MA3227 Numerical Analysis II

Assignment 2

Simon Etter, 2019/2020
Deadline: 18 March 2020, 12.00 (noon)
Total marks: 10

Disclaimer. Do not be discouraged by the length of this assignment sheet. The task descriptions are long, but they only ask you to fill in some very limited blanks in an existing code. I expect that you will spend most of your time figuring out the setting of the tasks. Once you have done that, you can complete the tasks in 30 minutes. You are welcome to discuss with me if you are unsure about what is asked of you.

1 Multigrid in two dimensions [4 marks]

Recall that the two building blocks of the multigrid method for solving $Ax = b$ are the relaxed Jacobi iteration

$$x_{k+1} = x_k + \theta\, D^{-1}\,(b - Ax_k) \tag{1}$$

for eliminating the high-frequency errors, and the coarse grid correction

$$x_1 = x_0 + P\,(P^T AP)^{-1} P^T(b - Ax_0) \tag{2}$$

for eliminating the low-frequency errors. Lecture 11 discussed how to choose the parameters θ and P if $A = -\Delta_n^{(1)}$ is the one-dimensional Laplacian matrix. In particular, we have seen that a suitable choice for P in the coarse grid correction step (2) is

$$P_n^{(1)} = \begin{pmatrix} \frac12 & & & \\ 1 & & & \\ \frac12 & \frac12 & & \\ & 1 & & \\ & \frac12 & \frac12 & \\ & & 1 & \ddots \\ & & \frac12 & \ddots \\ & & & \ddots \end{pmatrix} \in \mathbb{R}^{(2n+1)\times n}.$$

This task will discuss how to choose θ and P if $A = -\Delta_n^{(2)}$ is the two-dimensional Laplacian matrix.

We have seen that the eigenvalues $\lambda_{k_1,k_2}^{(2)}$ and eigenvectors $u_{k_1,k_2}^{(2)}$ of $-\Delta_n^{(2)}$ are given by

$$\lambda_{k_1,k_2}^{(2)} = \lambda_{k_1}^{(1)} + \lambda_{k_2}^{(1)} \qquad \text{and} \qquad u_{k_1,k_2}^{(2)} = u_{k_1}^{(1)} \otimes u_{k_2}^{(1)}$$

where $\lambda_k^{(1)} \in \mathbb{R}$ and $u_k^{(1)} \in \mathbb{R}^n$ are the eigenvalues and -vectors of $-\Delta_n^{(1)}$. It can be shown that the coarse grid correction (2) with $P_n^{(2)} = P_n^{(1)} \otimes P_n^{(1)}$ leads to $(u_{k_1,k_2}^{(2)})^T(x_1 - x) \approx 0$ for $k_1, k_2 \in \{1, \ldots, \frac{n}{2}\}$ regardless of the initial guess x_0. We should therefore choose the θ-parameter in the relaxed Jacobi iteration (1) such that it reduces all error components corresponding to the eigenvalues

$$\lambda_{k_1,k_2} \qquad \text{where} \qquad (k_1, k_2) \in \big\{1, \ldots, n\big\}^2 \setminus \big\{1, \ldots, \tfrac{n}{2}\big\}^2. \tag{3}$$

Your tasks are as follows.

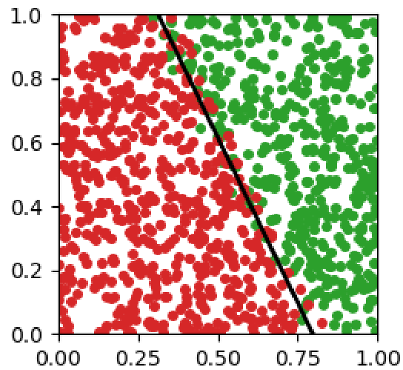
1. Determine the parameter θ_{best} such that the relaxed Jacobi iteration (1) leads to the fastest possible reduction in the error components associated with (3). Determine which of these error components converges the slowest, and determine their rate of convergence $|\hat\lambda_{\text{best}}|$.
2. Set the variable `jacobi_step_length` defined on line 7 in `sheet2_multigrid.jl` to the θ_{best} determined in step 1, and set the variable `jacobi_convergence_rate` on line 8 to $|\hat\lambda_{\text{best}}|$.
3. (unmarked) Run `plot_convergence()`. Does the resulting plot match your expectations? You may also check your answer for step 1 by changing `jacobi_step_length` and verifying that the rate of convergence indeed deteriorates.

Hint. All you need to know about $A = -\Delta_n^{(2)}$ for this task is that the diagonal D is given by $D = 4\,(n+1)^2\, I$ and the eigenvalues $\lambda_k^{(1)}$ of $-\Delta_n^{(1)}$ are monotonically increasing in k and satisfy

$$\lambda_k^{(1)} \in \begin{cases} [0, 2\,(n+1)^2] & \text{if } k \in \{1, \dots, \frac{n}{2}\}, \\ [2\,(n+1)^2, 4\,(n+1)^2] & \text{if } k \in \{\frac{n}{2}, \dots, n\}. \end{cases}$$

2 Gradient descent for classification [6 marks]

The file `sheet2_classification.jl` defines a function `generate_dataset(n)` which returns a vector of points `xs[i]` $\in \mathbb{R}^2$ and a vector of labels `ls[i]` $\in \{0, 1\}$. Plotting these points with different colours for the two labels results in the following picture.



Our aim in this task is to estimate the parameters `t[1]`,`t[2]`,`t[3]` of the line

$$L = \big\{ \mathtt{x} \in \mathbb{R}^2 \mid \mathtt{h(x,t)} = 0 \big\} \quad \text{with} \quad \mathtt{h(x,t) = t[1]*x[1] + t[2]*x[2] + t[3]}$$

which separates the two labels as neatly as possible (L is indicated by the black line in the picture). We do so by determining

$$\mathtt{t} = \arg\min \mathtt{g(xs,ls,t)} \tag{4}$$

where

$$\mathtt{g(xs,ls,t)} = \sum_{\mathtt{i}=1}^{\mathtt{n}} \Big((1 - \mathtt{ls[i]}) \, \log\!\big(1 + \exp\!\big(\mathtt{h(xs[i],t)}\big)\big) + \\ \mathtt{ls[i]} \, \log\!\big(1 + \exp\!\big(\mathtt{-h(xs[i],t)}\big)\big) \Big).$$

The rationale for this function is that

$$\log\big(1 + \exp\big(\texttt{h(x,t)}\big)\big) \approx \begin{cases} \texttt{h(x,t)} & \text{if } \texttt{h(x,t)} \gg 0, \\ 0 & \text{if } \texttt{h(x,t)} \ll 0, \end{cases}$$

and thus the first term in `g(xs,ls,t)` is large if there are points `xs[i]` with `ls[i]` $= 0$ which are far on the side of L where `h(xs[i],t)` > 0, but this term is approximately 0 if we have `h(xs[i],t)` < 0 for all points with `ls[i] = 0`. The same reasoning applies to the second term.

We will solve (4) using the gradient descent method with adaptive step sizes. To this end, you will have to do the following.

1. Complete the function `gradient_descent(g,dg,x,nsteps)` which is to return the ($k =$ `nsteps`)-th element of the sequence

   $$x_0 = \texttt{x}, \qquad x_{k+1} = x_k - \alpha_k \, \texttt{dg}(x_k).$$

   In each step, choose the step size α_k as $\alpha_k = 2^{-\ell}$ where ℓ is the smallest nonnegative integer such that $\texttt{g}(x_{k+1}) \leq \texttt{g}(x_k)$.

   Note that the functions `g(x)` and `dg(x)` passed as arguments to `gradient_descent()` are a function of only a single variable. Have a look at `plot_and_solve()` to see how this function will be used.

   Hint. You can test your code using `test_gradient_descent()`.

2. Complete the function `dg(xs,ls,t)` which provides the gradient of `g(xs,ls,t)` with respect to `t`.

   Hints.

   - Have a look at `g(xs,ls,t)` to see how the sum over the data points can be implemented conveniently. `sum()` also works for vectors, e.g. `sum([[1,2],[3,4]]) -> [4,6]`.
   - Note that the gradient of `h(x,t)` is implemented in `dh(x,t)`.
   - You can test your code using `test_dg()`.

3. (unmarked) Run `plot_and_solve()` and check that it reproduces the above picture.