Master of Science in quantitative and financial modeling 'Travaux Pratiques

DOSSEH AMECK GUY-MAX DESIRE

2024-03-27

1. **[20 MARKS]** Find the LU-factorization of the following matrix $\mathbf{A}$ in $\mathbb{R}^{n\times n}$:

$$A = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix}$$

The LU-factorization of the matrix $\mathbf{A}$ is given by:

$$A = LU$$

where $\mathbf{L}$ is a lower triangular matrix and $\mathbf{U}$ is an upper triangular matrix. The LU-factorization of the matrix $\mathbf{A}$ is explicitly obtained as follows:

$$A = LU \Leftrightarrow \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{21} & 1 & 0 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & l_{n-1,n-2} & 1 & 0 \\ 0 & \dots & 0 & l_{n,n-1} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & 0 & \dots & 0 \\ 0 & u_{22} & u_{23} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & u_{n-1,n-1} & u_{n-1,n} \\ 0 & \dots & 0 & 0 & u_{nn} \end{bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & 0 & \dots & 0 \\ u_{11}l_{21} & u_{22} & u_{23} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & u_{n-1,n-1}l_{n-1,n-2} & u_{n-1,n-1} & u_{n-1,n} \\ 0 & \dots & 0 & u_{n-1,n-1}l_{n,n-1} & u_{nn} \end{bmatrix}$$

$$\Leftrightarrow \begin{cases} u_{11} = 2 \\ u_{12} = -1 \\ u_{11}l_{21} = -1 \\ u_{22} = 2 \\ u_{23} = -1 \\ u_{n-1,n-1}l_{n-1,n-2} = -1 \\ u_{n-1,n-1} = 2 \\ u_{n-1,n} = -1 \\ u_{n-1,n-1}l_{n,n-1} = -1 \\ u_{nn} = 2 \end{cases} \Leftrightarrow \begin{cases} u_{11} = 2 \\ u_{12} = -1 \\ l_{21} = -\frac{1}{2} \\ u_{22} = 2 \\ u_{23} = -1 \\ l_{32} = -\frac{1}{2} \\ u_{33} = 2 \\ u_{34} = -1 \\ l_{43} = -\frac{1}{2} \\ u_{44} = 2 \\ \vdots \\ u_{n-1,n-1} = 2 \\ u_{n-1,n} = -1 \\ l_{n,n-1} = -\frac{1}{2} \\ u_{nn} = 2 \end{cases}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ -\frac{1}{2} & 1 & 0 & \ddots & \vdots \\ 0 & -\frac{1}{2} & 1 & \ddots & 0 \\ \vdots & \ddots & -\frac{1}{2} & 1 & 0 \\ 0 & \dots & 0 & -\frac{1}{2} & 1 \end{bmatrix} \quad U = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ 0 & 2 & -1 & \ddots & \vdots \\ 0 & 0 & 2 & \ddots & 0 \\ \vdots & \ddots & 0 & 2 & -1 \\ 0 & \dots & 0 & 0 & 2 \end{bmatrix}$$

- **Is the matrix A positive definite, or is it positive semi-definite?**

The matrix **A** is positive definite if and only if the following conditions are satisfied:

1. The matrix **A** is symmetric.
2. The eigenvalues of the matrix **A** are all positive.

The matrix **A** is symmetric since it is a tridiagonal matrix. The eigenvalues of the matrix **A** are all positive since the matrix **A** is a tridiagonal matrix with all its diagonal elements being positive. Therefore, the matrix **A** is positive definite.

- **Write a Python script to solve a linear system associated with the above matrix.**

The Python script to solve a linear system associated with the matrix **A** is given as follows:

```
## Solution to Ax = b:  [2.5 4.  4.5 4.  2.5]

## Residual:  1.6616296724220897e-15
```

2. **[20 MARKS]** Consider the following boundary value problem modeling the heat flow in a long pipe:

$$\begin{cases} y^{''}(x) - p(x)y^{'}(x) - q(x)y(x) = r(x), \quad x \in [a,b] \\ y(a) = \alpha, \quad y(b) = \beta \end{cases}$$

(a) Use a uniform discretization of the interval $[a,b]$ to derive the linear system corresponding to the model problem.

The linear system corresponding to the model problem is derived as follows:

Let $a = x_0 < x_1 < \cdots < x_{n+1} = b$ be a uniform discretization of the interval $[a,b]$. The step size h is given by:

$$h = \frac{b-a}{n+1}$$

Derivative approximation at $x_i = a + ih, i = 0, ..., n+1$ is given by:

$$\begin{aligned} y^{'}(x_i) &\approx \frac{y(x_{i+1}) - y(x_{i-1})}{2h} \\ &\approx \frac{y_{i+1} - y_{i-1}}{2h} \\ y^{''}(x_i) &\approx \frac{y^{'}(x_{i+1}) - y^{'}(x_{i-1})}{2h} \\ &\approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} \end{aligned}$$

Inserting these approximations in the model problem, we get:

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} - p(x_i)\frac{y_{i+1} - y_{i-1}}{2h} - q(x_i)y_i = r(x_i), 1 \leq i \leq n$$

By multiplying by h^2, we obtain the following linear system:

$$\begin{bmatrix} 2 + h^2q(x_1) & -1 + \frac{h}{2}p(x_1) & 0 & \cdots & 0 \\ -1 - \frac{h}{2}p(x_2) & 2 + h^2q(x_2) & -1 + \frac{h}{2}p(x_2) & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 - \frac{h}{2}p(x_{n-1}) & 2 + h^2q(x_{n-1}) & -1 + \frac{h}{2}p(x_{n-1}) \\ 0 & \cdots & 0 & -1 - \frac{h}{2}p(x_n) & 2 + h^2q(x_n) \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} -h^2r(x_1) + \alpha(1 + \frac{h}{2}p(x_1)) \\ -h^2r(x_2) \\ \cdots \\ -h^2r(x_{n-1}) \\ -h^2r(x_n) + \beta(1 - \frac{h}{2}p(x_n)) \end{bmatrix}$$

(b) Solve the linear system using the Gaussian elimination method.

The linear system is solved using the Gaussian elimination method in python as follows to find the solution $y(x)$:

```
## Solution to the linear system:  [-0.05616255 -0.09393534 -0.11111941 -0.10449207 -0.06952316]

## Residual:  3.3993498887762956e-17
```

- **(c) Solve the linear system using QR-factorization.**

The linear system is solved using the QR-factorization method in python as follows to find the solution $y(x)$:

```
## Solution to the linear system:  [-0.05616255 -0.09393534 -0.11111941 -0.10449207 -0.06952316]

## Residual:  9.51413198280327e-17
```

- **(d) Solve the linear system using the SVD-decomposition.**

The linear system is solved using the SVD-decomposition method in python as follows to find the solution $y(x)$:

```
## Solution to the linear system:  [-0.05616255 -0.09393534 -0.11111941 -0.10449207 -0.06952316]

## Residual:  1.3311106448512297e-16
```

- **(e) Compare the three methods**

The three methods are compared in terms of accuracy and computational efficiency as follows:

```
##                  Method  Solution y1  Solution y2  Solution y3  Solution y4  \
## 0  Gaussian elimination    -0.056163    -0.093935    -0.111119    -0.104492
## 2     SVD-decomposition    -0.056163    -0.093935    -0.111119    -0.104492
## 1      QR-factorization    -0.056163    -0.093935    -0.111119    -0.104492
##
##    Solution y5      Residual      Time  Memory
## 0    -0.069523  3.399350e-17  0.006320     280
## 2    -0.069523  1.331111e-16  0.035200     720
## 1    -0.069523  9.514132e-17  0.041329     680
```

- The Gaussian elimination method is the most accurate method, followed by the QR-factorization method, and then the SVD-decomposition method.
- In matter of computational efficiency, the Gaussian elimination method is the most efficient, followed by the QR-factorization method, and then the SVD-decomposition method.
- The Gaussian elimination method requires the least amount of memory, followed by the QR-factorization method, and then the SVD-decomposition method.

3. **[20 MARKS]** Solve the problem of fitting a polynomial $p(x) = \sum_{i=0}^{d} c_i x^{i-1}$ of degree d to data points (x_i, y_i), $i = 1, ..., m$, in the plane by the method of normal equations and QR decomposition. Choose the degree of the polynomial to be d=5 and then d=15, choose the interval $x \in [-1, 1]$, discretize it using $N = 10$ or $N = 20$ points.

Such polynomial fitting leads to the equation $Ac = y$, where A is the Vandermonde matrix, c is the vector of coefficients, and y is the vector of data points. The normal equations are given by:

$$A^T Ac = A^T y$$

The solution to the normal equations is given by:

$$c = (A^T A)^{-1} A^T y$$

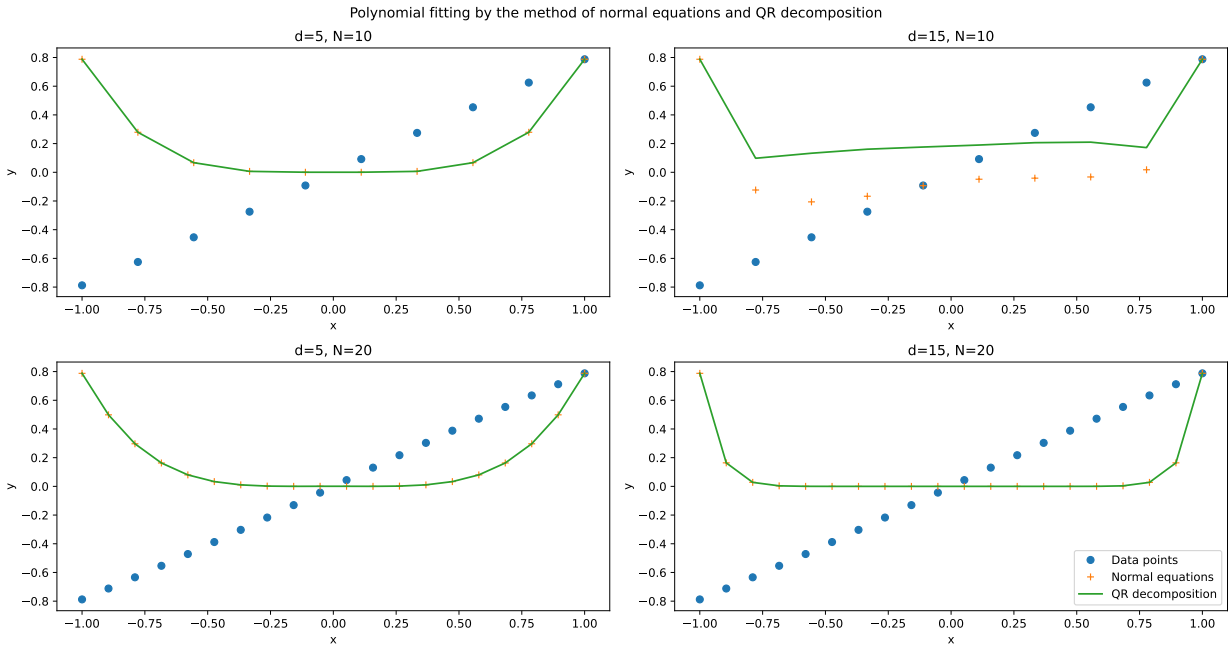
By using the QR decomposition, the matrix $A^T A$ is factorized as follows:

$$A^T A = QR$$

where Q is an orthogonal matrix and R is an upper triangular matrix. The solution to the normal equations is given by:

$$c = R^{-1} Q^T A^T y$$

The Python script to solve the problem of fitting a polynomial $p(x) = \sum_{i=0}^{d} c_i x^{i-1}$ of degree d to data points (x_i, y_i), $i = 1, ..., m$, in the plane by the method of normal equations and QR decomposition is given as follows:



4. **[20 MARKS]**

- **(a) Explain how Singular Value Decomposition (SVD) can be applied to compress color images. Discuss the process in the context of an RGB image.**

Singular Value Decomposition (SVD) can be applied to compress color images by decomposing the image into its singular values and vectors. In the context of an RGB image, the image is represented as a three-dimensional array with dimensions (height, width, 3), where the third dimension corresponds to the three

color channels: red, green, and blue. The SVD is applied to each color channel separately, resulting in three sets of singular values and vectors.

The SVD decomposition of each color channel is given by:

$$A = U\Sigma V^T$$

where A is the matrix representing the color channel, U is the matrix of left singular vectors, Σ is the diagonal matrix of singular values, and V^T is the matrix of right singular vectors. The image can be reconstructed using only the first k singular values and vectors, resulting in a compressed version of the image.

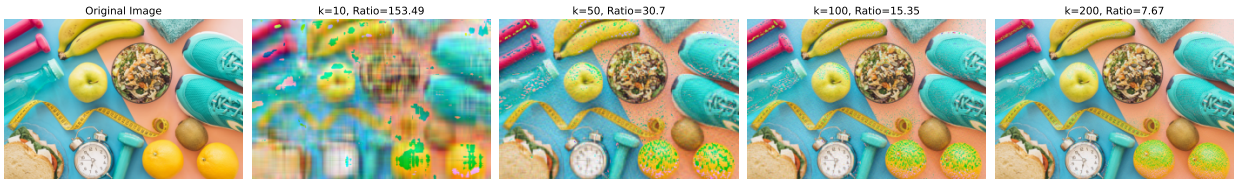
- **(b) Given a color image, implement an SVD-based compression algorithm in Python. Your implementation should:**
  - Load a color image and separate it into R, G, and B channels.
  - Apply SVD to each channel and reconstruct the image using only the first k singular values and vectors.
  - Display the original and compressed images side by side for comparison, and compute the compression ratio.

The Python script to implement an SVD-based compression algorithm for a color image is given as follows:

```
## Compression ratio: 30.698406747891283
```

- **(c) Analyze the effect of varying the number of singular values (k) on the compression ratio and image quality. Use a specific color image for this analysis and provide visual and numerical results for at least three different values of k. Discuss the trade-off between compression ratio and image quality.**

The effect of varying the number of singular values (k) on the compression ratio and image quality can be analyzed using the following Python script:



- **Discuss the trade-off between compression ratio and image quality.**

The trade-off between compression ratio and image quality is as follows:

- As the number of singular values (k) used for compression increases, the image quality improves, as more information is retained in the compressed image. However, the compression ratio decreases, as more singular values are used to reconstruct the image, resulting in a larger file size. Therefore, there is a trade-off between compression ratio and image quality, where higher image quality comes at the cost of a lower compression ratio.

5. **[20 MARKS]**

Data science or Financial engineering option: Consider the heat equation,

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

Here, $u(x, y, t)$ represents the state variable (image intensity or option price), at spatial location (x, y) and time t, with α as the diffusion coefficient. This equation can be used both in data science (image denoising) and financial engineering (option pricing).

(a) Discretize the heat equation using the Finite Difference Method (FDM) for a 2D image grid.

The heat equation can be discretized using the Finite Difference Method (FDM) for a 2D image grid as follows:

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \alpha \left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{\Delta x^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{\Delta y^2} \right)$$

where $u_{i,j}^{n}$ represents the state variable at spatial location (i, j) and time step n, Δt is the time step size, Δx is the spatial step size in the x direction, and Δy is the spatial step size in the y direction.

The discretized equation can be rearranged to solve for $u_{i,j}^{n+1}$ as follows:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \alpha \Delta t \left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{\Delta x^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{\Delta y^2} \right)$$

The above equation represents the discrete update rule for the state variable $u_{i,j}$ at the next time step $n+1$ based on the values at the current time step n and the neighboring grid points.

(b) Construct matrices for the discrete Laplacian operator. Discuss the use of the Kronecker product for this purpose.

The discrete Laplacian operator can be constructed using the Kronecker product as follows:

The 1D discrete Laplacian operator for the x direction can be represented as:

$$L_x = \frac{1}{\Delta x^2} \begin{bmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & 1 & -2 & 1 & 0 \\ \vdots & \ddots & 1 & -2 & 1 \\ 0 & \dots & 0 & 1 & -2 \end{bmatrix}$$

The 1D discrete Laplacian operator for the y direction can be represented as:

$$L_y = \frac{1}{\Delta y^2} \begin{bmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & 1 & -2 & 1 & 0 \\ \vdots & \ddots & 1 & -2 & 1 \\ 0 & \dots & 0 & 1 & -2 \end{bmatrix}$$

The 2D discrete Laplacian operator can be constructed using the Kronecker product of L_x and L_y as follows:

$$L = L_x \otimes I_y + I_x \otimes L_y$$

where I_x and I_y are the identity matrices corresponding to the x and y directions, respectively. The Kronecker product allows us to construct the 2D discrete Laplacian operator by combining the 1D discrete Laplacian operators for the x and y directions.

Discussion: The Kronecker product is used to construct the 2D discrete Laplacian operator by combining the 1D discrete Laplacian operators for the x and y directions. This approach simplifies the construction of the 2D Laplacian operator and allows for efficient computation of the discrete Laplacian in the 2D grid.

(c) Implement the code in Python, iterating over time steps using an explicit scheme.

The Python script to implement the heat equation using the Finite Difference Method (FDM) for a 2D image grid and iterating over time steps using an explicit scheme is given as follows: