**This document presents how the report and the project should be developed in order.**

**Your target audience is someone who is already familiar with the content of the course. There is no need to repeat a large part of the theory, as it is expected that you know how to do that, given enough time, books, slides, and internet bandwidth.**

1. Description of the problem(s) and the algorithm(s) that you plan to use. This is just a brief recall, to introduce notation and specify which variant(s) of the algorithm(s) you plan to use. In case adapting the algorithm(s) to your problem(s) requires some further mathematical derivation (such as developing an exact line search for your function, when possible, or adapting an algorithm to deal more efficiently with the special structure of your problem), you are supposed to discuss it here with all the necessary mathematical details. Discuss the reasons behind the choices you make (the ones you can make, that is, since several of them will be dictated by the statement of the project and should not be questioned unless you think you have found a serious conceptual flaw in it).
2. Next, a brief recall of the algorithmic properties that you expect to see in the experiments is required. Are there any relevant convergence results for your algorithm(s)? Are the hypotheses of these convergence results (convexity, compactness, differentiability, etc.) satisfied by your problem? If not, what are the “closest” possible results you have available, and why exactly are they not applicable? Do you expect this to be relevant in practice? What about complexity? Each time you use some specific result (say, a convergence theorem), please be sure to report in detail what the assumptions of the result are, what consequences exactly you can derive from them, and the source where you have taken it (down to the number of theorems/page). Discuss in detail why the assumptions are satisfied in your case and why, or which assumptions are not satisfied or you cannot prove they are.
3. Coding the algorithms is a major part of the project, You are expected to implement the algorithm yourself; it should not be a single line of library call. However, you can use the numerical libraries of your language of choice for some of the individual steps: even using existing solvers to tackle specific optimization problems that appear as sub-problems in your algorithms, but only as long as this is explicitly permitted by the project statement. You can (and should) also use existing libraries to compare their results to yours: for instance, you can check if your algorithm is faster or slower than Matlab’s quadprog (or whatever other applicable off-the-shelf software) and if it produces (up to a tolerance) the same objective value. When in doubt if you should use a library, feel free to ask. Your goal for this project is implementing and testing numerical algorithms: software engineering practices such as a full test suite, or pages of production-quality documentation, are not required. That said, well-written and well documented code is appreciated. You are free to use tools such as git to ease your work, if you are familiar with them, but giving us a pointer to the git repository is not the expected way to provide the code (especially as it would be appreciated if the repo was never made public).
4. Next, a brief description of the data you will test your algorithms on is required. The data will typically have to be either picked up from the Internet (repositories of AI/ML datasets, repositories of optimization instances, . . . ), or generated randomly, or a combination of both (such as in taking a dataset having most of the required data and randomly generating the few missing pieces). This is not always trivial: the random generation process can be tweaked to obtain “interesting” properties of the data (what kind of solution can be expected, how well or badly a given approach can be expected to perform, . . . ). These aspects should be described in the report. You are supposed to test the algorithm on a realistic range of examples, in terms of size, structure and/or sparsity: it is typically not OK if your largest example is 10 × 10 (whatever “10” measures). Get a sense of how algorithms scale, and what is the maximum size of problems that you can solve reasonably quickly on an ordinary machine. Using HPC systems or machines otherwise equipped with special features (such as high-end GPU) should not be required, save possibly for SMS++ projects (in which case assistance to get access to the required hardware will be provided). Numerical experiments have two purposes: • Confirm that the algorithms work as expected: how close do they get to the true solution of the problem? How can you check it? Is there a “gap” value that you can check? Do they converge with the rate (linearly, sublinearly, superlinearly, . . . ) that the theory predicts? Does the error decrease monotonically, if it is expected to do so? • Evaluate trade-offs and compare various algorithms: which algorithm is faster? If algorithm A takes fewer iterations than algorithm B, but its iterations are more expensive, which one is the winner? How does this depend on the characteristics of the problem to be solved (size, density, . . . )? Comparison with off-the-shelf software is also welcome (and often useful to check correctness) to assess whether your approach could ever be competitive under the right conditions, and what these are. Setting thresholds and algorithmic parameters is a key and nontrivial aspect. This is one of the “dark secrets” of numerical algorithms: basically any algorithm you can write has parameters that can have a huge impact on performance, and it will misbehave if you do not set them properly. Thus, a minimal testing activity about the effect of these parameters is almost surely needed. A full-scale test a-la hypermetameters optimization in ML is also possible, and welcome, but typically not necessary. Those already familiar with AI/ML techniques should also consider that, even in projects where the underlying model in a ML one (say, a NN, a SVR, . . . ) the properties of interest are fundamentally different from the ones one would be concerned with in a ML setting. **Indeed, ML is interested in learning (accuracy, recall, . . . ), while in this course one is looking at “how close the solution I got is to what I would have liked to get, and how costly was it to get there” Besides, it is common occurrence in ML to do hypermetameters tuning at the same time on model parameters (say, the weight of the regularization term or the topology of the NN) and algorithmic parameters (say, the fixed stepsize); this is unacceptable here, where the model parameters need be fixed (in any reasonable way) so that all the algorithms solve exactly the same problem. Hence, any hypermetameters optimization would need to be properly reasoned. When designing your experiments, and later your graphs and/or tables, you should have these goals in mind. To quote a famous mathematician, “the purpose of computing is insight, not numbers.”**
5. A few plots and/or tables are required to display your results. Have a purpose in mind: as for the choice of experiments, the plots should display the important features of the algorithm(s): convergence speed, gap/accuracy, computational time, . . . Plots should be readable. If 90% of your plot are barely-visible horizontal or vertical lines, look for a better way to display information. Logarithmic scales are usually a good idea to display quantities such gaps or gradient norms that vary by orders of magnitude. In particular, they display well convergence speed, since a sequence with linear convergence (vk+1 ≤ r · vk) should become a(n approximately) straight line. Always keep in mind what is the information that is important for you to show and do your best to show it effectively and efficiently: three pages-long tables of 7pt figures or 10 pages filled of very many small plots are typically neither efficient nor effective at conveying the information to your readers.