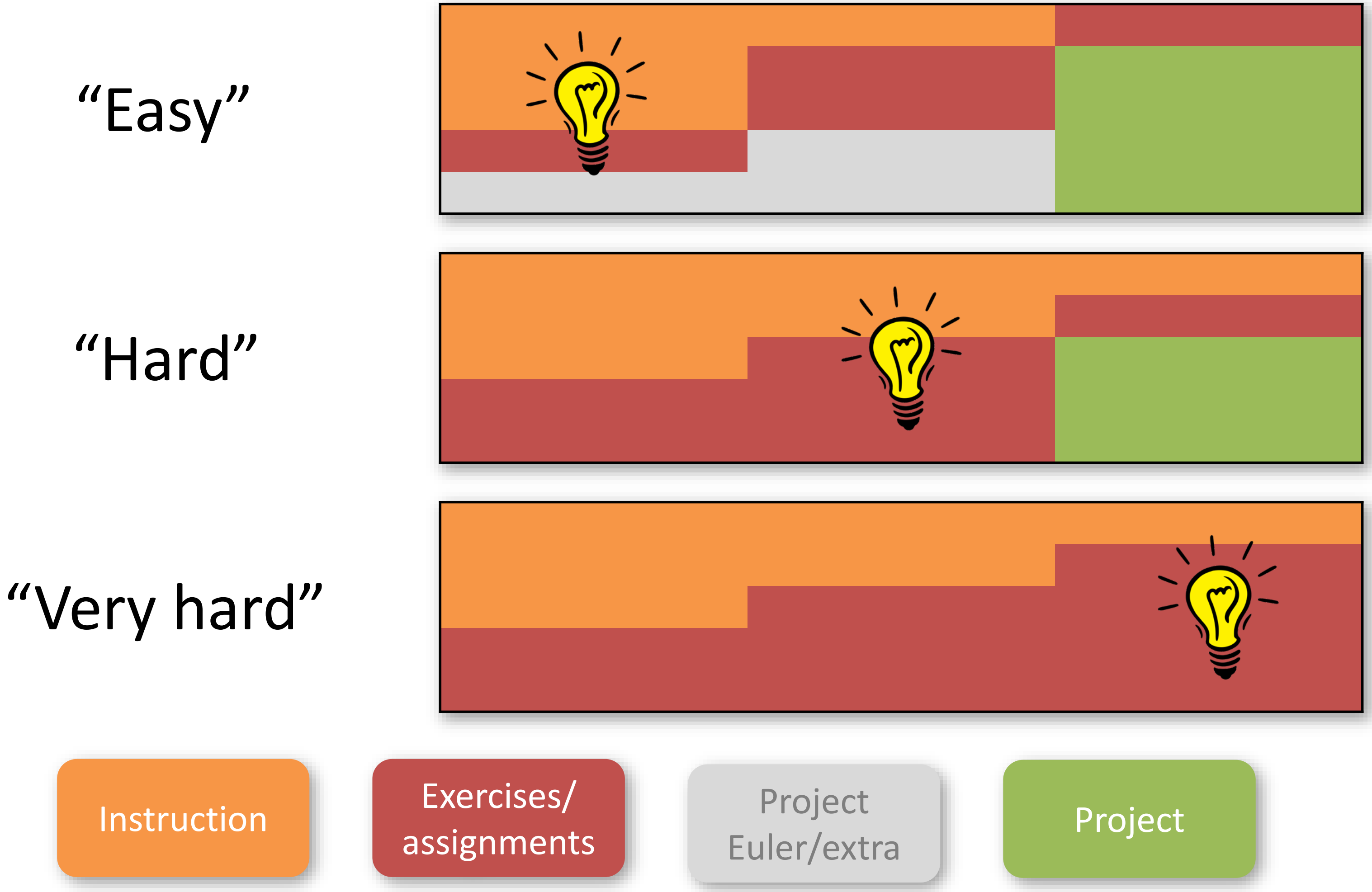
Set-up of course AE1205 Python

- Four parts (incremental)

  - **Theme 1**: General Purpose Programming (Thinking in algorithms)
  - **Theme 2**: Scientific Computing *(NumPy, SciPy, Matplotlib,* num integ*)*
  - **Theme 3**: Visualisation and animation (Graphics, *PyGame,* Animation*)*
  - **Theme 4**: Assignments/Competition/Tests (Simulations/Games, proficiency)

- Instruction, exercises, assignments (0.5 bonus point) under supervision
- Project Euler
- Programming contest (0.5 bonus point)
- ‘Written’ exam with computer part

Course is set up for differentiation



"It was really fun once I got it."

TU Delft

Course is set up fo differentiation

“Easy”

“Hard”

“Very h

Instruction

assignments

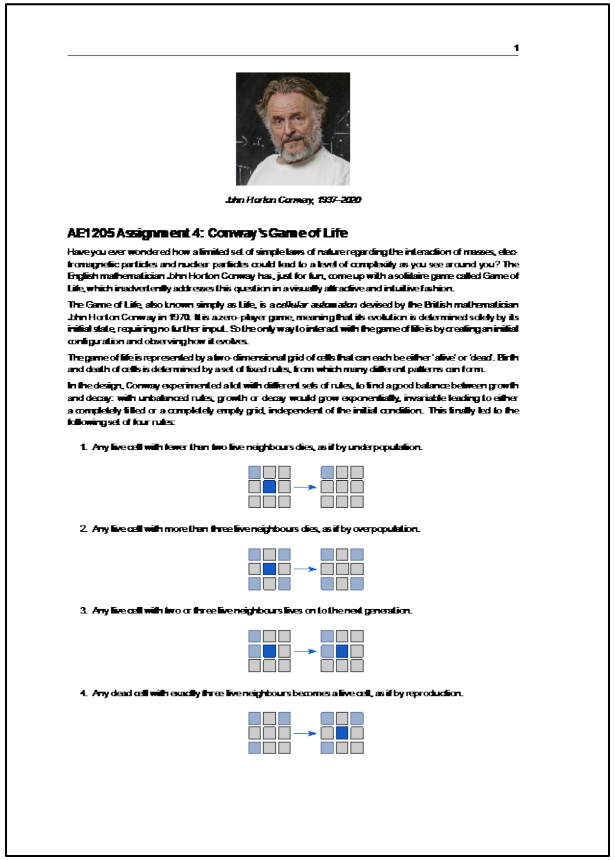
ect

TU Delft

“It was really fun once I got it.”

Assignment examples

- Fascinating mathematical and logical problems
- Games
- Aerospace-themed assignments


1

John Horton Conway, 1937–2020

AE1205 Assignment 4: Conway's Game of Life

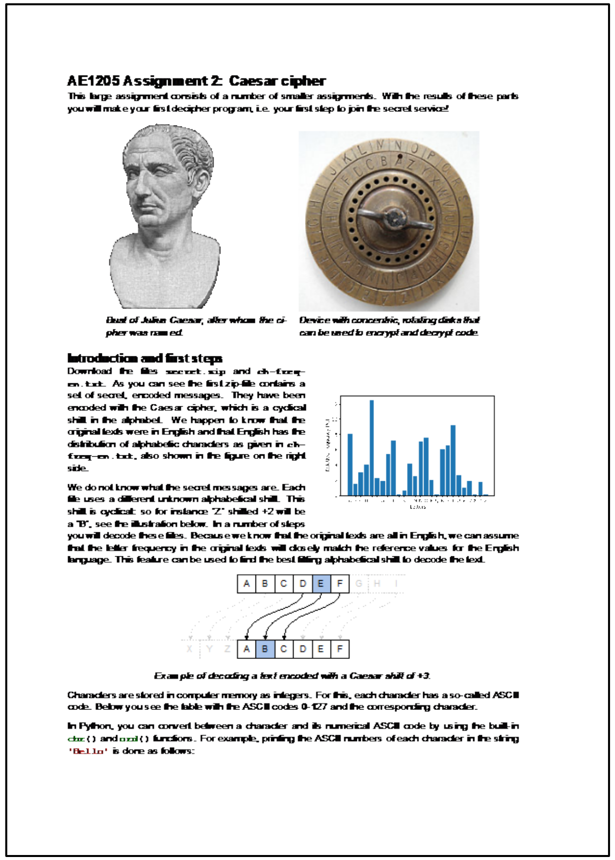
Have you ever wondered how a limited set of simple laws of nature regarding the interaction of masses, electromagnetic particles and nuclear particles could lead to a level of complexity as you see around you? The English mathematician John Horton Conway has, just for fun, come up with a solitaire game called Game of Life, which inadvertently addresses this question in a visually attractive and intuitive fashion.

The Game of Life, also known simply as Life, is a *cellular automaton* devised by the British mathematician John Horton Conway in 1970. It is a zero-player game, meaning that its evolution is determined solely by its initial state, requiring no further input. So the only way to interact with the game of life is by creating an initial configuration and observing how it evolves.

The game of life is represented by a two-dimensional grid of cells that can each be either 'alive' or 'dead'. Birth and death of cells is determined by a set of fixed rules, from which many different patterns can form.

In the design, Conway experimented a lot with different sets of rules, to find a good balance between growth and decay: with unbalanced rules, growth or decay would grow exponentially, invariable leading to either a completely filled or a completely empty grid, independent of the initial condition. This finally led to the following set of four rules:

1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with more than three live neighbours dies, as if by overpopulation.
3. Any live cell with two or three live neighbours lives on to the next generation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.


AE1205 Assignment 2: Caesar cipher

This large assignment consists of a number of smaller assignments. With the results of these parts you will make your first decipher program, i.e. your first step to join the secret service!

Bust of Julius Caesar, after whom the cipher was named.

Device with concentric, rotating disks that can be used to encrypt and decrypt code.

Introduction and first steps

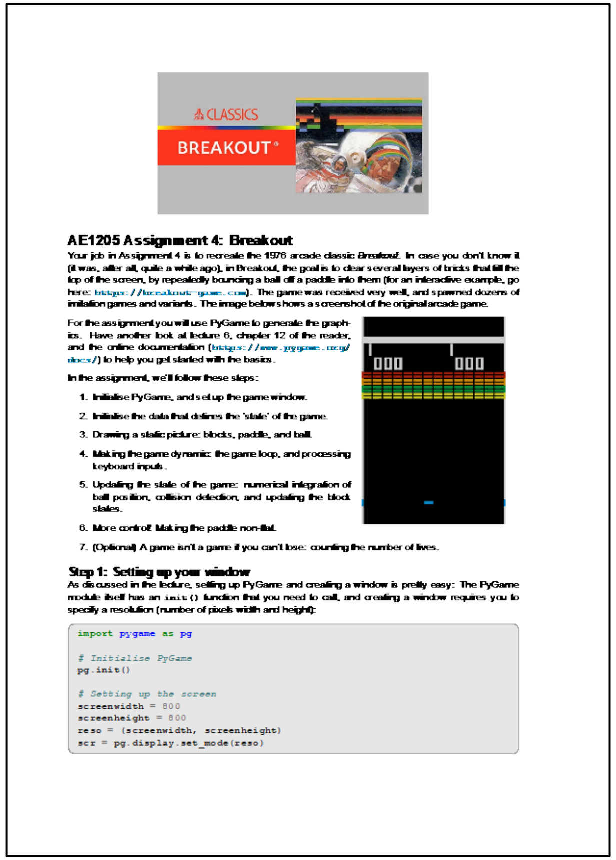
Download the files secret.zip and ch-freq-en.txt. As you can see the first zip-file contains a set of secret, encoded messages. They have been encoded with the Caesar cipher, which is a cyclical shift in the alphabet. We happen to know that the original texts were in English and that English has the distribution of alphabetic characters as given in ch-freq-en.txt, also shown in the figure on the right side.

We do not know what the secret messages are. Each file uses a different unknown alphabetical shift. This shift is cyclical: so for instance 'Z' shifted +2 will be a 'B', see the illustration below. In a number of steps you will decode these files. Because we know that the original texts are all in English, we can assume that the letter frequency in the original texts will closely match the reference values for the English language. This feature can be used to find the best fitting alphabetical shift to decode the text.

Example of decoding a text encoded with a Caesar shift of +3.

Characters are stored in computer memory as integers. For this, each character has a so-called ASCII code. Below you see the table with the ASCII codes 0-127 and the corresponding character.

In Python, you can convert between a character and its numerical ASCII code by using the built-in chr() and ord() functions. For example, printing the ASCII numbers of each character in the string 'Hello' is done as follows:



AE1205 Assignment 4: Breakout

Your job in Assignment 4 is to recreate the 1976 arcade classic *Breakout*. In case you don't know it (it was, after all, quite a while ago), in Breakout, the goal is to clear several layers of bricks that fill the top of the screen, by repeatedly bouncing a ball off a paddle into them (for an interactive example, go here: https://breakout-game.com). The game was received very well, and spawned dozens of imitation games and variants. The image below shows a screenshot of the original arcade game.

For the assignment you will use PyGame to generate the graphics. Have another look at lecture 6, chapter 12 of the reader, and the online documentation (https://www.pygame.org/docs/) to help you get started with the basics.

In the assignment, we'll follow these steps:

1. Initialise PyGame, and set up the game window.
2. Initialise the data that defines the 'state' of the game.
3. Drawing a static picture: blocks, paddle, and ball.
4. Making the game dynamic: the game loop, and processing keyboard inputs.
5. Updating the state of the game: numerical integration of ball position, collision detection, and updating the block states.
6. More control! Making the paddle non-flat.
7. (Optional) A game isn't a game if you can't lose: counting the number of lives.

Step 1: Setting up your window

As discussed in the lecture, setting up PyGame and creating a window is pretty easy: The PyGame module itself has an init() function that you need to call, and creating a window requires you to specify a resolution (number of pixels width and height):

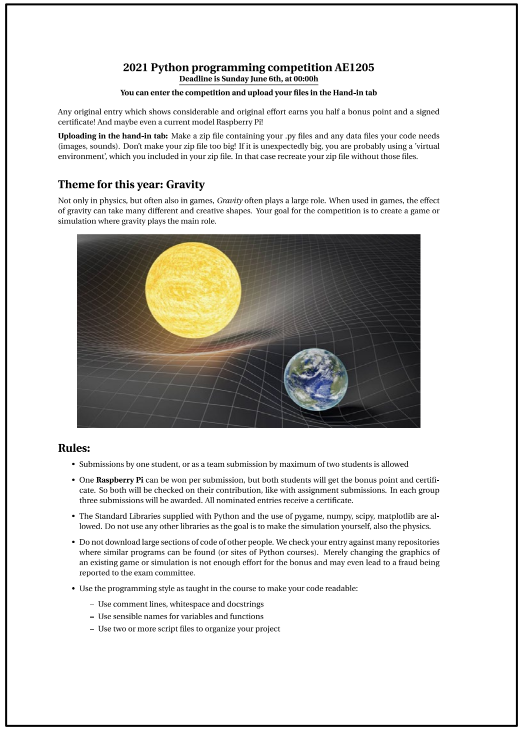
```
import pygame as pg

# Initialise PyGame
pg.init()

# Setting up the screen
screenwidth = 800
screenheight = 800
reso = (screenwidth, screenheight)
scr = pg.display.set_mode(reso)
```

Programming competition

- Themed programming competition. Entries are often games
- Entries in groups of one or two students
- Typically 60-80 entries each year
- Five winning entries earn a Raspberry Pi kit



2021 Python programming competition AE1205
Deadline is Sunday June 6th, at 00:00h
You can enter the competition and upload your files in the Hand-in tab

Any original entry which shows considerable and original effort earns you half a bonus point and a signed certificate! And maybe even a current model Raspberry Pi!

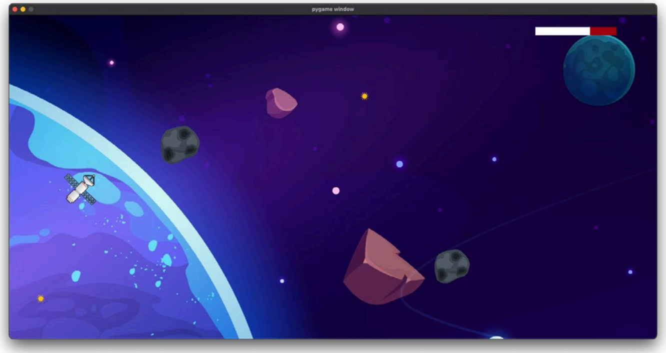
Uploading in the hand-in tab: Make a zip file containing your .py files and any data files your code needs (images, sounds). Don't make your zip file too big! If it is unexpectedly big, you are probably using a 'virtual environment', which you included in your zip file. In that case recreate your zip file without those files.

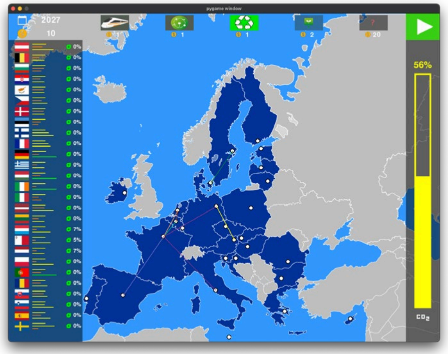
Theme for this year: Gravity

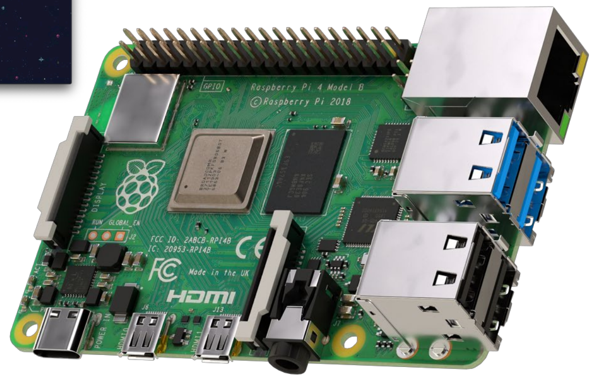
Not only in physics, but often also in games, *Gravity* often plays a large role. When used in games, the effect of gravity can take many different and creative shapes. Your goal for the competition is to create a game or simulation where gravity plays the main role.

Rules:

- Submissions by one student, or as a team submission by maximum of two students is allowed
- One **Raspberry Pi** can be won per submission, but both students will get the bonus point and certificate. So both will be checked on their contribution, like with assignment submissions. In each group three submissions will be awarded. All nominated entries receive a certificate.
- The Standard Libraries supplied with Python and the use of pygame, numpy, scipy, matplotlib are allowed. Do not use any other libraries as the goal is to make the simulation yourself, also the physics.
- Do not download large sections of code of other people. We check your entry against many repositories where similar programs can be found (or sites of Python courses). Merely changing the graphics of an existing game or simulation is not enough effort for the bonus and may even lead to a fraud being reported to the exam committee.
- Use the programming style as taught in the course to make your code readable:
  - Use comment lines, whitespace and docstrings
  - Use sensible names for variables and functions
  - Use two or more script files to organize your project







TU Delft

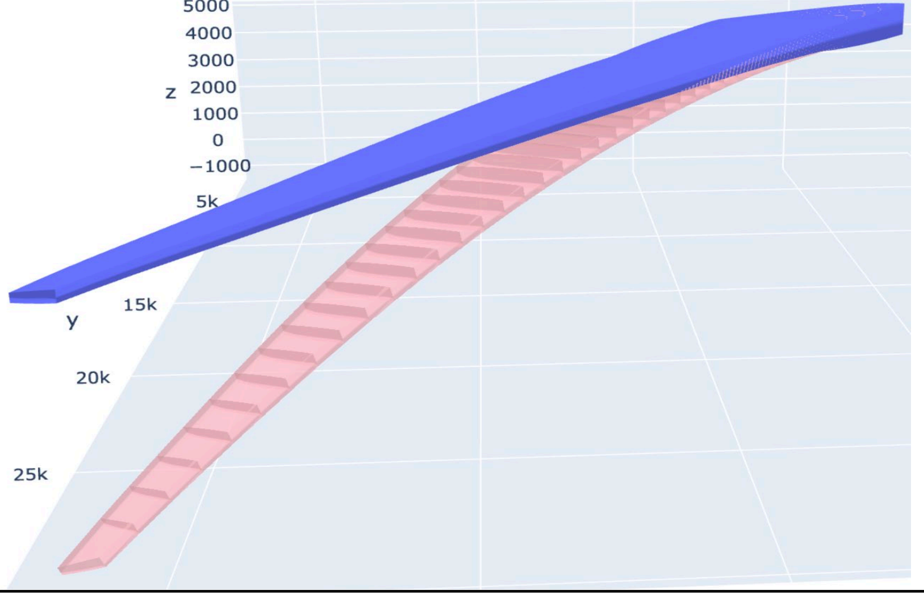
å

Python in an MSc course
at Aerospace Engineering
Aerospace Structures & Materials track

Use case: Structural Dynamics (SASII)

TUDelft

Why is Python used in the SASII course?



Programming used to:

- enable realistic problems with multiple degrees-of-freedom
- create first-hand experience
- give life and practical meaning to the theory

Main Python packages used:

- NumPy and SciPy
- tudaesasII: pure Python-based package developed for this course
- pyfe3d: general-purpose Python/Cython finite element code with common beam and plate elements

TU Delft

NumPy and SciPy in the MSc
Example: Boolean indexing

```
# known DOFs    --> bk
# unknown DOFs --> bu
```

Applying boundary conditions

```
# applying BC
# defining known DOFs
bk = np.zeros(N, dtype=bool)
at_clamp = np.isclose(x, 0.)
bk[0::DOF][at_clamp] = True
bk[1::DOF][at_clamp] = True
bk[2::DOF][at_clamp] = True
bk[3::DOF][at_clamp] = True
bk[4::DOF][at_clamp] = True
bk[5::DOF][at_clamp] = True
```

Matrix partitioning

$$\boldsymbol{K}\,\boldsymbol{u} = \boldsymbol{F}$$

$$\begin{bmatrix} \boldsymbol{K}_{uu} & \boldsymbol{K}_{uk} \\ \boldsymbol{K}_{ku} & \boldsymbol{K}_{kk} \end{bmatrix} \begin{Bmatrix} \boldsymbol{u}_u \\ \boldsymbol{u}_k \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{F}_k \\ \boldsymbol{F}_u \end{Bmatrix}$$

$$\boldsymbol{K}_{uu}\boldsymbol{u}_u = \boldsymbol{F}_k - \boldsymbol{K}_{uk}\boldsymbol{u}_k$$

```
uu = u[bu]
uk = u[bk]
Kuu = K[bu, :][:, bu]
Kuk = K[bu, :][:, bk]
```

Applying forces

```
force = np.zeros(N)
at_point = (np.isclose(x, 0.)
            & np.isclose(y, 3.))
force[0::DOF][at_point] = fx
force[1::DOF][at_point] = fy
force[2::DOF][at_point] = fz
```

TUDelft

NumPy and SciPy in the MSc
Example: Quadratic eigenvalue problem, stacking arrays

Quadratic eigenvalue problem: $-\boldsymbol{M}\omega^2\boldsymbol{U} + \boldsymbol{C}i\omega\boldsymbol{U} + \boldsymbol{K}\boldsymbol{U} = \boldsymbol{0}$

Assuming the following state space transformation:

$\boldsymbol{\varphi} = \omega\boldsymbol{U}$ (Eq. 1)

It comes that:

$-\boldsymbol{M}\omega\boldsymbol{\varphi} + \boldsymbol{C}i\boldsymbol{\varphi} + \boldsymbol{K}\boldsymbol{U} = \boldsymbol{0}$ (Eq. 2)

Combining Eqs. 1 and 2:

$$\omega \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{M} \end{bmatrix} \begin{Bmatrix} \boldsymbol{U} \\ \boldsymbol{\varphi} \end{Bmatrix} + \begin{bmatrix} 0 & -\boldsymbol{I} \\ \boldsymbol{K} & i\boldsymbol{C} \end{bmatrix} \begin{Bmatrix} \boldsymbol{U} \\ \boldsymbol{\varphi} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{Bmatrix}$$

Expanded linear eigenvalue problem

```
from numpy import row_stack, column_stack

I = np.identity(N)
ZERO = np.zeros_like(M)

A = row_stack((column_stack((ZERO,   -I)),
               column_stack((   K, 1j*C))))

B = row_stack((column_stack((   I, ZERO)),
               column_stack((ZERO,   -M))))
```

TU Delft

NumPy and SciPy in the MSc
Example: Facilitating the implementation of theory

The expanded linear, generalized eigenvalue problem:

$$\left(\omega\begin{bmatrix}\boldsymbol{I} & \boldsymbol{0}\\ \boldsymbol{0} & -\boldsymbol{M}\end{bmatrix}+\begin{bmatrix}0 & -\boldsymbol{I}\\ \boldsymbol{K} & i\boldsymbol{C}\end{bmatrix}\right)\begin{Bmatrix}\boldsymbol{U}\\ \boldsymbol{\varphi}\end{Bmatrix}=\begin{Bmatrix}\boldsymbol{0}\\ \boldsymbol{0}\end{Bmatrix}$$

Extra step below is added in the lecture to facilitate their experience implementing the theory

Rearranging for "scipy.linalg.eig":

$$\begin{bmatrix}0 & -\boldsymbol{I}\\ \boldsymbol{K} & i\boldsymbol{C}\end{bmatrix}\begin{Bmatrix}\boldsymbol{U}\\ \boldsymbol{\varphi}\end{Bmatrix}=-\omega\begin{bmatrix}\boldsymbol{I} & \boldsymbol{0}\\ \boldsymbol{0} & -\boldsymbol{M}\end{bmatrix}\begin{Bmatrix}\boldsymbol{U}\\ \boldsymbol{\varphi}\end{Bmatrix}$$

a @ vr = w * b @ vr

scipy.linalg.eig

`scipy.linalg.eig(a, b=None, left=False, right=True, overwrite_a=False, overwrite_b=False, check_finite=True, homogeneous_eigvals=False)` [source]

Solve an ordinary or generalized eigenvalue problem of a square matrix.

Find eigenvalues w and right or left eigenvectors of a general matrix:

```
a   vr[:,i] = w[i]        b   vr[:,i]
a.H vl[:,i] = w[i].conj() b.H vl[:,i]
```

where `.H` is the Hermitian conjugation.

å

Discussion Points

Programming deficiencies at the start of the master
Discussion Points

Students without TU Delft BSc

Transition from MATLAB to Python

More MSc elective disciplines focused on Python?

Thank you for your attention

Joost
Saullo

TU Delft