Instructions & grading scheme

- **DEADLINE for final discussion: Thursday 30.3.2023, 18:00**
- **DEADLINE for contacting TA to schedule the discussion: Tuesday 28.3.2023, 23:59**

You are allowed to work in groups of two. Each group is free to work on the same code/notebook or on individual codes/notebooks. However, the graded discussions of the project will take place individually for each group member.

Every student is responsible for contacting his/her TA to arrange a meeting for the discussion. Requests received after the scheduling deadline will not be admitted.

The discussion will take a maximum of 40 minutes. Please send the Python notebook/module files to the TA before the discussion starts, not doing this will subtract 10 points.

It is imperative to note that failure to meet the deadline for the discussion of each project will result in a failing grade for the project and therefore, the course. In the event of a failed project discussion at the initial attempt, there is an opportunity to rectify it within 48 hours, resulting in a "RESIT" grade rather than a grade in Progress for an "EXAM". However, in exceptional circumstances, such as a case of serious force majeure, the TA may be requested to defer the discussion.

This assignment consists of implementation questions and theory questions. Implementation questions ($\equiv$ all questions not marked 'Theory question') ask you to write python code, while theory questions ask you to interpret the behavior and results of the methods you implement.
For the implementation questions, the following **requirements** enter the grade:

R1. Functionality of the code: the code runs without errors and computes the correct results.

R2. Adequate use of control structures (`if-elif-else` statements, `for` or `while` loops) and functions – this means, apart from functionality, avoiding redundant, unnecessary and repeated code statements or groups of statements that can be coded efficiently with loops and/or functions.

R3. High quality of code:

(a) Descriptive variable and function names make the code easy to read and understand.

(b) Functions clearly structure and organize the code, encapsulating tasks/code segments to improve readability and proneness to errors and to enable code reusability, and break up large sections (so-called "spaghetti code").

(c) The code is well annotated with comments[1].

(d) Avoid inconsistencies in the naming of functions and variables (choose a format and stick to it, e.g., `min_max_search()` vs. `minMaxSearch()`), avoid mixing single and double quotes for strings, and inconsistent indenting of code blocks.

(e) Functions MUST NOT use global variables!

(f) Output formatting: printed output should be easy to read and understand, e.g.:
```
Computing time of this function is:  10.3 secs
```
or
```
N = 10, error = 0.0012343234
```
Plots need to appear with labels and legends.

For each implementation question, the grading scheme is the following:

- 100% of points if the code satisfies all requirements R1, R2, R3.
- 50% of points if the code satisfies requirements R1 and R3, but not R2, or R1 and R2 but not R3.

[1]Helpful comments are ones that help explain code segments the meaning of which is not immediately clear (e.g., from good variable/function naming), or why (as opposed to how) something has to be done. Bad comments are those that state the obvious (e.g., the comment `#loop` before a `for` loop or `#check if x is greater than y` before `if x > y:`).

- 25% of points if the code only satisfies requirement R1.
- 0% of points otherwise. (Failure to follow the instructions of the question also leads to 0 points, for example, solving a different problem or not writing a function when the question explicitly asked to "implement a function".)
- Additionally, for each bug (which leads to a wrong result), 10% of points will be discounted.

If during the discussion the student is unable to explain the code and the obtained results or is unable answer the questions of the TA, 0% of points are awarded for the question.

The discussion serves to provide personalized feedback and is a learning activity, too, not just an assessment. Answers to the project will not be provided, so please use the discussion to learn about your (possible) mistakes.

Assignments

Please take into account that:

- For all complexity calculations, define your elementary operations as addition, division, multiplication and square roots.
- **For question 1 and 2 do NOT use the `sparse` module of SciPy**. Use it only for question 3. For questions 1 and 2 you need to code everything yourself. You can of course re-use code you coded during the previous labs.

1 Python modules (5 points)

In this project you will make use of your functions from Lab 9 for solving sparse tridiagonal linear systems by means of LU decomposition and eigenvalue computation.

- Create a python module with a descriptive name containing all relevant functions from the labs and import your module in the Colab notebook for this project.

2 Measurement denoising (42 points)

Problem setup

When making measurements during a physical experiment (for instance measuring at different positions the velocity of a moving object) one will make measurement errors, noise.

Assume the exact values are given by $g(x)$ for $x \in [-1, 1]$ and and the measurements are done at $N+1$ equidistant points, including the extremes of the interval. Let us call those sampling points $x_0, \ldots, x_N$, where $x_0 = -1$ and $x_N = 1$. Assume the measurements y_i are given by $y_i = g(x_i) + \epsilon_i$, with ϵ_i the measurement noise at x_i. The goal is to remove that noise and recover a smooth function which we hope is closer to $g(x)$ than the measurements.

The values of the smoothed function s_i at x_i are found as the solution of the following optimization problem:

$$\arg \min_{s_0,\ldots,s_N} f(s_0, \ldots, s_N) \tag{1}$$

with

$$f(s_0, \ldots, s_N) = \frac{1}{2} \sum_{k=0}^{N} (y_k - s_k)^2 + \beta \, \frac{1}{2} \sum_{k=1}^{N} (s_k - s_{k-1})^2 \tag{2}$$

and $\beta \geq 0$ a user-defined parameter that balances data fit and the smoothness of the solution.

The aim is now to implement and assess two methods to solve this problem, one direct method solving the optimality condition through a linear system and one iterative method based on the gradient descent method.

Understanding the properties of the problem

The solution of the optimization problem (1) can be found by computing the gradient of (2). It can easily be shown that

$$\begin{aligned}
\frac{\partial f}{\partial s_0} &= -(y_0 - s_0) + \beta(s_0 - s_1), \\
\frac{\partial f}{\partial s_i} &= -(y_i - s_i) + \beta(2s_i - s_{i-1} - s_{i+1}), \qquad i = 1, \ldots, N-1 \\
\frac{\partial f}{\partial s_N} &= -(y_N - s_N) + \beta(s_N - s_{N-1}),
\end{aligned}$$

and ∇f can be written as

$$\nabla f = (I + \beta A)\vec{s} - \vec{y},$$

with I the identity matrix and A a tridiagonal matrix. Since A is symmetric positive semi-definite and $\beta \geq 0$, the Hessian $I + \beta A$ is non-singular and hence there exists a unique vector $\vec{s}$ such that $\nabla f(\vec{s}) = 0$. This vector is the solution of Problem (1).

1. **(2 pts)** Theory question: Determine the matrix A. Verify with your TA before continuing!

Solving the problem directly

2. **(2 pts)** Implement a function that, for a given vector $\vec{x}$ consisting of the sampling points $x_0, \ldots, x_N$ returns $\vec{g}$ and $\vec{y}$, with $g_i = g(x_i)$. Generate $\vec{\epsilon}$ using `0.05*np.random.randn(N+1)` and take $g(x) = p_5(x)$, with $x = [-1, 1]$ and $p_n(x) = \frac{1}{n+1} \sum_{k=0}^{n} x^k$.

3. **(8 pts)** Implement a function that, given $\vec{y}$ (the *data*) and β, returns the vector $\vec{s}$ (the *smoothed data*) that minimizes f. Solve the optimization problem by computing the solution to the linear system $(I + \beta A)\vec{s} = \vec{y}$, making use of the functions you implemented in Lab 9. Make use of the sparsity of the matrix $I + \beta A$ – the full/dense matrix should never be formed! Explain your implementation in the discussion.

   Use the following function skeleton and respect the argument/return value descriptions in the docstring (the text enclosed by the triple quotes):

```python
def smooth(data, beta):
    """Smooth noisy data (y) by means of solving a minimization problem.

    Args:
        data (numpy.array): noisy data to be smoothed (y)
        beta (float): parameter >= 0 that balances fit and smoothing

    Returns:
        numpy array of smoothed data (s)
    """
    # <YOUR CODE>
    return data_smoothed
```

4. **(2 pts)** Plot in one figure the results for the original (noise free) function/polynomial $g(x) = p_5(x)$, the noisy measurements (as data points) and the smoothed function, present the results in the discussion for $N = 1e2$ and $\beta = 10$.

5. **(2 pts)** Show what happens with $s_0, \ldots, s_N$ (both on paper and using your code) when $\beta \to \infty$.

6. **(2 pts)** Show what happens with $s_0, \ldots, s_N$ (both on paper and using your code) when $\beta \to 0$.

7. **(4 pts)** Theory question: determine on paper the order of the time complexity of the algorithm in terms of N. Include ALL the steps involved.

8. **(2 pts)** Theory question: consider as an approximation of the space complexity the memory needed to store the tridiagonal matrix. Compare the space in memory taken by the sparse tridiagonal matrix A with that of its dense counterpart. Assume a general $N \times N$ matrix and indicate the space as a function of N in Big-O notation (and not in actual units of Bytes) including the constant of the leading order term.

Solving the problem iteratively

We will now solve Problem (1) iteratively using the gradient descent method seen in the lecture.

9. **(2 pts)** Theory question: show that the method converges if $\alpha < 2/(1 + \beta\lambda_{max})$, with λ_{max} the largest eigenvalue of A.

10. **(10 pt)** Implement a function that, given $\vec{y}$, β and the number of gradient descent iterations, returns $\vec{s}$ using a gradient descent approach. The function needs to compute λ_{max} using the Power method described/implemented in Lab 7 and then assign a value to α such that the method converges. You can make use of the functions you implemented previously in this project and in the labs. You will for sure need to adapt the function estimating the eigenvalue of a matrix to your current tridiagonal matrix structure. You need to show this explicitly in the discussion.

    Use the following function skeleton and respect the argument/return value descriptions in the docstring:

```python
def smooth_iterative(data, beta, num_iterations):
    """Smooth noisy data (y) by means of solving a minimization problem
    iteratively with the gradient descent method.

    Args:
        data (numpy.array): of the noisy data to be smoothed (y)
        beta (float): parameter >= 0 that balances fit and smoothing
        num_iterations (int): number of gradient descent iterations (>0)

    Returns:
        numpy array of smoothed data (s)
    """
    # <YOUR CODE>
    return data_smoothed
```

11. **(2 pt)** For a large value of β compare, by making a plot, your result of the gradient descent method with your result of the direct method. Repeat for increasing number of iterations of the gradient descent method (choose suitable values for the number of iterations).

12. **(4 pt)** Theory question: determine on paper the order of complexity of the gradient descent algorithm in terms of the number of iterations of the gradient descent method and N, for the particular problem you are dealing with. How does this complexity compare with the complexity of the direct solver? Which method would you choose?

3 Image denoising (30 points)

An image consists of an array of values which are related to colors/intensities. Let us denote the "true" 2D image array $\mathbf{p} \in \mathbb{R}^{N_y \times N_x}$, with N_x and N_y the number of pixels in the horizontal and the vertical direction, i.e., the number of columns (N_x) and rows (N_y).

Let us now call the 2D noisy image $\mathbf{y}_{i,j} = \mathbf{p}_{i,j} + \epsilon_{i,j}$, and the values of the smoothed 2D image array $\mathbf{s}$, which we will find as the solution of the following optimization problem:

$$\arg\min_{\mathbf{s}} f(\mathbf{s}) \tag{3}$$

with

$$f(\mathbf{s}) = \frac{1}{2}\sum_{i=1}^{N_y}\sum_{j=1}^{N_x}(\mathbf{y}_{i,j} - \mathbf{s}_{i,j})^2 + \beta\,\frac{1}{2}\sum_{i=1}^{N_y}\sum_{j=2}^{N_x}(\mathbf{s}_{i,j} - \mathbf{s}_{i,j-1})^2. \tag{4}$$

Note that the smoothing is applied only in the "row" dimension of the matrix (i.e., on the pixels in the x-direction).

By rearranging $\mathbf{y}$ and $\mathbf{s}$ row by row as 1D vectors $\vec{y}$ and $\vec{s}$ of size N_xN_y, the term multiplying β in Equation (4) can be rewritten as

$$\frac{1}{2}\|\mathbf{C}\vec{s}\|^2$$

with $\mathbf{C}$ a block diagonal matrix of dimension $N_xN_y \times N_xN_y$ consisting of N_y matrices B of dimension $N_x \times N_x$. The diagonal block matrices B follow directly from the second term in Equation (4) as upper diagonal matrices given by

$$B = \begin{bmatrix} -1 & 1 & 0 & \dots & \dots & \dots & 0 \\ 0 & -1 & 1 & 0 & & & \vdots \\ & 0 & -1 & 1 & 0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ \vdots & & & 0 & -1 & 1 & 0 \\ & & & & 0 & -1 & 1 \\ 0 & \dots & \dots & \dots & 0 & 0 & 0 \end{bmatrix}.$$

It follows that ∇f can be written as

$$\nabla f = (I + \beta \mathbf{C}^T \mathbf{C})\vec{s} - \vec{y},$$

The Hessian $I + \beta \mathbf{C}^T \mathbf{C}$, a matrix of dimension $N_x N_y \times N_x N_y$, is non-singular and hence there exists a unique vector $\vec{s}$ such that $\nabla f(\vec{s}) = 0$. Rearranging the 1D vector $\vec{s}$ as 2D array $\mathbf{s}$ gives the solution of Problem (3).

1. **(10 pt)** Implement a function that by receiving β, N_x, N_y returns the Hessian of f given by $I + \beta \mathbf{C}^T \mathbf{C}$ (so the hard part here is to implement $\mathbf{C}$). Since N_x, N_y can become a very large number even for a simple image, you will need to work with matrices in sparse format using SciPy's module `sparse`.

   Use the following function skeleton and respect the argument/return value descriptions in the docstring:

```python
def hessian(beta, nx, ny):
    """Compute the Hessian of the image denoising problem, given by
    `H = I + beta C^T C`.

    Args:
        beta (float): parameter >= 0 that balances fit and smoothing
        nx (int): number of pixels in x-direction
        ny (int): number of pixels in y-direction

    Returns:
        H: sparse hessian matrix (scipy.sparse.csr_array)
    """
    # <YOUR CODE>
    return H
```

2. **(10 pt)** In Brightspace the Python script `image_tools.py` (with functions that load an image, add noise to it, plot and save it) as well as the image `sonic.jpg` are available.

   - Upload both to your Google drive. Your drive should already be accessible for Google colab from assignment 1.
   - In your Colab notebook, using the python script as a module, call the `create_noisy_image()` function from the module to create a noisy grayscale version of the image, which is saved as `sonic_gray_noisy.jpg`.
   - Read in the noisy image as done in `image_tools.py`, for instance:

```python
from PIL import Image  # PIL is an image processing module
import numpy as np
# path to file, e.g.
filepath = "/content/drive/My Drive/sonic_gray_noisy.jpg"
image = np.asarray(Image.open(filepath))
```

   - Now, use your `hessian` function from question 3.1 to build the Hessian and solve the linear system $(I+\beta \mathbf{C}^T \mathbf{C})\vec{s} = \vec{y}$ (using the `linalg` submodule of `scipy.sparse`) to find the smoothed image satisfying Equation (3). You can use the numpy functions `flatten` and `reshape` to rearrange the 2D image array $\mathbf{y}$ as 1D vector $\vec{y}$ and the 1D vector $\vec{s}$ as 2D image array $\mathbf{s}$. (Pay attention to whether these functions handle the 2D arrays row-by-row or column-by-column!) Plot the smoothed images (as done in `image_tools.py`) for a few values of β. You should take the value of β such that you can notice the effect of the smoothing, but you can still recognize the character in the image.

3. **(10 pts)** The second term in Eq. (4) smoothes the image along the horizontal direction by "penalizing" the horizontal pixel-to-pixel variation of the intensity values. Consider now the case of *vertical* smoothing. Find the equivalents of Eq. (4) and, as a consequence, B for this case, and repeat the previous assignment 3.2.

Remark: The results should look quite blurred because this simple method for denoising also smoothes the image itself, not just the noise. More advanced methods exist but are out of the scope of this course. Our goal was just to give you fun examples to learn programming, we hope you enjoyed it! A better method is the so-called total variation denoising leading to a more complex optimization problem. You will see this in the course Introduction to Opimization.