Introduction to Numerical Optimization

Assignment 1

March 2021

General Instructions:

- Submissions in pairs only.
- You are free to use either MATLAB or Python for any coding exercise.
- Sections labeled by [**Report**] should be included in your final report, and sections labeled with [**Code**] include a short coding task.
- Sections with no associated label include explanatory text only.
- More details about what should be included in your final report and your submission file are available in sections 3 and 4.
- Due date - April 26th, 2021 (23:59).

1 Analytical and Numerical Differentiation

In this section, we will experiment with analytical and numerical differentiation. Specifically, we will derive and evaluate the analytic expressions for the gradient and Hessian of scalar functions, and compare our findings with the numerical differentiation evaluation results.

1.1 Analytical Differentiation

In this part we will derive and evaluate the analytical expressions for the gradient and Hessian of some given scalar functions.

1.1.1 Gradient and Hessian of f_1 [Report]

Derive the analytical expressions for the gradient and Hessian (with respect to x) of the following scalar function:

$$f_1(x) = \phi(Ax) \tag{1}$$

Where:

- $f_1 : \mathbb{R}^n \to \mathbb{R}$
- $x \in \mathbb{R}^{n \times 1}$
- $A \in \mathbb{R}^{m \times n}$
- $\phi : \mathbb{R}^m \to \mathbb{R}$

You can assume that the gradient and Hessian of ϕ are given.

1.1.2 Gradient and Hessian of f_2 [Report]

Derive the analytical expressions for the gradient and Hessian of the following scalar function:

$$f_2(x) = h(\phi(x)) \tag{2}$$

Where:

- $f_2 : \mathbb{R}^n \to \mathbb{R}$
- $x \in \mathbb{R}^{n \times 1}$
- $\phi : \mathbb{R}^n \to \mathbb{R}$
- $h : \mathbb{R} \to \mathbb{R}$

You can assume that the gradient and Hessian of ϕ and the first and second derivatives of h are given.

1.1.3 Gradient and Hessian of ϕ [Report]

Derive the analytical expressions for the gradient and Hessian of the following scalar function $\phi : \mathbb{R}^3 \to \mathbb{R}$:

$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \sin(x_1 x_2 x_3) \tag{3}$$

1.1.4 First and Second Derivatives of h [Report]

Derive the analytical expressions for first and second derivatives of the following function $h : \mathbb{R} \to \mathbb{R}$:

$$h(x) = \sqrt{(1 + x^2)} \tag{4}$$

1.1.5 Analytical Evaluation [Code]

1. Write a function that receives as input a vector $x \in \mathbb{R}^3$ and a square matrix $A \in \mathbb{R}^{3\times3}$, and returns back the evaluation of the value, gradient and Hessian at the point x of the function f_1 from section 1.1.1, when ϕ is given by section 1.1.3. For evaluating the gradient and Hessian of f_1, use the expressions you have derived in section 1.1.1 and section 1.1.3.

2. Write a function that receives as input a vector $x \in \mathbb{R}^3$, and returns back the evaluation of the value, gradient and Hessian at the point x of the function f_2 from section 1.1.2, when ϕ is given by section 1.1.3 and when h is given by section 1.1.4. For evaluating the gradient and Hessian of f_2, use the expressions you have derived in sections 1.1.1, 1.1.3 and 1.1.4.

Please try to avoid unnecessary computations - that is, design your functions interface to accept additional flags that determine whether the function's value, gradient or Hessian should be evaluated or not.

1.2 Numerical Differentiation

In this part we will evaluate numerically the gradient and Hessian of the scalar functions described in section 1.1 using the method of finite differences. For more information about numerical differentiation, please see the following two Wikipedia pages:

- Numerical Differentiation
- Finite Difference

One of the main reasons to use numerical differentiation in practice is as a debug tool, which helps to verify the correctness of the analytical derivations of derivatives of functions.

1.2.1 Numerical Gradient

Given a scalar function $f : \mathbb{R}^n \to \mathbb{R}$, we can approximate its gradient at a point x by evaluating the ratio df to dx of each component of the gradient ∇f, between two infinitesimally close points to x. This method is called *Finite Differences* using *Central Differences*, and is given as follows:

$$\nabla f\left(x\right) \approx \begin{bmatrix} \frac{f(x+\varepsilon e_1)-f(x-\varepsilon e_1)}{2\varepsilon} \\ \frac{f(x+\varepsilon e_2)-f(x-\varepsilon e_2)}{2\varepsilon} \\ \vdots \\ \frac{f(x+\varepsilon e_n)-f(x-\varepsilon e_n)}{2\varepsilon} \end{bmatrix} \tag{5}$$

Where $x \in \mathbb{R}^n$, e_i is the ith standard basis vector of $\mathbb{R}^n$, and $\varepsilon \in \mathbb{R}$ is infinitesimally small, such that equation (5) becomes an equality when $\varepsilon \longrightarrow 0$.

1.2.2 Numerical Hessian

We can also estimate the Hessian matrix of a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ at a point x by repetitive evaluation of f at nearby points, entry by entry, as explained in Finite Differences for the Hessian. However, if the analytical expression for the gradient ∇f is known, we can exploit it to estimate the Hessian $\nabla^2 f$ at a point x more efficiently as follows:

$$\nabla^2 f^{(i)}(x) \approx \frac{\nabla f(x + \varepsilon e_i) - \nabla f(x - \varepsilon e_i)}{2\varepsilon} \tag{6}$$

Where $\nabla^2 f^{(i)}(x)$ is the ith column of $\nabla^2 f(x)$.

1.2.3 Numerical Evaluation of Gradient and Hessian [Code]

Write a function that receives as input a function reference $f(x, \ldots)$ (Python Example, MATLAB Example), a vector $x \in \mathbb{R}^n$, a scalar $\varepsilon \in \mathbb{R}$ and any other additional required argument (depends on your implementation), and returns back the **numerical** evaluation of the gradient and Hessian at the point x of the input function f, as explained in sections 1.2.1 and 1.2.2.

1.3 Comparison [Code and Report]

In this section, we will compare the difference between the analytical and numerical evaluation of the gradient and Hessian of functions f_1 and f_2.
Draw a random vector $x \in \mathbb{R}^3$ and a random matrix $A \in \mathbb{R}^{3\times3}$, and evaluate the infinity norm of the difference between the numerical and analytical gradient and Hessian of f_1 and f_2 at the point x (given by $||\nabla_x^{\text{analytical}} f_{1,2}(x) - \nabla_x^{\text{numerical}} f_{1,2}(x)||_\infty$ for the gradient, and $||\nabla_x^{2\,\text{analytical}} f_{1,2}(x) - \nabla_x^{2\,\text{numerical}} f_{1,2}(x)||_\infty$ for the Hessian) for values of ε between 2^0 to 2^{-60}. Plot your results on four logarithmic-scaled line-charts (two for f_1 (gradient and Hessian) and two for f_2 (gradient and Hessian)) where the x-axis represents ε and the y-axis represents $||\cdot||_\infty$. **Clarification (25/3/2021)**: Only the y-axis should be logarithmic scaled. The x-axis should be the absolute value of the exponent (0, 1, 2, ..., 60).

2 Singular Value Decomposition (Bonus Section, 0.4 Points to Final Grade)

2.1 Motivation

In computer graphics, among other fields, one of the methods to apply a 2D texture to a 3D mesh is by mapping its triangles onto the 2D euclidean plane without "tearing apart" triangle adjacency (so the texture is rendered continuously on the mesh's surface), such that some desired geometric properties are preserved (e.g. angle preservation, length preservation, area preservation, orientation preservation, and more).

These geometric properties are affected by the Jacobian matrix of the mapping at each mesh triangle. Specifically, it is the singular values of the Jacobian matrix that determine how the mapping distort each mesh triangle when it is mapped onto the 2D domain.

One way to achieve a mapping which renders a texture on the 3D mesh such that some artistic desiderata is satisfied, is by solving an optimization problem which yields a mapping of which the singular values of the Jacobian of each triangle on the mesh satisfy some requirement. Here are a few examples:

- If the (two) singular values are identical ($\sigma_1 = \sigma_2$), we would get a conformal mapping which preserves angles, but not necessary preserves length.
- If the two singular values equal 1 ($\sigma_1 = \sigma_2 = 1$), we would get an isometric mapping, which preserves length.
- If the product of the two singular values is 1 ($_1 \cdot \sigma_2 = 1$), we would get an Authalic mapping, which preserves area.

Since the Jacobian matrix is affected directly from the coordinates (u, v) of the vertices that make up the mapped triangle on the 2D domain, we would like to know how a small movement of a triangle vertex will affect the singular values of the Jacobian of the triangle, so we can formulate an appropriate optimization problem that suits our requirements. In other words, we would like to calculate the gradient of the function $f_i(J) = \sigma_i$ which outputs the ith singular value of a matrix J.

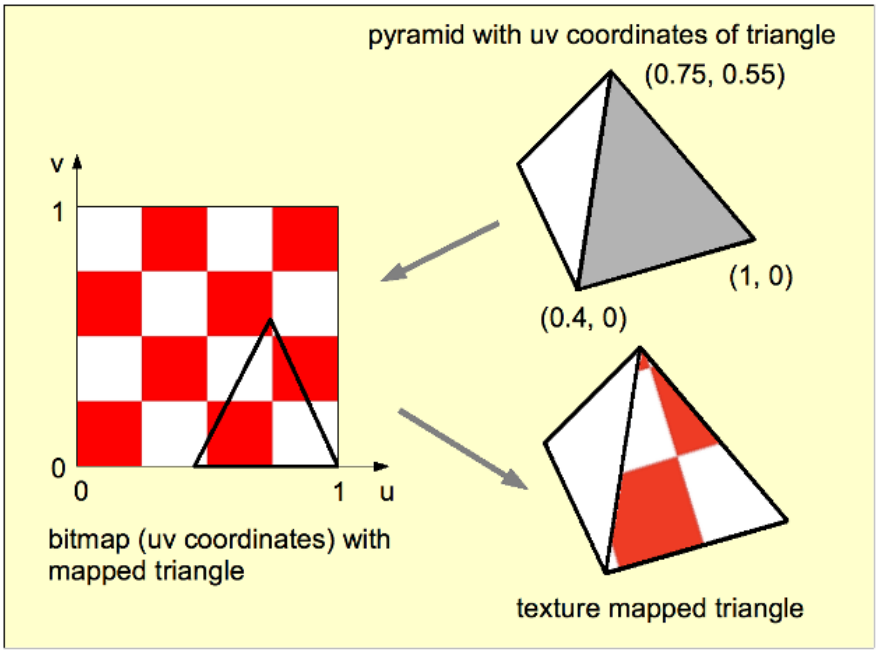

Figure 1: Texture mapping

2.2 Singular Value Gradient [Report]

Let $A \in \mathbb{R}^{m \times n}$ be a real matrix. The singular value decomposition of A is given by $A = USV^T$ where:

- $U \in \mathbb{R}^{m \times m}$ is an orthonormal matrix.
- $S \in \mathbb{R}^{m \times n}$ is a rectangular diagonal matrix, of which the ith diagonal entry is the ith singular value σ_i.
- $V \in \mathbb{R}^{n \times n}$ is an orthonormal matrix.

Calculate the gradient of $f_i(A) = \sigma_i$.

3 What Should Be Included In Your Written Report?

1. Detailed derivations of sections 1.1.1, 1.1.2, 1.1.3 and 1.1.4.
2. Chart plots of section 1.3. Please provide an explanation for the behavior of each chart, and specify for which ε you get an optimal result.
3. Full solution for section 2.2 (Not mandatory, bonus).

4 Assignment Submission

1. Please zip your report's PDF file along with the source code you have written in sections 1.1.5, 1.2.3 and 1.3 into a file named **hw1.zip**, and submit your file through Coursera.
2. Please make a video recording with oral explanations to each of the following items:
   - Explanation of each transition in the derivation of sections 1.1.1, 1.1.2, 1.1.3 and 1.1.4.
   - Documentation to the code that you have written in section 1.1.5.
   - Documentation to the code that you have written in section 1.2.3.
   - Documentation to the code that you have written in section 1.3.
   - Explanation of each transition in the derivation of section 2.2 (Not mandatory, bonus).

You are free to record each video on your own pace and style.

Please zip all your videos into a file named **videos.zip**, and upload it to Coursera. Please note, each individual has to record his own set of videos, and upload them separately to Coursera.