Introduction to Modeling & Simulation
PM1: (Project Milestone #1) - Model Formulation

As we delve into the details of numerical algorithms, we wish to maintain sight of their applications. The course is composed of participants from a wide range of backgrounds, and we want to encourage students to explore how modeling and simulation techniques can find application in the fields of interest of many students. This motivates the development of self-proposed team-based course-long projects. Instead of developing your project toward the end of the course as a stand alone effort, we will divide up the class project work throughout the entire term, allowing students to apply to their own project progressively each new concept after it is first see it in lectures and practiced in problem sets.

What to bring to the Project Milestone Meeting

You must come to the project milestone meeting fully prepared to show to our staff your well organized written up thoughts, conclusions **and code** related to your tasks. Ideally, all members of the team should be present, and at least one member of the team should be ready to share and show during the PM meeting all of the following:

- your two-pages project formulation summary (if you have changed/modified your project remember to update it) which will be critical to remind quickly your TA of your project setup;
- your shared common team code related to your tasks, running live on your machine;
- in particular you must show and run a script calling a full set of codes that test every one of the main project functions developed so far and in particular EVALF. This is called a regression test and is described below in more details below. The idea is that you must and will run it after any tiny change on any part of your code to ensure you your new additions/modifications did not introduce bugs in other parts of the code).
- any screenshot or pictures/figures/sketches that you may find useful to facilitate your quick description and discussion on the tasks below. Make sure you actually "do look" at the numbers in your plots/tables and are able to explain why they are reasonable/plausible (or not). Not realizing they are not reasonable/plausible will for sure produce loss of points.
- an itemized list of each and all specific contributions (e.g. which task or what part of a task/tasks) for each individual team member. Grades are assigned <u>INDIVIDUALLY</u> to each different member of the team (i.e. each member gets a different grade) based on assessment of individual contributions / participation / understanding of each member independently from the other members. It is ok to divide up tasks among team members, but every member must contribute to the development of a substantial portion of a task or multiple tasks for full credit. Furthermore, each member must demonstrate understanding of ALL the parts i.e. even those completed entirely by other members. If a team member cannot attend a graded project milestone meeting without justification: penalties will be assigned. See the posted course information for all details about grading rubric and justified absence protocols.

If any of the previous items is missing or incomplete PM points will be deducted.

For this project milestone complete ALL the following tasks:

A - Produce ideas and form a team

Form a group of 3 people (number strictly enforced in the first few weeks, exceptions might be allowed later on if people should drop the class), agreeing on a common project. You can/should use our form on the course website to advertise your ideas for your project, ideally from the area of your research or field of interest (remember to include also some links to related material), or use the form to explore ideas from other students and find a group to join.

Note: You will be allowed to modify or even change completely at anytime your project as you progress through the class by resubmitting a new or updated two-pages problem formulation, if you find that your initial idea is too difficult to implement, or you discover a more interesting approach or problem. However, if you settle right away on a project and you keep using the same in each project milestone, by the end of the last project milestone you will pretty much have already done the vast majority of the work for your class project and you will be able to focus all your energy in just "presenting" it well. The key goal of these Project Milestones is to distribute some of the course project work throughout the course rather than compressing it all at the end of the course. In order to increase your chances of using the same project throughout all assignments, you may want to think of a project that possibly has as many of the following attributes as possible (although not necessarily all of them):

(a) your system displays some <u>non-linear</u> behavior or contains some non-linear components. Make sure you understand where exactly all the non-linear components are or exactly which terms of the equations are responsible for the non-linear behaviour of the system. Here is a non-exhaustive list of typical phenomena that can result in non-linearity:

- for excitations that are "too large" many components eventually stop working perfectly (e.g. saturation phenomena) or often even break completely;
- some components have a non-symmetric behavior for example with respect to the sign of the excitation. E.g. some are completely uni-directional not responding at all when the sign of the excitation is negative (i.e. diodes valves, one way roads etc.) or they respond in a completely different way when the excitation changes sign.
- that is actually a special case of the more general situation when components or parts of systems switch from one kind of behavior to another when certain conditions happen, so that they have multiple modes of operations determined not just by the sign of the excitation but also by its magnitude or more complex conditions.
- inspect the constitutive equations of your components... if you see any exponent or roots or divisions or absolute values or sign functions or trigonometric functions where the argument is the excitation... that is most likely a nonlinear component...
- note that some "systems" (e.g. electronic amplifiers for music or for signal processing) are made out of extremely non-linear components, but are designed to work in a very linear way as a system. In this case it will be your choice how you treat the non-linearities, based on the reason you are developing the simulator: if you are developing a simulator as a small part of a large system, then you may get away with modeling the system as linear, but if your goal in developing the simulator was to assess how good the design was in making the system linear... well then you MUST model everything as nonlinear otherwise you will not see anything of what you are looking for.
- some components (i.e. structural strut) when they are aligned with one of the coordinate axis or when they are excited exclusively along their length without any rotation, but when they are pulled in different directions they may rotate creating non-linearity due to the different breakdown of force components in the chosen coordinate system.

Having said that, in "many" tasks throughout the course, we may just ask you to "linearize" your system, i.e. compute a Taylor Series expansion of all equations and truncate it at first order (i.e. just using the Jacobian). We also recognize that many components and systems are indeed always fully linear and if your system is one of them no problem: it will still fit the class provided you will have "some" of the other elements listed below.

(b) your system has some "<u>dynamical</u>" behaviour, so that eventually it will be of interest to analyze how your system evolves "over time" in response to either generic or periodic inputs/ loads/ sources/ excitations, $u(t)$. The word "dynamical" could be interpreted differently in different fields. So here is a non-exhaustive list of typical phenomena that can result in the specific meaning of the word "dynamical behavior" that we are using in this class:

- inspecting you equations you notice that at least some of your equations include also some derivatives with respect to time dx/dt of some (or all) the variables in the system state $x(t)$.
- some of your nodes or components have storage capability or inertia, i.e. they store energy and they can give it back, or can maintain it... e.g. a heavy particle in motion in vacuum will continue moving even if not excited... it stores kinetic energy at a level represented by its velocity, and in order to change that velocity you need to act on it providing or removing energy. A huge body of water such as an ocean is difficult to heat up and cool down despite the sun being super strong on a given day... surprisingly the idea of being "slow" at changing temperature is an indication that the system does have a "dynamical" behavior in the sense of the words "dynamical" we intend in this class
- in general if you conclude that the future behavior of your system depends NOT ONLY on the future excitations but also what you did to your system in the past, then you have a "dynamical" behavior.
- if you were to "kick" the system, (i.e. provide some very short impulse excitation and then nothing else...) and you observe or expect a response even after the kick stopped acting, your system does probably have a "dynamical" behavior".

Having said that, in "many" tasks throughout the course, we may just look at steady state behavior (i.e. no dynamics: all you will have to do, just for those cases, will be to set those state time derivatives to zero). Also, if your system does indeed not have any dynamical behavior it could still be a good fit for our class provided you have some of the other behaviors in this list.

(c) your system ise *very* large or could be scaled easily to a very large example (e.g. much more than 10-20 components and nodes, actually ideally at least hundreds, if not thousands, or millions...) so that you will be able to test the effectiveness of the simulation techniques you will be implementing in dealing with practical very large problems, or you could appreciate the speed-up produced by some "model order reduction" techniques. Having said that, in the early weeks of the course you can start with a simplified much much smaller versions of your project. But do make sure that your code can very easily scale up the size later on by simply changing a parameter from say $N = 20$ to $N = 2000$ producing and solving now 2000 equations rather than just 20, **without having to modify any code** (which would result in having to spend another couple of weeks debugging it).

Do make sure you consult very carefully the table in the appendix of the Project Report Template on Overleaf posted on the course website. That table contains a non-exhaustive list of possible systems organized by properties (e.g. linear/non-linear, static/dynamical, small/large etc...) and for each of those many combinations it shows what kind of technical challenge you could be developing and which project milestones would help you with that and hence would be strongly suggested if not required for your project.

Hint: one of the best ways to both get inspiration for your possible class project as well as get a good idea for what kind of projects could work out is to see what has already been done in the past. Therefore, please browse the list of projects developed by previous students in the class here:
`https://docs.google.com/document/d/1jQMu8H73UpZ5dFaYHVI-UgK8enhhv65LQjYxHgbGPTY/edit?usp=sharing`

Hint: Here are examples of ways you could loose points from this task:

- you are not showing up or you are late without an acceptable justification;
- you have not found a team and worked as part of that team in preparation for the PM meeting;
- you have not made enough efforts to fit your ideas to the specific framework of the class (e.g. did not read or did not follow the indications in the notes above, or did not watch the presentations from previous years)
- your team is not keeping the project/team signup form updated at all times, specifically maintaining up to date team composition and members' information, project title, references and additional project keywords and short descriptions. One crucial thing you need to keep up to date at all times on that document will be which project milestones you are planning on having graded, and why (i.e.

relating the properties of your system to the topics covered in each project milestone). Remember you can change which PM you will be graded at no penalty, but you will loose points if you do not constantly update that document as soon as you decide to change them any project milestone.

- you have not considered/discussed decided how your system classifies with respect to the properties listed above in this task and in the table of technical challenges at the end of the report template document.

B - Sketch-up a two-pages-max project formulation summary

In order to allow our staff to quickly learn (on the first meeting) or remember (in future meetings) your project, each team should maintain an online version and bring to the project milestone meeting two-pages-max description. If your handwriting and your hand-drawing is crystal clear, you can even show up with a hand-sketched piece of paper: your focus should be on your ideas/contents. (Note: don't worry about perfect formatting and perfect technical English at this stage: we will focus on your actual technical writing skills later in the course requiring different documents.) Your two-pages-max of notes should include only:

(a) a list of which of the remaining Project Milestones you tentatively plan to have graded and how your system properties specifically relate to the topic developed in those project milestones you chose to have graded (Remember that the project milestones are just a way to "distribute" the course project throughout the term, therefore you should not have a specific PM graded just to get it done or get rid of it early if it does not allow you to complete part of your actual project! On the opposite you MUST have a specific PM graded if it does cover some fundamental aspect of your course project... e.g. if you are planning to do some model order reduction you must have PM6 graded. If you have dynamics you must have PM5 graded etc)

(b) one or more pictures/figures/sketches of your system

(c) clear and **concise** answers to the following questions:

- what do the nodes of your network represent? [Note: nodes do not always represent geometrical or physical entities. They could be also distributed entities such as the overall concentration of each chemical species in a room, or abstract entities such as the impact factor of a research journal or of an internet Twitter. Also note that there could be more than one quantity associate with each physical node, e.g. a temperature, and an electrical potential]
- what are the quantities (i.e. variables i.e. unknowns) associated with each node? [Note: there could be more than one quantity associated with each node, (e.g. each node could be associated with the 3 cartesian coordinates of a structural joint). And they could even represent different phenomena and have different units (e.g. each node could have both a temperature and a voltage associated with it)]
- what do the components of your network represent? [Note: as for nodes, these might not be necessarily "physical" components. They could be representing for instance the friendship links connecting two people on Facebook. There could be different "types" of components. Each single type of component could be connected to one, two or *even multiple nodes*, even all nodes at ones...]
- what are the quantities (i.e. variables i.e. unknowns) associated with the each component? [Note: make sure you do not confuse them with physical "parameters" of the component (e.g. in a resistor the component variable is the current, while the resistance is a parameter. In a spring or strut, the component variable is its reaction force while its elasticity is a parameter. Also note that there could be different phenomena involved in the same system. So some components could carry a heat flow and others could carry electrical current and other could even carry both at the same time: in that case treat them as two distinct components)]

- what are the equations that represent the behavior of the network at each node (e.g. any conservation laws, or anything else that specifies which components are connected to each specific node) [Note: there could be more than one equation in each node (e.g. three cartesian components of the balance of forces of a structural joint). Note: different nodes could have different number and different kind of equations, with different units (e.g. on the same node there could be a current conservation law as well as a heat flow conservation law. Finally, note that although in class we keep using conservation laws, your equations do not necessarily need to be conservation laws]
- what are the equations that specify the behavior of each kind of component? [Note: your network could have and actually most likely WILL have a variety of different components each characterized by a different behavior equation. Typically these constitutive equations specify a function relating the component quantities to the nodal quantities attached to that component. Note: component constitutive equations may end up being VERY often non-linear: not only that is perfectly ok and we can handle it... we also VERY much do like and encourage that! although not all components need necessarily be non-linear. And honestly even if every single component is linear your system could still work for this class.]

(d) a clear, **concise** mathematical problem formulation summary:

- Choose an appropriate "formulation type" for your problem (e.g. node-branch or nodal analysis,... or... some modified nodal analysis). Note: if your system does not have conservation laws and components connected at some notes, you could still fit the class and you could skip this part, however you will still need to clearly identify all the remainder of the items below.
- (points will be for sure deducted if this is not completed and presented as stated) Specify the "state vector" of your problem formulation, substituting $x_1, x_2, ..., x_N$ below with your actual state components (or true and only unknowns):

$$x = [x_1 \ x_2 \ ... \ x_N]^T$$

- Specify the parameters (e.g. geometrical dimensions, material properties etc of your components)? Collect them and list them all substituting $p_1, p_2, ...$ below with your actual parameters

$$p = \{p_1 \ p_2 \ ...\}$$

  where each element in the set/structure could be of different kind if desired/useful e.g. p_1 could be a scalar, while p_2 could be a vector or a matrix, p_3 could be a character or a string etc...
- Specify inputs/ sources/ excitations/ loads of your system. Collect them and list them all substituting $u_1, u_2, ...$ below with your actual sources in the input vector

$$u = [u_1 \ u_2 \ ...]^T$$

- (points will be for sure deducted if this is not completed and presented as stated) Summarize the "final" set of equations that describe entirely your problem casting them into the following standard dynamical state space system

$$\frac{dx}{dt} = f(x, p, u) \tag{1}$$

  In particular, you job would be to specify explicitly each scalar equation $f_i(\cdot)$ of the "vector " function $f(\cdot)$, by substituting explicit expressions for your own $f_1(\cdot), f_2(\cdot), ...f_N(\cdot)$ in the expression below:

$$f(x) = \begin{bmatrix} f_1(x, p, u) \\ f_2(x, p, u) \\ ... \\ f_N(x, p, u) \end{bmatrix} \tag{2}$$

[**Hint1:** One easy way to cast your equations into the standard format above is to identify and isolate all *time* derivatives of each state component bringing them to the left hand side. The right hand side will simply contain everything else so that the i-th equation would look like $\frac{dx_i}{dt} = f_i(x, p, u)$. That is to say, if your model equations have *time* derivatives, you should **not** approximate them with finite differences at this stage! If instead some of your equations in your system really do not have any derivatives *in time*, you would end up with a zero on your left hand side on some of the equations, $0 = f_i(x, p, u)$. Please note this often happens when scientists and engineers tend to oversimplify their problem by neglecting some dynamics that they know to be much much faster than all the other dynamics, thinking it would result in a computationally easier problem: well... that is NOT necessarely always true... Actually often it complicates things. We strongly suggest that you spend the time to analyze and quantify the speed of those very fast dynamics and do add a time derivative with an appropriately small coefficient that matches that actual speed of those dynamics.
If none of the equations has time derivatives (i.e. $0 = f(x, p, u)$), we call that a "steady state" problem, because it expresses the fact you are not at all interested in how the system evolved into that steady condition. Once again many people think that neglecting dynamical evolutions, if not of interest, would make the computation easier... but that is not necessarily true... actually it is surprisingly harder to solve directly a steady state problem rather than a dynamical evolution problem. A steady state formulation "might" indeed be ok as project for the class in some cases, but do check with the instructor. [By the way: setting all time derivatives to zero is actually what all other teams will be asked to do in many initial tasks in the first month of the class, but not all!]
Hint2: By the way, although not highly desirable, it will also still be ok if your formulation needs to include a few algebraic equations together with the differential equations (DAEs): $\begin{bmatrix} \frac{dx}{dt} \\ 0 \end{bmatrix} = \begin{bmatrix} f_d(x, p, u) \\ f_a(x, p, u) \end{bmatrix}$, or even $\begin{bmatrix} E\frac{dx}{dt} \\ 0 \end{bmatrix} = \begin{bmatrix} f_d(x, p, u) \\ f_a(x, p, u) \end{bmatrix}$. However, your model vector function $f(x, p, u)$ should still always be defined to include all scalar functions i.e. both the differential and the algebraic ones: $f(x, p, u) = \begin{bmatrix} f_d(x, p, u) \\ f_a(x, p, u) \end{bmatrix}$
Finally, we should also admit that being a "network of components with conservation and constitutive equations" is actually not a strict requirement for the project. However **your proposed system of equations must at least match the standard "continuous time state space system" framework in eq. (1).**

- Specify the quantities of interest that you care to analyze/observe/characterize? Collect them and list them all substituting $y_1, y_2, ...$ below with your actual quantities of interest

$$y = [y_1 \; y_2 \; ...]^T$$

Specify equations that allow to compute your output quantities, if the state is known. Specifically, substitute expressions for your own $g_1(\cdot), g_2(\cdot), ...$ in the equations below:

$$\begin{aligned} y_1 &= g_1(x, p, u) \\ y_2 &= g_2(x, p, u) \\ &... \end{aligned}$$

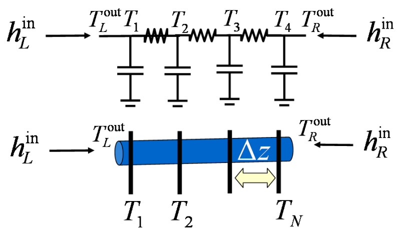
EXAMPLE: If your system were for example a heat conducting bar "similar" to the one described in lecture4, then your two pages document should look something like the following page:

—————— START OF PROJECT DESCRIPTION EXAMPLE —————-

Title: Heat Conducting Bar Project

(a) Internal Team Meeting :
- date and time (must be on the day before the PM meeting):
- attendee list (but everyone must attend):

(b) Graded Project Milestones:
PM1, required for all
PM5: required since the problem involves time simulation
PM6: required since the problem is fully linear and large and could use model order reduction
Either PM2 or PM3, since the problem is linear and sparse
We will not have PM4 graded because our probably is fully linear.



(c)
- The bar is subdivided into N sections.
  Nodes: the connection points between sections and are characterized by their **temperature**.
  Components: the bar sections, and **heat flow** is the quantity associated with each component.
- state: $x = [T_1\ T_2\ ...\ T_N]^T$, a vector containing all the nodal temperatures.
- parameters: $p = \{L,\ \kappa_m,\ \kappa_a,\ \gamma,\ T_a\}$, where:
  L is the length of the bar,
  κ_m thermal conductivity of the bar,
  κ_a is the leakage constant to the surrounding ambient,
  γ is the thermal capacitance of the bar,
  T_a is the ambient temperature
- inputs: $u = [h_{s_L}\ h_{s_R}]^T$, representing two independent sources located on the far left side and at the far right of the bar.
- The final set of state-space equations is given by: $\frac{dx_i}{dt} = f_i(x,p,u))$ where,

$$f_1(x,p,u) = -\frac{1}{\gamma}\left[\frac{\kappa_m N^2}{L^2}(\qquad x_i \ - x_{i+1}) + \kappa_a(x_1 - T_a) - u_1\right]$$
$$f_i(x,p,u) = -\frac{1}{\gamma}\left[\frac{\kappa_m N^2}{L^2}(-x_{i-1} \ + 2x_i - x_{i+1}) + \kappa_a(x_i - T_a) \qquad \right], \qquad i = 2,...,N-1$$
$$f_N(x,p,u) = -\frac{1}{\gamma}\left[\frac{\kappa_m N^2}{L^2}(-x_{N-1} + x_N \qquad) + \kappa_a(x_N - T_a) - u_2\right]$$

The original governing equation in differential form for the system was $\frac{dx}{dt} = f(x,p,u)$ where,

$$f(x,p,u) = \frac{1}{\gamma}[\nabla \cdot (\kappa_m \nabla x - \kappa_a(x - T_a) + \delta(z-0)u_1 + \delta(z-L)u_2]$$

- outputs: $y = [T_L\ T_R]^T$, the temperatures at the far left side, and at the far right of the bar.
- the output can be obtain from the state using $y = g(x,p,u)$ where

$$g(x,p,u) = \begin{bmatrix} 1\ 0\ \dots\ 0\ 0 \\ 0\ 0\ \dots\ 0\ 1 \end{bmatrix}$$

—————— END OF PROJECT DESCRIPTION EXAMPLE —————-

C - Code up a function implementing your model evaluation and its test-bench

Code up a function EVALF that evaluates your own $f(x, p, u)$ given specific values for vectors x, p, and u. Make sure your function is a stand alone file since you will need to call that code (and NOT cut and paste) over and over in all future Project Milestone Tasks. In addition to the two-page project summary, remember to bring to the Project milestone meeting also at least "one" laptop to show to the staff your code running.

Coding tips: Cut and Paste can often be a recipe for disaster. You may think at first that it is safe. But imagine that for instance you will later decide to change something in your evalf (either because you found a bug, or because you need to add a feature or a term to make it more accurate):

- will you be able to remember and update "all" the places where you cut and past the buggy or simpler version?
- will you even remember that you do need to update those places?

Never duplicate functions. Write and keep modifying a single copy; keep "calling" that same exact function from any other code that needs to use it. The more places you call that function from, the more chances you have to find and fix all its bugs. And in general try to avoid the temptation of making continuously feature additions to that function, since every change could introduce more bugs, or break the behavior of other codes that were using without issues the previous version). To this extent it is crucial (and points will be deducted if missing) to generate a set of tests (i.e. a regression test, described below in more details) that you must run after any tiny change on a function that is used by many other codes.

For those same debugging and bug-control purposes, in general it is also a good idea to keep short and limited in scope any of those functions that need to be called from other codes. This means you will want to have possibly many of such short/limited reusable and fully debugged relieable functions to be called bu other codes.

Hint. As an example if you were to write a function evaluating the heat conducting bar formulation, it would have a pseudocode description similar to the following:

Algorithm 1 Example of EVALF model function using stamping for a heat conducting bar

```
Require: $f$ is the vector field of system: $\frac{dx}{dt} = f(x, p, u)$
Ensure: $f(x, p, u) = \frac{1}{\gamma}[\nabla \cdot (\kappa_m \nabla x) - \kappa_a(x - T_a) + \delta(z - 0)u_1 + \delta(z - L)u_2]$
  function EVALF($x, p, u$)                                   ▷ $p = [L\ \kappa_m\ \kappa_a\ \gamma\ T_a]^T$
    $\Delta z \leftarrow \frac{L}{N-1}$
    for each bar section $j = 1$ to $N - 1$ do                 ▷ Stamp each bar section
      $\tilde{h} \leftarrow$ BARSECTIONFLOW($x_j, x_{j+1}, \kappa_m, \Delta z$)
      $f_j \leftarrow f_j - \tilde{h}$
      $f_{j+1} \leftarrow f_{j+1} + \tilde{h}$
    for each node $i = 1$ to $N$ do                            ▷ Stamp each leakage element
      $\tilde{h} \leftarrow$ LEAKAGEFLOW($x_i, T_a, \kappa_a$)
      $f_i \leftarrow f_i - \tilde{h}$
    $f_1 \leftarrow f_1 + u_1$                                  ▷ Heat source at the far left
    $f_N \leftarrow f_N + u_2$                                  ▷ Heat source at the far right
    for each node $i = 1$ to $N$ do                            ▷ Don't forget the thermal capacitance!
      $f_i \leftarrow \frac{f_i}{\gamma}$                       ▷ Can specify different values $\gamma_i$ if needed
    return $f$
```

```
  function BARSECTIONFLOW($T_1, T_2, \kappa_m, \Delta z$)     ▷ Define a function for each type of component
    $\tilde{h} \leftarrow \frac{\kappa_m}{\Delta z^2}(T_1 - T_2)$
    return $\tilde{h}$
```

```
function LeakageFlow($T, T_a, \kappa_a$)        ▷ Keep component functions in the same file as evalf
    $\tilde{h} \leftarrow \kappa_a(T - T_a)$
    return $\tilde{h}$
```

You may find it more convenient (both in your code but also in your two-pager and in your pseudo-code) to define p as a "structure" rather than a vector (e.g. in matlab `p.ThermalConductivity`, `p.BarLength`, etc... note that a structure could conveniently contain also matrices such as: `p.A, p.B`).

Please note that the key goal at this stage is **NOT AT ALL** to write any simulator or to solve your problem. The key and ONLY goal at this stage is to write exclusively the function model EVALF that "describes" your system, without any attempt to solving it yourself!!! That means not only "writing" the function (which should take no more than 10% of your coding time) but rather fully and completely testing it and fully and reliably debugging it. And THAT should always take no less than 90% of your total coding time... no matter what that time is, no matter that your code is!

So how to test and debug effectively a function? One of the best ways is to test a new or modified function is by calling it from another already extensively pre-tested and pre-debugged code. For instance a solver that is part of numerical package built into your programming language (e.g. in matlab there is a variety of such pre-made solvers you can choose from and you could use more than one, e.g. ode45). You could use also the SimpleSolver code posted with PM1 on the course website. However You do NOT want to (and you should NOT) write or modify also the solver code when you are trying to test your model function. Why? Because if the result of "your" new/modified solver calling "your" new/modified model function look suspicious or plain out wrong... you will not know if the bugs are in the solver or in the model. If you use a reliable/debugged/unmodified solver on your model function and you get suspicious results... you KNOW the bug is in your model function! Remember once again that the key goal "at this very stage" is NOT at all to understand our SimpleSolver or Matlab's ode45! At this very stage, those ODE solvers should be considered exclusively as black-box debugging tools that you can/should use to accomplish **your main and only current task, i.e. debugging your model function:** EVALF.

What should you be looking for when you debug your EVALF *model function?* This has to do with building a **regression test**, e.g. a set of tests that allow you to identify any issues with your function. For instance you what to set parameters and inputs for which you do know what the result should be (e.g. very simple setting where you can solve the problem analytically) or you have a very strong intuition for what plausible results should look like. You then use the solver from the library (or the one we provided) to produce and **visualize the time domain evolution of your unknowns**. Do they look "reasonable" (meaning you can reason to yourself and to the staff that the curves you are getting do make some intuitive sense)? If not... most likely in 90 % of the cases you simply have a bug in your model function (although in some cases it is also possible you forgot to include some effects in your model function EVALF).
Getting more specific on what other tests you should run to debug your EVALF: if the solver from your library allows your to set a parameter similar to ω in our `SimpleSolver`), then first start with a relatively large value of ω and decrease it until the solutions x you obtain "converge" (i.e. the solutions do not change significantly anymore from iteration to iteration as you keep decreasing ω). The amount of change in x you observed between the last two values of ω can be used as a "reasonable" indicator for the level of accuracy you have achieved with that last ω. Decide what level of accuracy in x you "really" need, (i.e. no need to aim at 16 digits of precision all the times... think what you need practically speaking) and report it during the PM1 meeting, explaining the practical reasons for your choice: why not higher? why not lower? Keep using that value for any future experiments.
Next, test your model function $f(x, p, u)$ applying different constant input source values and setting different parameter values in p, for which you have an intuitive expectation for what the answer should or could be and make sure your code reasonably matches your expectations. **Hint1**: if your model does not have a time derivative it means you only care about the steady state problem $0 = f(x, p, u)$. well... you can still find and use some library-provided solver or you can use again our same SimpleSolver to debug your model function $f(x, p, u)$ by looking at what values of x the solution seems to stabilize at, and deciding if

they do make some intuitive sense to you]. **Hint2**: By the way, our same SimpleSolver can also give you (again without modifications to the solver... and this is important) the steady state step response in case you have a mixture of differential and algebraic equations (DAEs):
$\begin{bmatrix} \frac{dx}{dt} \\ 0 \end{bmatrix} = \begin{bmatrix} f_d(x,p,u) \\ f_a(x,p,u) \end{bmatrix}$, by simply defining $f(x,p,u) = \begin{bmatrix} f_d(x,p,u) \\ f_a(x,p,u) \end{bmatrix}$ and passing it to our SimpleSolver.

Come prepared to the meeting to demonstrate how you convinced yourself that your code evalf evaluating $f(x,p,u)$ is bug-free running and showing and explaining the results of your regression test. What tests did you include in your regression test script? Think of special cases where you know what the solution should be and make sure the code produces it. And most importantly make sure that ALL those functions you wrote to test your evalf are nicely organized in a folder and are called by a single "**regression test**" script testevalf which you will run over and over for the rest of the course after each and every time you will have to make even a tiny change to your evalf for whatever reason!!! Your TAs will ask to see your evalf running and passing all your tests succesfully. It is often a good idea to have a fastmode and a overnight mode. The modes would defer only by the value of one parameter defining the system size e.g. the total number of nodes N. For a fastmode you would set N to values so small that you can run the entire testbench in few seconds. Instead in the overnight mode you would first run the fastmode, and then you would rerun it with N much larger: you would run at the end of every day since it would take several hours to run and you will be able to start your day with an evalf that you know is absolutely solid for both small and large problems.

D - Code up a test-bench for a Jacobian function

Develop a test-bench for either our provided function or your own function that evaluates the linearized matrix i.e. the Jacobian matrix $J_f(x,p,u)$ given the function $f(x,p,u)$ and given specific values for x, p, and u.

Note: if you develop your own function, do make sure it is a stand alone file and it shares a common interface structure shownd below, since you will be reusing it over and over in all your future Project Milestone Tasks.

Algorithm 2 Evaluate Jacobian of the model function

```
Ensure: $J^f = \frac{\partial f}{\partial x}\Big|_{x,p,u}$
  function EVALJACOBIANF($f(\cdot), x, p, u$)
      ...
      $J^f \leftarrow \dots$                    ▷ add here the analytical expression for your Jacobian
      return $J^f$
```

A test-bench consists of a regression test script which should run several tests calling the Jacobian function and making sure they it produces correct or at the very least plausible results. Note: every time you will make changes in the future to either model function f() or potentially your own Jacobian funciton, you will use this regression test script to identify potential bugs your moifications may have introduced. In particular, come prepared to the project milestone meeting to show your regression test script in action and to demonstrate how you convinced yourself that the Jacobian matrices produced by the Jacobian function are correct or at the very least plausible. What tests did you include in your Jacobian regression test script? Think of special cases where you know what the solution should be and make sure the code produces it. Try solving simplified linearized problems using back slash for which you have an intuitive expectation for the answer etc... For instance if your project happens to be completely linear (i.e. $f(x) = -Ax + b$, you could make sure that indeed what it produces is $J^f(x,p,u) = -A$.

The pseudocode of a finite difference Jacobian function (such as the one we provided) is given for reference below, where $\hat{e}_k$ is the unit vector in direction k, and dx_k is a "small" number representing a small perturbation of state component x_k.

Algorithm 3 Evaluate Jacobian of the model function using Finite Difference

```
Ensure: $J^f \approx \left.\frac{\partial f}{\partial x}\right|_{x,p,u}$
  function FiniteFifferenceJacobian($f(\cdot), x, p, u, dx$)
    for each column $k$ of $J^f$ do
      $\left[J^f\right]_k \leftarrow \frac{f(x+dx_k\,\hat{e}_k,p,u)-f(x,p,u)}{dx_k}$
    return $J^f$
```

It is imperative that in your test-bench you include experimentation with different values of dx_k. What happens as you start from a large value and decrease it? and what happens at one point when it is too small? You should at the very least experiment and show and understand what happens with the following two options to begin with:

- $dx_k = \sqrt{\texttt{eps}}$ where `eps` is the relative machine precision of your computer.
- $dx_k = 2\sqrt{\texttt{eps}}\max(1, |x_k|)$, which should help re-normalizing when $|x_k|$ is very large. Note: if you are using the same dx for each of the state components x_k then you can use instead $dx_k = 2\sqrt{\texttt{eps}}\max(1, ||x||)$

None of the two options above might work well and we cannot give you a value that is guaranteed to work for all cases since values can be very case specific. Therefore you need to understand the range of values that work well for your case, and be safely within that range. This will turn out to be a very valuable function that you will use over in future Project Milestones tasks, specially if you were to change your project later on! To do a better job in your analysis, make a plot with dx on the horizontal axis and $||J_{FiniteDiffer} - J_{othermethod}||$ or $||J_{FiniteDiffer}(dx) - J_{finiteDifference}(2dx)||$ on the vertical axis. Use log scales for both axis. Initially you may see behavior that is hard to explain: you still have bugs somewhere. Eventually as you get rid of most bugs, you should start seeing a plot that at first keeps decreasing toward an optimal value of dx and then starts increasing back.

Notice also that your state x might contain quantities expressed in different units and different orders of magnitude in size. Therefore the range of values dx_k that work best for one state component might be different from the range of values for a different state component.

The key goal of this task is to develop a regression test for based on your system and your own EVALF. Therefore you are not required to develop also your own Jacobian function for this task (e.g. you can use the finite difference Jacobian function we provided). However if you choose to develop also your own Jacobian function, here are two additional approaches you could consider:

(a) you could use hand-calculated analytical derivatives of each *complete* equation in your system;

(b) OR you could iterate over each component in the network and 'stamp' the Jacobian using hand-calculated analytical derivatives for each component. This is a more scalable way of implementing it, hence more recommended (for a much later stage of your project in the course). See our pseudocode example below to help you get started);

Algorithm 4 Example of Stamping Jacobian function for heat conducting bar

```
Require: $f(x,p,u) = \frac{1}{\gamma}[\nabla \cdot (\kappa_m \nabla x) - \kappa_a(x)(x - T_a) + \delta(z-0)u_1 + \delta(z-L)u_2]$
Ensure: $J^f = \frac{\partial f}{\partial x}\Big|_{x,p,u}$
  function StampJacobianHeatBar($f(\cdot), x, p, u$)                    ▷ $p = [L\ \kappa_m\ \kappa_a\ \gamma\ T_a]^T$
    $\Delta z \leftarrow \frac{L}{N-1}$
    for each bar section $j = 1$ to $N-1$ do                              ▷ Stamp each bar section
      $g \leftarrow$ BarSectionFlowDerivative($x_j, x_{j+1}, \kappa_m, \Delta z$)
      $J^f(j, j) \leftarrow J^f(j, j) - g$
      $J^f(j, j+1) \leftarrow J^f(j, j+1) + g$
      $J^f(j+1, j) \leftarrow J^f(j+1, j) + g$
      $J^f(j+1, j+1) \leftarrow J^f(j+1, j+1) - g$
    for each node $i = 1$ to $N$ do                                       ▷ Stamp each leakage element
      $g \leftarrow$ LeakageFlowDerivative($x_i, T_a, \kappa_a$)
      $J^f(i,i) \leftarrow J^f(i,i) - g$
    for each row $i = 1$ to $N$ do                                        ▷ Don't forget the thermal capacitance!
      for each column $j = 1$ to $N$ do
        $J^f(i,j) \leftarrow \frac{1}{\gamma} J^f(i,j)$                   ▷ Can specify different values $\gamma_i$ if needed
    return $J^f$
```

Algorithm 5 Define a different function for each component type

```
  function BarSectionFlowDerivative($T_1, T_2, \kappa_m, \Delta z$)
    $g \leftarrow \frac{\kappa_m}{\Delta z^2}$                            ▷ Linear component: does not depend on temperatures
    return $g$

  function LeakageFlowDerivative($T, T_a, \kappa_a$)                      ▷ Linear case
    $g \leftarrow \kappa_a$                                               ▷ Note does not depend on temperature
    return $g$

  function LeakageFlowDerivative($T, T_a, \kappa_a(\cdot)$)               ▷ Non linear case
    $d \leftarrow \ldots$                 ▷ Put here the expression of the analytical derivative $\frac{\partial \kappa_a}{\partial T}\Big|_{T-T_a}$
    $g \leftarrow \kappa_a(T - T_a) + d \cdot [T - T_a]$                  ▷ Note does depend on temperature
    return $g$
```