A code-centric flipped classroom course on analytical mechanics

Víctor A. Bettachini[1,2][0000−0001−7485−8884],
Mariano A. Real[1,3][0000−0003−3022−7516], and
Edgardo Palazzo[4][0009−0006−8783−8261]

[1] Universidad Nacional de La Matanza - UNLAM, Buenos Aires, Argentina
vbettachini@unlam.edu.ar
https://ingenieria.unlam.edu.ar/
[2] Instituto Geográfico Nacional - IGN, Buenos Aires, Argentina
[3] Instituto Nacional de Tecnología Industrial - INTI, Buenos Aires, Argentina
[4] Universidad Tecnológica Nacional - UTN, Buenos Aires, Argentina

Abstract. This paper outlines the methodologies, tools and organisation of mechanical engineering undergraduate course that employs code as the sole means to perform all its calculations. Modeling simple mechanical devices as rigid bodies and employing the analytical mechanics Euler-Lagrange equation the code is able to simulate the dynamics and mechanical efforts of these systems. No prior programming skills are required, provided code that solves example problems is modified by students to address new ones. Jupyter Notebooks running on Google Colaboratory provides an unified platform for lesson's material embedding code among Markdown formatted text and LaTeX mathematical expressions.
Originally conducted online during the SARS-CoV-2 pandemic, the course has transitioned to in-person sessions and a flipped classroom approach. Exercises sets turn-in is mandatory but keeping overall homework at the minimum to free-up student's time for reading of lessons before weekly face-to-face meetings that provide opportunities for clarification and progress discussions. A learning management system is used to keep track of each student progress and to provide asynchronical consultations.
The course culminates in a challenging final problem: analysing forces and torques on a simplified industrial robotic arm. Beyond technical skills, students enhance presentation and synthesis abilities, defending their solutions through concise oral presentations to teachers.

Keywords: Code · Flipped Classroom · Mechanical Engineering

1 Introduction

1.1 Valuing teachers' and students' time

The psychologist J.C.R. Licklider stated in the 1950s that 85% of his "thinking" time was in fact used in information related tasks such as finding it or plotting graphs; an observation that set him to become an advocate of interactive

computing and push the creation of ARPANET the antecessor to today's INTERNET [1]. In the 21st century, it is not unusual to find university courses following a routine where professors transcribe lessons by heart to blackboards or repeat slide presentations. Students then transcribe this information once again into their paper or digital notebooks, a methodology largely unaltered since its introduction in the 19th century. The situation is aggravated in science and engineering courses, where the same calculations and drawings are made repeatedly, not only in the classroom but also when exercises based on the subject of the class require performing very similar tasks to those done by the teachers.

This excercise on repeated transcription can be seen as a waste time due to the ubiquitous availability of technologies product of the information revolution during the last half of the 20th century propelled by the efforts of Licklider and colleagues. Trying to avoid this our university course harness the power of interactive computing for in-class lessons as well as for solving problem sets using the same digital platform. All material that the teaching staff want to share to students is in a digital format that not only contains all theory on the subject but code that solves problems related to it. A document containing executable code for each lesson is stored in a public repository from where the students generate their own copy. The copy's code is fully modifiable, and students are required to do so in order to solve the problem sets proposed by the teaching staff in a procedure akin to recycling. The matematician S. Papert, a pioneer of employing computers in a constructivist framework for education, onec stated that learning takes place when the learner takes charge of the operation [2]. Following this advise, the problem-sets in our course are built to lead the student gradually to becoming autonomous by reusing not the code provided by the teaching staff but his own code to address increasingly more complex problems.

In the current century, the fact that university courses do not ubiquitously employ coding to save time an effort of everyone in our classrooms is a situation alike the rejection of pocket calculators in 1970s argumenting that abilities on arithmetics would be jeopardised [3]. Nowadays after students learnt arithmetics at elementary school they can freely use pocket calculators. The same should be the case after they passed their calculus and algebra courses, they should be able to employ freely their computational counterparts in all their following courses. Not only would the focus of their effort be diverted from those automatable calculations, but they could also tackle with the tools of numerical analysis problems beyond what can be solved on a blackboard or on paper.

1.2 Numerical analysis and computers

The stored-program digital computer was conceived as a more flexible tool for solving numerical analysis problems than their electro-mechanical predecessors that had to be reconfigured to address different problems [4]. Originally funded by and reserved for military research programs at the forties, it was towards the end of that decade that civilians at universities started to exploit them for other purposes [5]. So, PhD candidates were among the first students to use these mainframe computers in the so called punch card and batch processing era that

stretched into the 1970s. By then, higher-level languages and lower-cost minicomputers had opened up access for undergraduate students [6]. Time-sharing based live interaction with computers became an established practise, allowing to serve several students at once to deploy computer-based instruction [7]. These computer-assisted instruction systems provided tailor-made lessons created by the teaching staff, alongside questionnaires and other interactive feedback mechanisms to evaluate students' understanding of a variety of subjects. This approach avoided pressuring students into programming [8]. A contemporary push on the opposite direction directed to school age students sought to foster mathematical and geometrical abilities, creating languages aimed for educational use based on ideas of the constructivist philosophy [9], such as LOGO [10] or Smalltalk [11].

Coming full circle to the origin of digital computers, exploitation of numerical analysis by undergraduate students performed with languages such as Python, that inherits the simplicity of LOGO and object orientation of Smalltalk, could provide them a useful calculation tool at almost any science or engineering course. As digital pocket calculators freed students of repetitive arithmetic tasks, allowing to explore otherwise avoided problems, current programming languages provide symbolic and numerical analysis as well as plotting capabilities that allow to visualise and explore the mathematical solutions of any conceivable problem presented in university courses.

2 Course cornerstones: code reuse and flipped classroom

2.1 Analytical mechanics for mechanical engineers

Numerical analysis courses are ubiquitous in the mechanical engineering degrees programme's firsts years in the Argentine Republic. There are usually preceded by a course on programming fundamentals, allowing to focus the efforts of the teaching staff of the former to rely on what was taught in the later. Unfortunately it is not unusual that in later courses the problems presented to students are limited to those having analytical solutions of the models, so the knowledge and practice obtained not only on the subject of numerical analysis, but also of programming as a tool, are seldom exploited at full at later courses. In an effort to revert this trend, our course on analytical mechanics precedes those later ones and is placed immediately after those where these abilities were acquired. As shown by the roadmap presented in Figure 1 for the degree on mechanical engineering at La Matanza University (UNLaM). One reason to do so, is to put mechanical engineering students to work on the subject matter of mechanical devices quickly, capitalising what they learnt in subjects devoted to algebra, mathematical analysis, numerical calculus, and Newtonian mechanics. Another reason is to assure the teaching staff of the subjects on the mechanics of stability, devices or fluids, that the students they receive are experienced on the tools of analytical mechanics but also on computer-based numerical analysis. In this way the course acts as a link between the first specific mechanical courses and the basic ones.

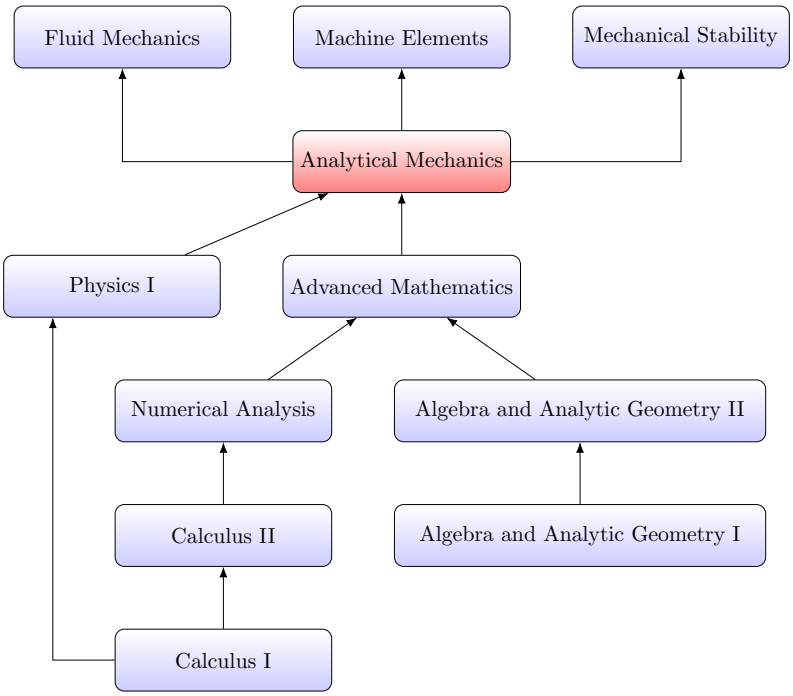

Fig. 1. Preceded by subjects of algebra, calculus, and physics, analytical mechanics is the first in which such knowledge is applied to mechanical engineering. It also serves as a foundation for subsequent specialised mechanical engineering courses.

But, why is the physics subject of analytical mechanics, also known as classical mechanics, a worthwhile tool for mechanical engineers? The mere statement that it confers the power to model the physics of simple mechanical systems does make it sound like a redux of Newtonian mechanics. And it is in putting in contraposition with that another philosophy of how to understand physical mechanics that the value of the analytical approach stands appart. Instead of an artisanal analysis of vector forces, the analytical approach is absolutely systematic. Creativity is only required to produce a physical model out of the observation of the actual system, then it follows the same identical steps, independently of the system under study, to produce results, making it highly automatable for machines or people alike [12]. The creation of a physical model from observations involves simplifications, the student must determine what is important and what is not to capture the main features of the physical system.

Training this ability of students is the most demanding challenge to the teaching staff of this course. Once a physical model is constructed the analytical mechanics steps based on the Euler-Lagrange formalism, reviewed later in this work, produce a set of differential equations. These can describe the dynamics

or the mechanical efforts the actual system is subjected to, and next one needs only to solve them.

2.2 Limits on analytical mechanics usefulness

Traditionally, the physical systems studied in analytical mechanics courses are relatively simple in order to limit the time and/or difficulty of mathematical analysis and/or algebra calculations required by the steps commented in the previous paragraph. But this extreme simplification leads to a later noticeable jump in the complexity required to model mechanical devices, fluid dynamics, or rigid structures, in the courses, shown in the figure 1, coming up next where the student must apply this tool. The limitation to complexity of the examples is imposed by what can be calculated on a blackboard or presentation slides. The same goes for the length and complexity of the problems that students can be asked to solve.

Computer algebra systems, that will soon turn 60 [13], have of course certain limitations on what they can compute, but these are well beyond of the most complex problem an undergraduate student of engineering could meddle in. Libraries for symbolic algebra and calculus are available for all major general programming languages, both libraries and languages being published as free software. So the only reason that limits the reach of what students can calculate is a culture that keeps the blackboard, slides or paper as the centre piece of the work at classrooms in place of computers. Although numerical analysis is reportedly used to solve problem sets [14, 15], it is rarely used by the teaching staff during classes. But if the application of numerical analysis during classes is rare, the use of symbolic computer algebra is even rarer. The use of computer algebra and calculus solutions re-focus the students attention towards the subject matter of the current course.

2.3 Programming vs. code reuse

Having fluency in a computer language's code in order to be able to comprehend what was written by another person, modify it or even write its own is of course an ability required for programming, but by itself fells short of the analytical skills and knowledge of algorithms and computer architecture required by professionals in the field. For the matters of our course we state a strict distinticion between "programming" and the less demanding "coding" to the students at the first meeting they have with the teaching staff. Altough at UNLaM they had a previous course of introduction to programming based on `C++`, so they already posess ability on programming algorithms, that in any way is something they need to apply in the course. In our experience, even for students that allegedly "have forgotten" what they learnt on programming due to lack of use found that Python's concise and simple syntax [16] makes it relatively accesible to grasp it to the level required to understand and modify the code provided by the teaching staff.

For each class the teaching staff makes available examples of codes that perform all the procedures required to model a certain characteristic of a mechanical system. Related exercises are provided in a problem set that can be solved by making small modifications to the code. This later ability to adapt the code by slight changes is nothing more than code reuse, a common practice in the computing industry since its inception.

A good deal of courses appeared during the last decade that take advantage of tools such as the ones used in our course, to produce graphical output to clarify the results of numerical simulations in diverse areas such as computational fluid dynamics [17] or chemical thermodynamics [18]. There are also teaching materials, once again relying on these tools, that are published in a book alike format in the fields of chemistry [19]. But these cited examples used their published material containing code to support the lessons, not to be the basis of their problems sets. There are of course others that do exactly that, even providing example problems with the code required to solve them embedded into them, such one on stucture modelling for civil engineers [20], but these do not rely on the media to be the basis for their lessons. We understand that not using the same digital format with embedded code both at lessons and to solve the problem sets results in the exercise of transcription discussed previously. To the best of our knowledge there is no course, besides ours, that integrates analytical mechanics with numerical analysis in lessons to provide code able to calculate problems of dynamics and efforts in rigid body mechanical systems using the same digital media.

2.4 Flipped classroom

The course first edition was coincidental with the outbreak of the SARS-CoV-2 pandemic. Lockdowns forced us to implement emergency remote lessons based on digital teleconference software solutions. The teaching staff having previously used Jupyter Notebooks at laboratory courses wrote the lessons for the first weeks taking advantage of Python code to perform the required calculations and also plotting its results. While the first synchronical meeting was conducted, it was verified that each one of the students was sitted in front of a computer. The opportunity to force students to use code to solve the exercises set was quickly grasped. After an initial adaptation period, the students recognized the virtues of this methodology. Even assessments were more enriching than in a conventional course as it reached the complexity of simulating industrial-like mechanical systems. During the four semesters when lessons were remote, adoption of asynchronous consultations grew. As the lesson material was published in advance to the meetings, the teaching staff started to suggest that it be read before it in order to discuss it rather than present it. This then led to suggest that some excercises be solved also in advance. Gradually this led to free time at the meetings thus allowing the students to work on the excercises while they could make synchronical consultations on them to the staff. By the time in person meetings at classrooms began, the staff decided to put to test a radical proposal: avoid

giving lectures altogheter, switching to the *flipped classroom* methodology [21, 22] schematised in figure 2.

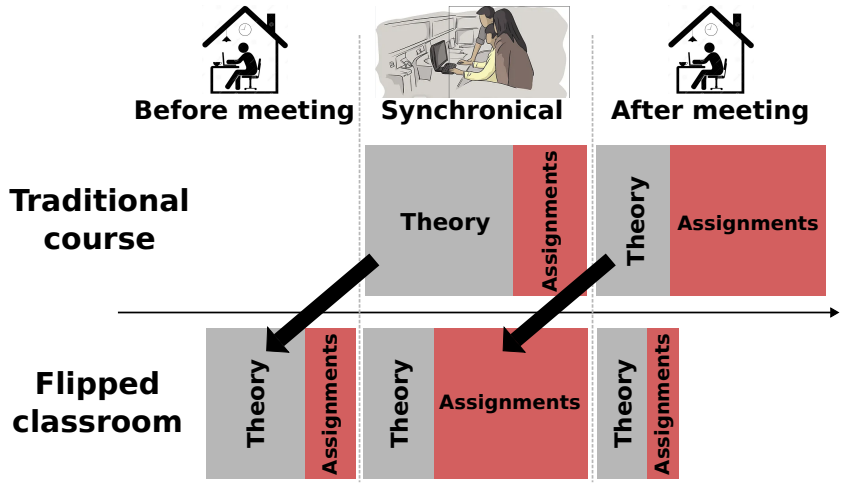

Fig. 2. This schematizes how the student allocates the same amount of time in our flipped classroom course compared to a traditional one. There is a synchronic meeting for each thematic block. Reading theory and starting the work on their exercise set is mandatory before the meeting.

The current course syllabus is strictly divided into weekly distinct thematic blocks with a single synchronic meeting devoted to it. No lecturing occurs during the meeting, thus all the staff's available time is reserved for one to one consultations on the excercise sets. If a single subject is perceived to need some review, a quick lecture on the particular is improvised based on the material at the repository. Students are required to read in anticipation the respective material on theory published beforehand in the course repository as Jupyter Notebooks, alongside videos with discussions on important topics. They also must begin working on the exercises before the meeting where they seek clarification on both, the theory and difficulties found while trying to solve the practice problems. As they finish up them in a later date and usually that requires some review on the theory, an asynchronic communication channel is available where the teaching staff can answer their questions on both matters.

3 Course tools

3.1 Analytical mechanics and physical modeling

The course employs the most traditional principles of rational mechanics, as outlined in standard reference literature [23]. Modeling is understood as a series

of procedures with which a simplified scheme of physics is constructed based on a semi-quantitative evaluation of the forces and fields that act on the system as well as the constraints that limit its degrees of freedom. With such information, some of these are prioritized and others are discarded to arrive at the aforementioned scheme. Having such a model allows:

- to choose generalized coordinates to describe the relevant degrees of freedom,
- to write mathematical relationships between them that account for constraints,
- to describe generalized forces that are not the effect of fields (gravitational, electromagnetic, etc.),
- and to describe the potential and kinetic energy of the system as a whole.

After performing the above, the Euler-Lagrange formalism is demonstrated and put into practice in the course to obtain a set of differential equations that describe the dynamics of the system and/or the mechanical stresses that each of its components must withstand over time.

3.2 Python, Sympy, Numpy, Scipy and Matplotlib

The Python programming language is by default devoid of scientific and engineering calculation capabilities. This is a design decision to require that such functionalities be added by specialized libraries, whose development is carried out by users who apply them in various fields of science and technology. A conscious decision was made to use the most standard and lesser in number to address the distinct requirements of the course:

- Capability for symbolic algebra and calculus as provided by the Sympy library. Its *Mechanics* module is particularly useful to generate equations for the dynamics of rigid body systems with multiple degrees of freedom and in various reference frames [24].
- Differential equation systems are solved by numerical methods supported by functions for the manipulation of algebraic elements of the Numpy library [25] and the numerical optimization and integration algorithms of Scipy [26].
- Engineering analysis of numerical results is usually interpreted with graphical representations. This capability is provided by functions of the Matplotlib library [27].

3.3 Jupyter Notebooks on Google Colaboratory

The environment used in the course to run code is the web-based application called JupyterLab, whose document format is the Jupyter Notebook [28]. As shown by the figure 3 these documents are divided in an arbitrary number of independent sections called cells that can contain executable code of some programming language, being Python one of the possible ones, or either annotations.

Annotations cells are written in the Markdown markup language [29] that allows to embed text and mathematical expressions in LaTeX format interspersed

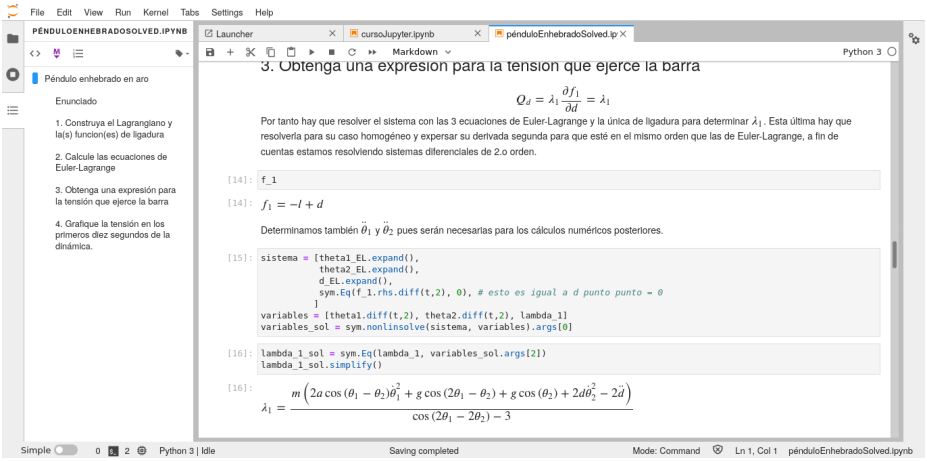

Fig. 3. A Jupyter notebook is a set of cells containing either exectuable or Markdown code. The former can contain text, mathematical expressions or multimedia content. The latter are lines of code in various programming languages. Interspersing titles in Markdown cells generates the index (on the left) that facilitates location within the document.

as well as multimedia content such as web links, images, video and sound players. The use of LaTeX syntax for the mathematical symbology provides a standardised and clear notation under the guidelines of the American Mathematical Society [13]. This is a further advantage of the Jupyter Notebook as the media for lessons over the blackboard or slides prepared with software such as Microsoft Powerpoint.

Students are not required to install any special software on their computer to work with Jupyter Notebooks, only a standard web browser is neeeded to use online services that run Jupyter Notebooks. This can be an installation of JupyterHub from the Jupyter Project on university-owned servers or commercial clouds, or alternatively one of the services that offer even free alternatives such as CoCalc, IBM Watson or Google Colaboratory[5]. The later, colloquialy known as Colab, is currently in use by the course as it conveniently presents the ability to run notebooks hosted by the online service GitHub. A modification in the URL that points to the notebook leads a web browser to open it in Colab. [18, 30]. Which can be worked on concurrently by students and teachers. Students are encouraged to collectively solve exercises using this functionality. As shown by figure 4, comments can also be included, an useful feature to correct exercises since the location of errors in the code can be indicated. The teaching staff can also include inline helping code and new sections into the students notebook.

[5] The authors do no endorse any of the products or services listed here or later in this work that are mentioned only in an informative way.

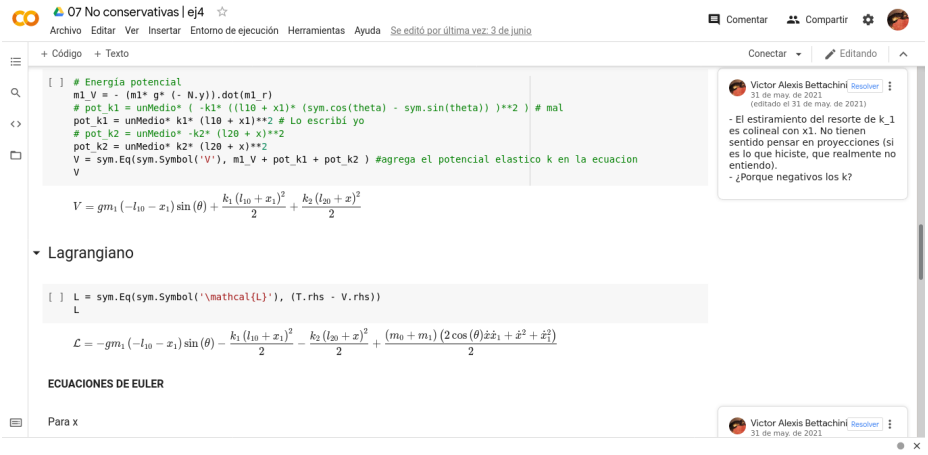

Fig. 4. The Google Colaboratoy website allows Jupyter notebooks to be edited and run concurrently between students and teachers, as well as including comments. The latter feature is useful for corrections.

3.4 Git Repository on GitHub

The aforementioned repository on GitHub is organized into separate folders for each class of the course, as shown in figure 5.

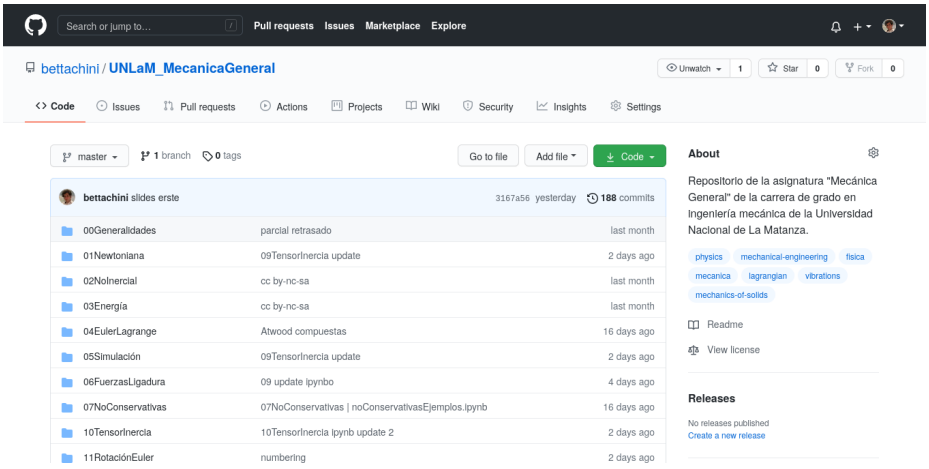

Fig. 5. Github repository: the students find the material organized in separate directories per class [31].

There is a folder for each subject matter containing the Jupyter Notebooks for lessons and exercises as well as exercise set and occasional notes in the portable

document format known by its acronym in English PDF. This arrangement facilitates both the teacher and the students an overview of the material of each topic as well as verifying any updates to it. In this way, the course material is publicly available to interested parties at `https://github.com/bettachini/MecanicaAnaliticaComputacional/` [31] as long as they cite its origin and do not use it commercially as indicated by its Creative Commons CC-BY-NC-SA license [32]. As the course is dictated in castillian (spanish) for the time being all material available in the repository is in that language.

3.5 Learning Management System: Microsoft Teams

At UNLaM, the Microsoft Teams platform was used to replace classroom interaction with students with videoconferencing during the SARS-CoV-2 lockdowns. After each meeting its recording was published to a different page, or channels in the system's terminology, as shown at the left at the figure 6.

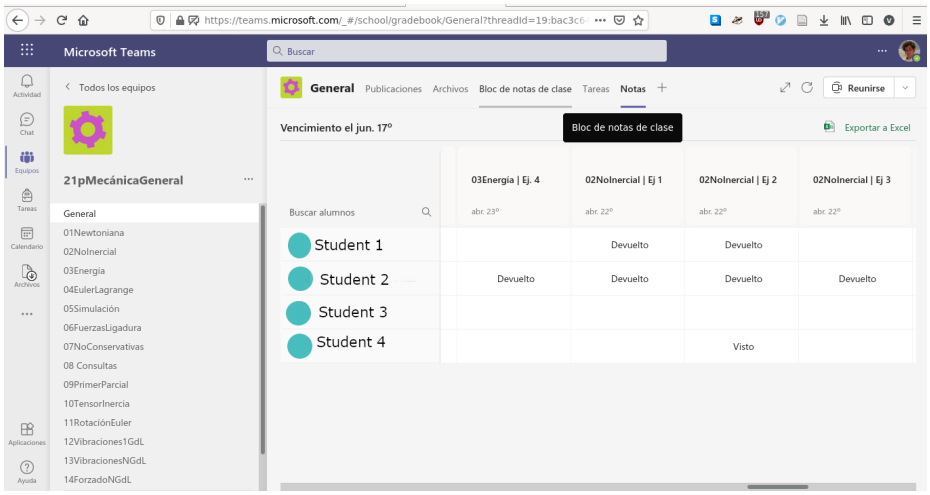

Fig. 6. Separate channels per class (on the left) present links to their material. Student progress tracking is shown on the right.

Microsoft Teams provides the rudiments of a Learning Management System by allowing tasks to be assigned to students with acceptance deadlines. At right of the figure 6 a table that summarizes whether individual students uploaded a link to a Google Colab Notebook containing a solution to a given problem.

4 Syllabus and schedule

Out of the 16 weekly meetings throughout a semester, 13 present new topics. By the end of the second week generalised coordinates were used to calculate

the mechanical energy of various systems with ad-hoc Sympy-based functions provided by the teaching staff. From third to fifth week the Euler-Lagrange approach is implemented into code to simulate the dynamics of multiple point mass systems. By the eighth week, constrained forces and non-conservative ones are calculated. Starting at the ninth week the course focus on extended bodies modelled as rigid bodies employing inertia tensors and Euler equations for rotation. The final weeks are devoted to vibrations including forced harmonic oscillations and modal analysis at systems with multiple degrees of freedom. Further details beyond this brief description of the weekly topics can be obtained from the full schedule available at the course repository [31]. The progression of the course is illustrated by the following selection of some of these topics.

Week 1: vector kinematics After a brief introduction to the flipped classroom methodology and description of the coding tools to use, a hand-on practice on the use of Google Colab begins. As a first exposure to Sympy calculus capabilities, students are presented with a review of vector kinematics with code for time-differention of position vector in some reference frames and coordinate systems as shown in figure 7.

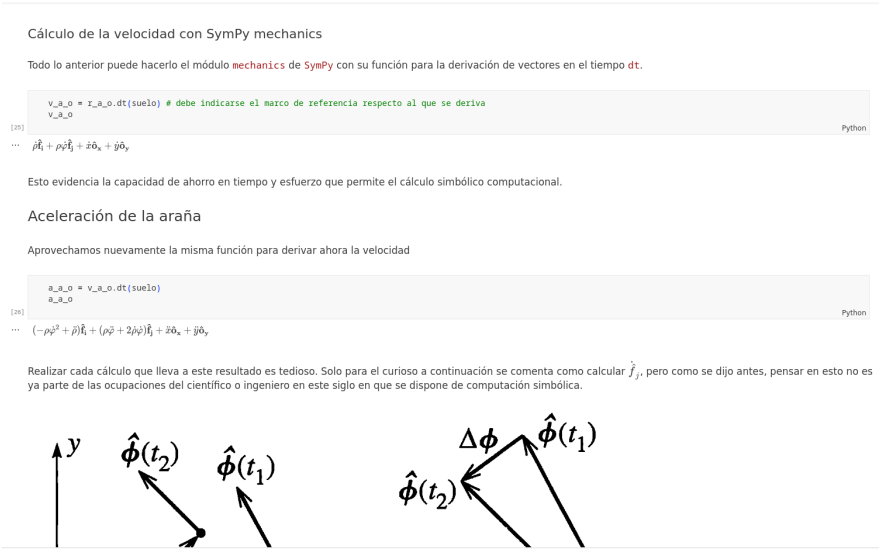

Fig. 7. The first lesson in Jupyter notebooks: a review of kinematics where Sympy calculus functions are used.

Then in a second lesson notebook the construction of a function to calculate the kinetic energy for a compound pendulum is explained step-wise. Students are encouraged to review the LaTeX notation used in that notebook.

Week 3: Euler-Lagrange formalism This is the main tool of analytical mechanics that students will instrumentalise to generate diffential equations to

describe, at this stage of the course, the dynamics of multiple point mass systems. After a step-wise construction, akin that followed with functions for energies, a function to generate these equations is provided to the student in an example problem as illustrated by figure 8. This code will be re-used by the student from now on through the course to address problems at every exercises set.

Fig. 8. The Euler-Lagrange formalism is the center piece of the analytical tools used in the course. After a step by step construction, a fully assembled function would allow to apply it in code used thereafter.

So far, code has been used to perform the same steps that are solved on a blackboard or paper in a conventional rational mechanics course to arrive at differential equations that are only solved for trivial cases. In contrast, using Sympy quickly solves complex systems for acceleration a function of generalized coordinates and velocities as shown in Figure 9. Performing such a task manually would require a non-negligible amount of time and effort even for this system with only two degrees of freedom.

Week 5: simulation Students passed a numerical analysis course to enroll in this course where such knowledge will be used. In this lesson, the fundamentals of numerical resolution methods for differential equations are reviewed and how they would be implemented in a state vector notation suitable for efficient processing. Immediately after the review of fundamentals, the functions of the scientific calculation library Scipy are shown in action to efficiently obtain solutions for the dynamics of a two-degrees-of-freedom system as illustrated by Figure 10.

```
[14]: sistemaEcuaciones = [
          x_EL,
          phi_EL,
      ]
      variablesDespeje = [x.diff(t,2), phi.diff(t,2)] # despejar aceleraciones generalizadas
      variablesDespeje_sol= sym.nonlinsolve(sistemaEcuaciones, variablesDespeje ).args[0]

[15]: x_pp = sym.Eq(variablesDespeje[0], variablesDespeje_sol.args[0] ) # [m s-2]
      phi_pp = sym.Eq(variablesDespeje[1], variablesDespeje_sol.args[1] ) # [m s-2]
      x_pp, phi_pp
```

[15]:

$$\left(\ddot{x} = \frac{-\ell g m_2 \sin(\phi) + \frac{\ell m_2 \left(\ell m_2 \cos(\phi)\dot{\phi}^2 + g m_1 + g m_2\right) \sin(\phi)}{m_1 + m_2 \sin^2(\phi)}}{\ell m_2 \cos(\phi)},\ \ddot{\phi} = -\frac{\left(\ell m_2 \cos(\phi)\dot{\phi}^2 + g m_1 + g m_2\right) \sin(\phi)}{\ell \left(m_1 + m_2 \sin^2(\phi)\right)} \right)$$

Fig. 9. The resolution of systems of differential equations of certain complexity is avoided in conventional courses. In this course, it only takes a couple of lines of code with functions in the Sympy library.

```
[22]: # defino una función con el sistema de derivadas
      # t : no se usa en este sistema pero lo dejamos para uso posterior
      # y : lista de estado con [y[0], y[1], y[2], y[3]]
      # y[0]: x
      # y[1]: x punto
      # y[2]: phi
      # y[3]: phi punto
      # dydt : lista de derivadas
      def y_punto(t, y):
          dydt = [y[1],
                  x_pp_numpy(y[0], y[1], y[2], y[3]),
                  y[3],
                  phi_pp_numpy(y[0], y[1], y[2], y[3]),
                 ]
          return dydt

[23]: # Integración de a pasos en el tiempo
      y_ode2 = solve_ivp(y_punto, (t_rango[0], t_rango[-1] ), y_inicial, t_eval = t_rango)

[25]: y_ode2.y[0]

[25]: array([ 1.        ,  0.95510744,  0.92131146,  0.89820932,  0.88468059,
              0.87877042,  0.87745354,  0.87702754,  0.87352768,  0.86357726,
              0.84474673,  0.81565733,  0.77559949,  0.72423163,  0.66166451,
              0.588266  ,  0.50468237,  0.41250381,  0.31433661,  0.21366454,
              0.11444308,  0.02023394, -0.06599563, -0.14244216, -0.20809592,
             -0.26250272, -0.30576388, -0.33796804, -0.35953138, -0.37175469,
              0.3769223 ,  0.37781911,  0.37854158,  0.38287486,  0.39407826
```

Fig. 10. The system of equations for the dynamics of a two-degrees-of-freedom system is numerically solved with functions of the SciPy library.

Generalized positions and velocities obtained numerically at times of interest are graphically represented, as shown by figure 11, which is useful to discuss students whether the behavior of the system is consistent with what can be predicted from a qualitative analysis of this simple system. Confirming that the symbolic and numerical calculation tools used obtain correct results gives confidence in them in view of applying this tools to more complex systems.

Final weeks. Towards the end of our course the students of our course have already developed the ability to autonomously analyse "realistic" systems resembling mechanical devices found in the industry. To present them a challenge in this line, they are assigned a final project in which they must calculate the torques required for the motors of a highly simplified industrial robotic arm to

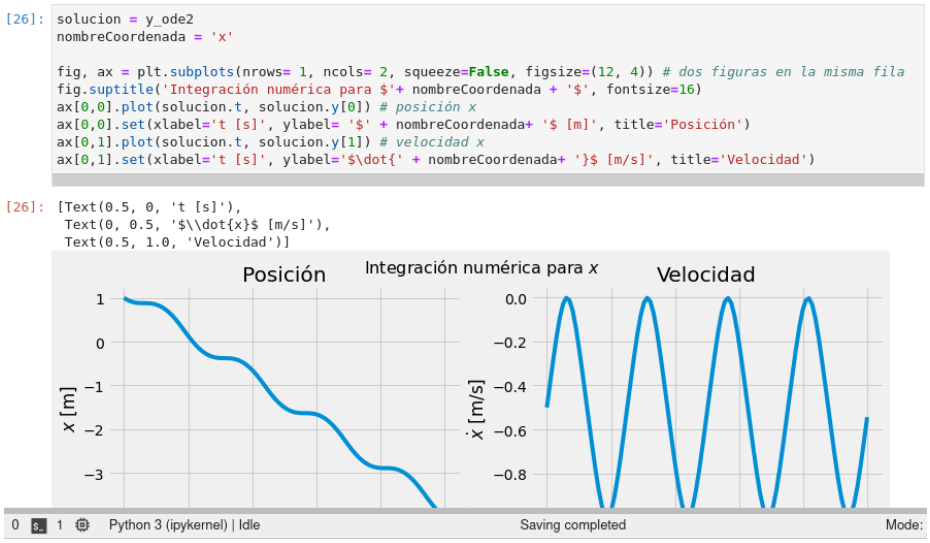

Fig. 11. Visualisation of results obtained by numerical calculations. Corroborating that what is represented corresponds to a qualitative analysis of the dynamics of the system creates confidence in the students in this tool.

perform a sequence of movements. Examples of the results of students' work in response to this proposal are shown in figure 12.

Students are required to present their results in an oral presentation to the teaching staff. With this we aim to push them to improve their conmunication skills, not only by writting their work in an ordely fashion, but also to prepare a speech to defend it, a stressful situation but a must for any business-like setting.

5 Conclusions

This course differs from conventional ones in two ways:

- Code-centric
  - Avoids the repetitive nature of blackboard or paper based calculations.
  - By iteratively modifying previously tested code (initially designed for simpler mechanical systems), students expand their analytical capabilities.
  - The complexity of the code evolves alongside the mechanical system's intricacies introduced each class.
  - This approach eliminates the need to *start from scratch* when dealing with the extensive calculations required for analyzing complex mechanical systems using the Euler-Lagrange formalism.

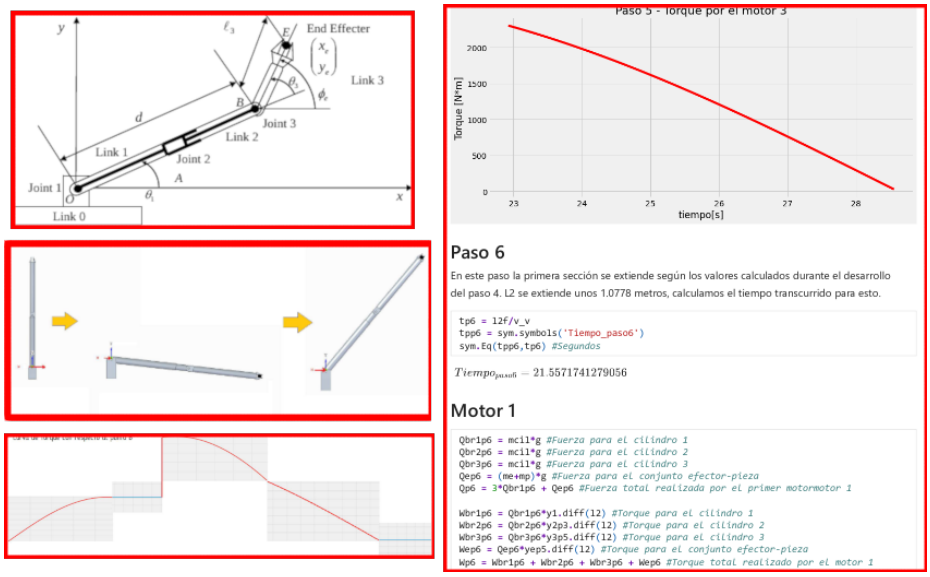

Fig. 12. To be able to perform even a simple movement, motors of an industrial arm must apply a sequence of torques. Students calculate them in a final assignment that reflects their mastery of analytical and computer tools taught at the course.

  - All systems used are currently available online on a non-cost basis, from the student point of view. Being based on free software, if any of them is later placed behind a paywall, it would be simple to run them from on the premise servers.
- Flipped classroom
  - Students are provided with online theory and example problems to study before weekly meetings. These asynchronous activities save classroom time for discussions and problem solving.
  - During synchronic meetings they can rise to teachers any questions related to theory or problem-solving so they can finish their exercise sets.
  - All exercises are turned-in for evaluation. Compliance is tracked with an online learning management system.

Currently, there is limited statistical data available on the impact of the course and the described methodologies. However, feedback from students consistently indicates a high level of satisfaction, especially with the code-driven aspect of the course. Additionally, students express interest in the final examination as it provides an opportunity to apply both their presentation skills and the knowledge acquired throughout the course.

In relation to the flipped classroom model, students acknowledge that it requires more effort, but a majority of them agree that it is a positive and beneficial implementation. as former students assesed that the tools employed in the course were usuful to them in subsequent subjects and professional lives,

the authors had consequently gained confidence on having choosed the current approach over traditional ones.

Acknowledgments. The authors would like to thank the coordination of the Mechanical Engineering career at DIIT-UNLaM for accompanying the development and implementation of this course.

Disclosure of Interests. The authors declare no competing interests.

References

[1] M. Mitchell Waldrop. *The Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal*. Viking Penguin, July 2001. 528 pp.

[2] Seymour Papert. *The children's machine: rethinking school in the age of the computer*. USA: Basic Books, Inc., May 1993. 241 pp.

[3] Dennis M. Roberts. "The Impact of Electronic Calculators on Educational Performance". In: *Review of Educational Research* 50.1 (Mar. 1, 1980). Publisher: American Educational Research Association, pp. 71–98. URL: https://doi.org/10.3102/00346543050001071.

[4] Karl-Eugen Kurrer. "Konrad Zuse und die Baustatik — Zur Formierung der Computerstatik (Teil 2)". In: *Bautechnik* 87.12 (Dec. 2010), pp. 763–783. URL: https://onlinelibrary.wiley.com/doi/10.1002/bate.201010051.

[5] Simon H. Lavington. *A history of Manchester computers*. NCC Publications, 1975. URL: http://archive.org/details/HistoryOfManchesterComputers.

[6] John G. Kemeny and Thomas E. Kurtz. "Dartmouth Time-Sharing". In: *Science* 162.3850 (Oct. 11, 1968). Publisher: American Association for the Advancement of Science, pp. 223–228. URL: https://www.science.org/doi/10.1126/science.162.3850.223.

[7] Charles S. Tidball. "Using Minicomputers for Courseware Delivery". In: *Information Technology in Health Science Education*. Ed. by Edward C. DeLand. Computers in Biology and Medicine. Boston, MA: Springer US, 1978, pp. 229–241. URL: https://doi.org/10.1007/978-1-4684-2460-7_14.

[8] Bill Cope and Mary Kalantzis. "A little history of e-learning: finding new ways to learn in the PLATO computer education system, 1959–1976". In: *History of Education* 52.6 (Nov. 2, 2023), pp. 905–936. URL: https://doi.org/10.1080/0046760X.2022.2141353.

[9] M. Ben-Ari. "Constructivism in computer science education". In: *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)* 30.1 (1998), pp. 257–261.

[10] Cynthia Solomon et al. "History of Logo". In: *Proceedings of the ACM on Programming Languages* 4 (HOPL June 14, 2020), pp. 1–66. URL: https://dl.acm.org/doi/10.1145/3386329.

[11] A.C. Kay. "The early history of smalltalk". In: 2nd ACM SIGPLAN Conference on History of Programming Languages, HOPL 1993. 1993, pp. 69–95.

[12] Cornelius Lanczos. *The Variational Principles of Mechanics*. University of Toronto press, 1952.

[13] *AMS Style Guide. American Mathematical Society.* URL: https://www.ams.org/arc/styleguide/index.html.

[14] A Mirasso, S. Raichman, and E. Totter. "Articulación de estratégias y recursos para el aprendizaje de métodos numéricos en ingeniería". In: *Mecánica Computacional Vol XXXIII*. XXI Congreso sobre Métodos Numéricos y sus Aplicaciones. Bariloche, Argentina, 2014, pp. 2099–2109.

[15] Marta Caligaris et al. "Desarrollo de habilidades matemáticas durante la resolución numérica de problemas de valor inicial usando recursos tecnológicos". In: *Revista Educación en Ingeniería* 14 (Feb. 2018), pp. 30–40.

[16] Jeffrey M. Perkel. "Programming: Pick up Python". In: *Nature* 518.7537 (Feb. 2015). Number: 7537 Publisher: Nature Publishing Group, pp. 125–126. URL: https://www.nature.com/articles/518125a.

[17] Lorena A. Barba and Gilbert F. Forsyth. "CFD Python: the 12 steps to Navier-Stokes equations". In: *Journal of Open Source Education* 2.16 (Nov. 12, 2018), p. 21. URL: https://jose.theoj.org/papers/10.21105/jose.00021.

[18] William Vallejo, Carlos Díaz-Uribe, and Catalina Fajardo. "Google Colab and Virtual Simulations: Practical e-Learning Tools to Support the Teaching of Thermodynamics and to Introduce Coding to Students". In: *ACS Omega* 7.8 (Mar. 1, 2022), pp. 7421–7429. URL: https://pubs.acs.org/doi/10.1021/acsomega.2c00362.

[19] Charles J. Weiss. "A Creative Commons Textbook for Teaching Scientific Computing to Chemistry Students with Python and Jupyter Notebooks". In: *Journal of Chemical Education* 98.2 (Feb. 9, 2021). Publisher: American Chemical Society, pp. 489–494. URL: https://doi.org/10.1021/acs.jchemed.0c01071.

[20] Laureline Duvillard. *Using code to better understand the physics behind equations.* EPFL. URL: https://www.epfl.ch/education/educational-initiatives/jupyter-notebooks-for-education/teachers-experience-with-jupyter-notebooks/using-code-to-better-understand-the-physics-behind-equations/.

[21] Jacqueline O'Flaherty and Craig Phillips. "The use of flipped classrooms in higher education: A scoping review". In: *The Internet and Higher Education* 25 (Apr. 1, 2015), pp. 85–95. URL: https://www.sciencedirect.com/science/article/pii/S1096751615000056.

[22] *Flipped Classroom | Center for Teaching & Learning.* 2024. URL: https://ctl.utexas.edu/instructional-strategies/flipped-classroom.

[23] L.D. Landau and E.M. Lifshitz. *Mecánica.* 1st ed. Ciudad de México, México: Reverté, 1994.

[24] Aaron Meurer et al. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. URL: https://doi.org/10.7717/peerj-cs.103.

[25] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. URL: https://doi.org/10.1038/s41586-020-2649-2.

[26] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272.

[27] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95.

[28] Thomas Kluyver et al. "Jupyter Notebooks – a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87–90.

[29] J. Gruber. *Daring Fireball*. Sept. 28, 2023. URL: https://daringfireball.net/projects/markdown/.

[30] Google LLC. *Using Google Colab with GitHub*. 2021. URL: https://colab.research.google.com/github/googlecolab/colabtools/blob/master/notebooks/colab-github-demo.ipynb#scrollTo=K-NVg7RjyeTk.

[31] V. A. Bettachini. *Repositorio de la asignatura Mecánica General*. URL: https://github.com/bettachini/UNLaM_MecanicaGeneral/.

[32] Creative Commons. *Atribución-NoComercial-CompartirIgual 4.0 Internacional*. URL: https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es.