Computational Project 2

Numerical Analysis

Lado Turmanidze

May 24, 2024

Summary

The provided Python code implements several numerical interpolation methods, including Bézier curves, Catmull-Rom splines, Lagrange interpolation, and Radial Basis Function (RBF) for restoring corrupted digital images. The code is divided into two main parts: audio interpolation (Problem 1) and image restoration (Problem 2).

Problem 1

The 'Problem 1.py' file is responsible for audio interpolation. It prompts the user to enter a small text, which is then converted to speech using the 'gTTS' library. The resulting audio file is saved as a WAV file, and the audio data is processed and divided into segments.

First, to test how well audio interpolates, I half the discrete data by only leaving data which have odd numbered indices. If audio is good after amount of signals has been halved, then interpolation is good. Then, for each segment, the code applies three different interpolation methods: Bézier curves, Catmull-Rom splines, and Lagrange interpolation. The interpolated segments are then combined and saved as separate WAV files.

The 'mathematics.py' file contains the implementations of the interpolation methods and helper functions used in 'Problem 1.py'.

For more technical details:

As I mentioned above, first, I half the signal and then, for text-to-speech conversion, I use Google Translate's gTTS (Google Text

To Speech). For fun, I have set the language to Japanese, but it can be changed as desired. This conversion generates an MP3 file, which I convert to a WAV file for easier manipulation. The bit rate and data are extracted from the WAV file, and we only need the bit rate after creating the new audio files. The audio data, which consists of integers, is divided into 882 segments(Since we were asked to divide data in 20 millisecond intervals).

Afterwards, I use three different interpolation methods from 'mathematics.py', namely Bézier curves, Catmull-Rom splines, and Lagrange interpolation with Chebyshev nodes. These methods operate on each of the 882 intervals, and the new results are flattened. Bézier curves and Catmull-Rom splines perform well at approximating the polynomial function to the actual signal, but Lagrange Interpolation with Chebyshev nodes performs poorly (although the code functions correctly). For demonstration purposes and thanks to matplotlib, I also graph each of the interpolated curves by a custom made(more fancy, as it has black background and a grid) function for a certain segment. The new WAV files can also be played.

Problem 2

The 'Problem 2.py' file focuses on image restoration using RBF interpolation. It loads a 32 by 32 image and randomly removes a specified percentage of pixels (10%, 25%, 50%, and 75%). I chose 32 by 32 images because Default code should test all 3 method and all 4 percentage levels. The modified images are saved with the corresponding percentages in their filenames.

The code then attempts to restore the modified images using three different radial basis functions: thin plate spline $\varphi(r) = r^2 \ln(r)$, Gaussian $\varphi(r) = e^{-(\epsilon r)^2}$ and Inverse Quadratic $\varphi(r) = \frac{1}{1+(\epsilon r)^2}$. It does this by first computing weights vector w by multiplying radial basis interpolation matrix with non-removed coordinates for given φ function and then, for each non-removed coordinate, calculating function approximation $y(x) = \sum_{i=1}^{N} w_i \varphi(\|x - x_i\|)$. The restored images are saved with the corresponding radial basis function and the percentage of removed pixels in their filenames.

'mathematics.py' file contains the implementations of the RBF interpolation methods and helper functions used in 'Problem 2.py'.

Images

(a) Original image (b) Gojo 32

Figure 1: Original image and 32x32 version

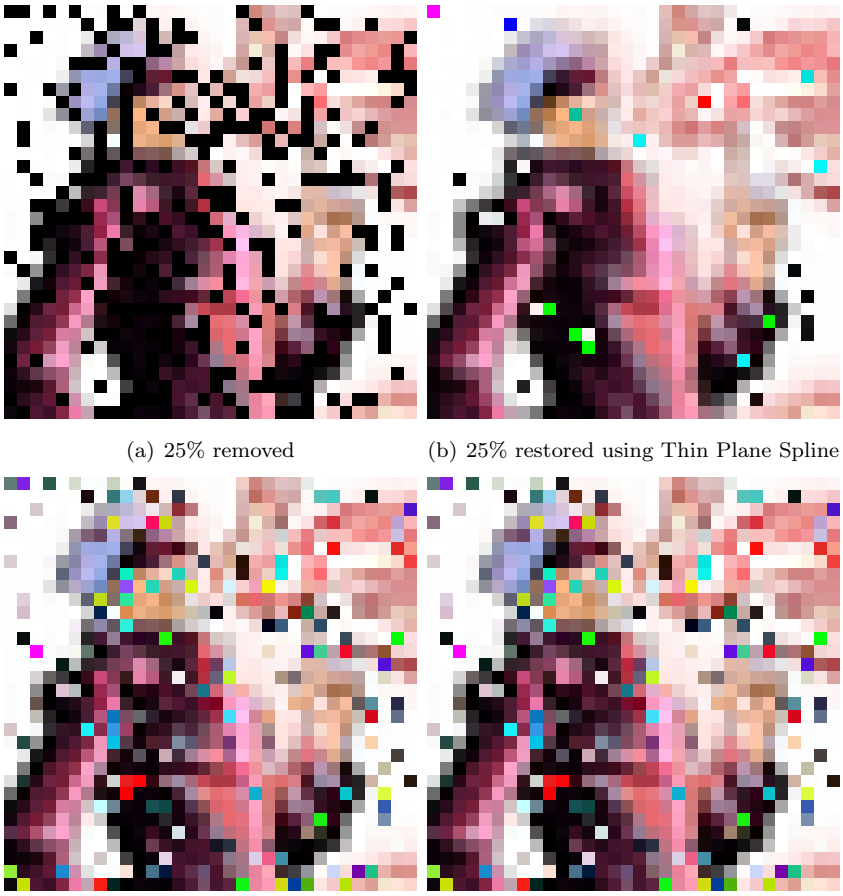
(a) 10% removed

(b) 10% restored using Thin Plane Spline

(c) 10% restored using Gaussian

(d) 10% restored using Inverse Quadratic

Figure 2: 10% removed and restored images



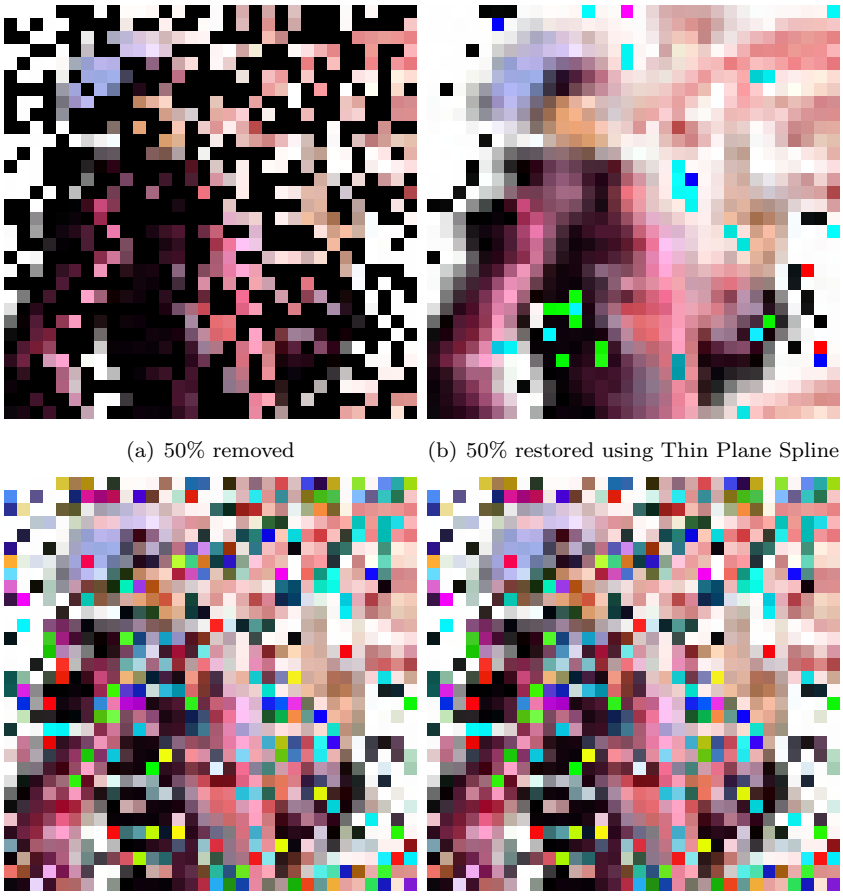
(a) 25% removed

(b) 25% restored using Thin Plane Spline

(c) 25% restored using Gaussian

(d) 25% restored using Inverse Quadratic

Figure 3: 25% removed and restored images



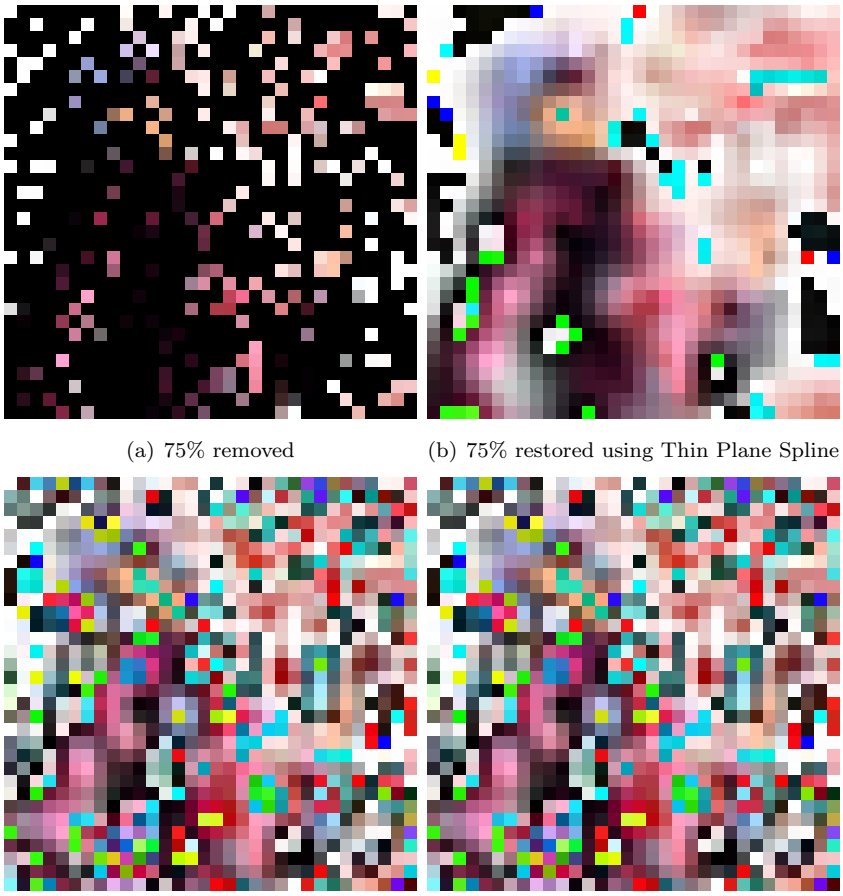
(a) 50% removed

(b) 50% restored using Thin Plane Spline

(c) 50% restored using Gaussian

(d) 50% restored using Inverse Quadratic

Figure 4: 50% removed and restored images



(a) 75% removed

(b) 75% restored using Thin Plane Spline

(c) 75% restored using Gaussian

(d) 75% restored using Inverse Quadratic

Figure 5: 75% removed and restored images