Functional

Matteo Caldana

10/03/2023

Warning

To complete some points of the following exercises you need to install the utilities in "Examples Utilities" and the "Extras". If the submodules folders (json and muparser) are empty, you forgot the `--recursive` option when cloning the pacs repo, you can fix it by running `git submodule update --init` in the root folder of the pacs repo.

Exercise 1 - Horner algorithm

1. Implement the `eval()` and `eval_horner()` functions to compute:

$$p_{\text{eval}}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n,$$
$$p_{\text{Horner}}(x) = a_0 + x\,(a_1 + x\,(a_2 + x\,(a_3 + \ldots + x\,(a_{n-1} + x\,a_n)\ldots))).$$

2. Implement an `evaluate_poly()` function by manually looping over the input points.
3. Modify `evaluate_poly()` to use `std::transform`.
4. Implement an `evaluate_poly_parallel()` that makes use of the parallel execution policies of `std::transform` (available since C++17).
5. Convert `eval` and `eval_horner` from function pointers to `std::function`.
6. (*Homework*) Let the user choose from a json parameters file the degree of the polynomial and the discretization interval. The `json` library can be installed by running `./install_PACS.sh` from `${PACS_ROOT}/Extras/json`

NB: the parallel version requires to link against the Intel Threading Building Blocks (`TBB`) library (preprocessor flags `-I${mkTbbInc}`, linker flags `-L${mkTbbLib} -ltbb`).

Exercise 2 - Newton solver

1. Implement a `NewtonSolver` class.
2. Let algorithm parameters be read from the command line using `GetPot`.
3. Write different `main` files that pass functions and derivatives as:
   - 3.1 function pointers;
   - 3.2 (*Homework*) lambda functions;
   - 3.3 `muParser` functions (the `muParser` library can be installed by running `./install_PACS.sh` from `${PACS_ROOT}/Extras/muParser`).
   - 3.4 (*Homework*) Let the user pass the function and the parameters from command line using `GetPot` and `muParser`.
4. Use the solver to solve the equation

$$x^3 + 5x + 3 = 0.$$