Numerical Analysis using Maple and Matlab

Dr. Seongjai Kim
Professor of Mathematics

Department of Mathematics and Statistics
Mississippi State University
Mississippi State, MS 39762

skim@math.msstate.edu

Contents

MA-4313/6313: Numerical Analysis I

MA-4323/6323: Numerical Analysis II

7. Iterative Algebraic Solvers

In This Chapter:

Topics	Applications/Properties
Norms of Vectors and Matrices	Estimation of error bounds
Eigenvalues and Eigenvectors	
Spectral radius	Convergence $\Leftrightarrow \rho(\text{Iteration Matrix}) < 1$
Iterative Algebraic Methods	Regular splitting
Jacobi method	
Gauss-Seidel method	
SOR methods	
Convergence analysis	Graph theory
Conjugate Gradient Method	Symmetric positive definite systems

For $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, we will consider iterative methods for an approximate solution of

$$Ax = b \tag{1}$$

Iterative methods we will consider can be formulated as

$$x^k = x^{k-1} + G\left(b - A x^{k-1}\right) \tag{2}$$

where $b - A x^{k-1} = r^{k-1}$ is the $(k-1)$th residual and G is an operator which can be either a scalar or a matrix.

Maple built-in command:
with(*Student*[*NumericalAnalysis*]) :
IterativeApproximate(*A*, *b*, *opts*)

7.1. Norms of Vectors and Matrices

Vectors

A real ***n*-dimensional vector x** is an ordered set of *n* real numbers and is usually written in the coordinate form

$$\mathbf{x} = \left(x_1, x_2, \cdots, x_n\right)^T = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

Definitions: Let **x** and **y** be *n*-dimensional vectors.

- **Liner combination**:
  $c\,\mathbf{x} + d\,\mathbf{y} = \left(c\,x_1 + d\,y_1, c\,x_2 + d\,y_2, \cdots, c\,x_n + d\,y_n\right)^T$
- **Norm (length)**: $\|\mathbf{x}\| = \left(x_1^2 + x_2^2 + \cdots + x_n^2\right)^{1/2}$, which is often referred to as the **Euclidean norm**, or **Euclidean ℓ_2 norm.**
- **Distance**: $\|\mathbf{x} - \mathbf{y}\| = \left(\left(x_1 - y_1\right)^2 + \left(x_2 - y_2\right)^2 + \cdots + \left(x_n - y_n\right)^2\right)^{1/2}$
- **Dot product**: $\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$. Thus $\mathbf{x} \cdot \mathbf{x} = \|\mathbf{x}\|^2$.

Let θ be the angle between the vectors **x** and **y**. Then,

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \ \|\mathbf{y}\| \cos(\theta).$$

Definition: On a vector space V, a **norm** is a function $\| \bullet \|$ from V to the set of nonnegative real numbers that obeys the following three postulates:

- $\| x \| > 0$, if $x \neq 0, x \in V$
- $\| \lambda x \| = |\lambda| \ \| x \|$, if $\lambda \in \mathbb{R}, x \in V$
- $\| x + y \| \leq \| x \| + \| y \|$, if $x, y \in V$ (triangle inequality)

Examples of norms:

Example: Let $x = \left(x_1, x_2, \bullet\bullet\bullet, x_n \right)^T$.

Euclidean ℓ_2-norm: $\| x \|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}$

The ℓ_1**-norm**: $\| x \|_1 = \sum_{i=1}^{n} |x_i|$

The ℓ_∞**-norm**: $\| x \|_\infty = \max_{1 \leq i \leq n} |x_i|$

Theorem (Cauchy-Schwarz Inequality):

$$|\mathbf{x} \bullet \mathbf{y}| = \left| \sum_{i=1}^{n} x_i y_i \right| \leq \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2} \left(\sum_{i=1}^{n} y_i^2 \right)^{1/2} = \| \mathbf{x} \| \ \| \mathbf{y} \|$$

Theorem: The sequence of vectors $\left\{ \mathbf{x}^{(k)} \right\}$ converges to $\mathbf{x}$ in $\mathbb{R}^n$ with respect to ℓ_∞ norm if and only if $\lim_{n \to \infty} x_i^{(k)} = x_i$ for each $i = 1, 2, \bullet\bullet\bullet, n$.

Example: $\mathbf{x}^{(k)} = \left(1, 2 + \frac{1}{k}, \frac{3}{k^2}, e^{-k} \sin k \right)^T \to (1, 2, 0, 0)^T$ with respect to ℓ_∞ norm.

Theorem: For each $x \in \mathbb{R}^n$,

$$\| \mathbf{x} \|_\infty \le \| \mathbf{x} \|_2 \le \sqrt{n} \ \| \mathbf{x} \|_\infty$$

Proof.

The first inequality comes from

$$\| \mathbf{x} \|_\infty^2 = x_j^2 \le \sum_{i=1}^{n} x_i^2 = \| \mathbf{x} \|_2^2 \text{ for some } j.$$

On the other hand,

$$\| \mathbf{x} \|_2^2 = \sum_{i=1}^{n} x_i^2 \le n\, x_j^2 = n \ \| \mathbf{x} \|_\infty^2$$

and therefore the second inequality follows.

Matrix Norms:

Definition: If a vector norm $\| \cdot \|$ has been specified, the **matrix norm** subordinate to (associated with) it is defined by

$$\| A \| = \sup\{ \| Ax \| : x \in \mathbb{R}^n, \| x \| = 1 \}$$

It is equivalent to

$$\| A \| = \sup_{x \ne 0} \frac{\| Ax \|}{\| x \|}, \quad x \in \mathbb{R}^n$$

Such a norm is called the **natural (induced) matrix norm**.

Matrix norms:

1. $\| A + B \| \le \| A \| + \| B \|$
2. $\| AB \| \le \| A \| \, \| B \|$
3. $\| Ax \| \le \| A \| \, \| x \|$
4. $\| A \|_2 = \sqrt{\rho(A^T A)}$, where $\rho(A^T A)$ denotes the spectral radius of $A^T A$.
5. $\| A \|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|$

6. $\|A\|_{\infty} = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$

Definition: A **condition number** of a matrix A is the number

$$\kappa(A) = \|A\| \cdot \left\|A^{-1}\right\|$$

Example: Let $A := \begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & -1 \\ 1 & -2 & 1 \end{bmatrix}$:

where $A^{\%T}.A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 17 & -7 \\ 0 & -7 & 3 \end{bmatrix}$ and $A^{-1} = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ -\frac{1}{2} & 1 & \frac{1}{2} \\ -\frac{3}{2} & 2 & \frac{3}{2} \end{bmatrix}$

1. Find $\|A\|_1$, $\|A\|_2$, and $\|A\|_{\infty}$.
2. Find the ℓ_1-condition number.

Theorem on Neumann Series: If A is an $n \times n$ matrix such that $\|A\| < 1$ for any subordinate matrix norm, then $I - A$ is invertible and

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k .$$

7.2. Eigenvalues and Eigenvectors

Definition: An **eigenvector** of an $n \times n$ matrix A is a nonzero vector $\mathbf{x}$ such that

$$A\,\mathbf{x} = \lambda\,\mathbf{x},$$

for some scalar λ. The scalar λ is called an **eigenvalue** of A corresponding to $\mathbf{x}$.

Hence, the eigenvector is a nontrivial solution of $(A - \lambda I)\mathbf{x} = 0$, which implies that $(A - \lambda I)$ is singular. The eigenvalues of A are solutions of

$$det(A - \lambda I) = 0.$$

Claim: λ is an eigenvalue of A if and only if $det(A - \lambda I) = 0$.

Example: Find eigenvalues and eigenvectors of $A := \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$:

Example: Determine the eigenvalues and eigenvectors for the matrix

$$A := \begin{bmatrix} 2 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & -1 & 4 \end{bmatrix} :$$

Solution:

$$with(LinearAlgebra) : \ Eigenvectors(A) = \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix}, \begin{bmatrix} -2 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$p := CharacteristicPolynomial(A, \lambda); \ factor(p)$$

$$-12 + \lambda^3 - 7\lambda^2 + 16\lambda$$

$$(\lambda - 3)(\lambda - 2)^2 \qquad \textbf{(1.1)}$$

Now, let us find them by using pencils.

Spectral Radius

Definition: The spectral radius $\rho(A)$ of a matrix A is defined by

$$\rho(A) = \max |\lambda|, \text{ where } \lambda \text{ is an eigenvalue of } A.$$

Theorem: If $A \in \mathbb{R}^{n \times n}$, then

a. $\|A\|_2 = \sqrt{\rho(A^T A)}$

b. $\rho(A) \leq \|A\|$, for any natural matrix norm $\|\cdot\|$.

Proof.

The proof of part (a) requires more advanced matrix algebra.

For part (b), let λ be an eigenvalue of A with its corresponding eigenvector x with $\|x\| = 1$. Then

$$|\lambda| = |\lambda| \, \|x\| = \|\lambda x\| = \|Ax\| \leq \|A\| \, \|x\| = \|A\|$$

Thus, the assertion follows.

Convergent Matrices:

Definition: A matrix $A \in \mathbb{R}^{n \times n}$ is **convergent** if

$$\lim_{k \to \infty} \left(A^k\right)_{ij} = 0 \quad \text{for each } 1 \le i, j \le n.$$

Example: Is $A := \begin{bmatrix} \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{1}{2} \end{bmatrix}$: convergent?

Solution:

$$A^2 = \begin{bmatrix} \frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix} \quad A^3 = \begin{bmatrix} \frac{1}{8} & 0 \\ \frac{3}{16} & \frac{1}{8} \end{bmatrix} \quad A^4 = \begin{bmatrix} \frac{1}{16} & 0 \\ \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

In general,

$$A^k = \begin{bmatrix} \frac{1}{2^k} & 0 \\ \frac{k}{2^{k+1}} & \frac{1}{2^k} \end{bmatrix} \to 0$$

Thus the matrix A is convergent.

Theorem: The following statements are equivalent.

(i) A is convergent.

(ii) $\lim_{n \to \infty} \left\| A^n \right\| = 0$ for some natural matrix norm.

(iii) $\lim_{n \to \infty} \left\| A^n \right\| = 0$ for all natural matrix norms.

(iv) $\rho(A) < 1$.

(v) $\lim_{n \to \infty} A^n x = 0$ for all $x \in \mathbb{R}^n$

Invertible (nonsingular) Matrices:

Let $A = (a_{ij}) \in \mathbb{R}^{n \times n}$.

Definition: The matrix A is **invertible** if there is an $n \times n$ matrix B such that $AB = BA = I$. The matrix B is called an inverse of A, and is denoted as $B = A^{-1}$.

Definition: The **transpose** of A is $A^T = (a_{ji})$. The matrix is **symmetric** if $A = A^T$.

Theorem: A square matrix can possess at most one right inverse.

Theorem: Let A and B be invertible square matrices. Then,

- $(AB)^T = B^T A^T$
- $(AB)^{-1} = B^{-1} A^{-1}$

Theorem: If A and B are square matrices such that $AB = I$, then $BA = I$.

Proof:

Let $C = BA - I + B$. Then

$$AC = ABA - A + AB = A - A + I = I.$$

By the uniqueness of the right inverse, we can conclude $B = C$, which implies $BA - I = 0$.

Invertible (Nonsingular) Matrix Theorem:

For $A \in \mathbb{R}^{n \times n}$, the following properties are equivalent:

1. The inverse of A exists, i.e., A is invertible.
2. There is a $n \times n$ matrix B such that $AB = I$.
3. There is a $n \times n$ matrix C such that $CA = I$.
4. The determinant of A is nonzero.
5. The rows of A form a basis for $\mathbb{R}^n$.
6. The columns of A form a basis for $\mathbb{R}^n$.
7. As a map from $\mathbb{R}^n$ to $\mathbb{R}^n$, A is injective (one-to-one).
8. As a map from $\mathbb{R}^n$ to $\mathbb{R}^n$, A is surjective (onto).
9. The equation $A\mathbf{x} = 0$ implies $\mathbf{x} = 0$.
10. For each $\mathbf{b} \in \mathbb{R}^n$, there is exactly one $\mathbf{x} \in \mathbb{R}^n$ such that $A\mathbf{x} = \mathbf{b}$.
11. A is a product of elementary matrices.
12. 0 is not an eigenvalue of A.

Example: (a) Show that if $A \in \mathbb{R}^{n \times n}$, then

$$det(A) = \prod_{i=1}^{n} \lambda_i$$

where λ_i are eigenvalues of A. (Hint: Consider $p(0)$.)

(b) Show that A is singular if and only if $\lambda = 0$ is an eigenvalue of A.

7.3. Iterative Algebraic Solvers

Basic Concepts:

We consider iterative methods in a more general mathematical setting. A general type of iterative process for solving the algebraic system

$$A\,x = b \tag{1}$$

can be described as follows.

Iteration via Matrix Splitting:

Split the matrix A as

$$A = M - N \tag{1.1}$$

where M is a prescribed invertible matrix. Then, the system **(1)** can be expressed equivalently as

$$M\,x = N\,x + b \tag{1.2}$$

Associated with the splitting is an iterative method

$$M\,x^{k} = N\,x^{k-1} + b \tag{1.3}$$

Since $N = M - A$, Equation **(1.3)** can be rewritten as

$$x^{k} = \left(I - M^{-1}A\right) x^{k-1} + M^{-1}\,b \qquad (k \geq 1)$$

or

$$x^{k} = x^{k-1} + M^{-1}\left(b - A\,x^{k-1}\right) \qquad (k \geq 1)$$

where x^{0} is arbitrary.

We shall say that the iterative method in **(1.3)** is **convergent** if it converges for *any* initial vector x^{0}. A sequence of vectors $x^{1}, x^{2}, \cdots$ will be computed from Equation **(1.3)**, and our objective is to choose M so that these two conditions are met:

1. The sequence $\{x^{k}\}$ *is easily computed. (Of course, M must be invertible.)*

2. The sequence $\{x^{k}\}$ *converges rapidly to the solution.*

We will see that both of these conditions follow if M is easy to invert and M^{-1} approximates A^{-1}.

Convergence: If the sequence $\{x^k\}$ converges, say to a vector x, then

$$x = (I - M^{-1}A)\,x + M^{-1}\,b$$

Thus, by letting $e^k = x - x^k$, we have

$$e^k = (I - M^{-1}A)e^{k-1},$$

which implies

$$\|e^k\| \le \|I - M^{-1}A\|\ \|e^{k-1}\| \le \|I - M^{-1}A\|^2\ \|e^{k-2}\|$$
$$\le \cdots \le \|I - M^{-1}A\|^k\ \|e^0\|$$

Thus, it can be concluded as in the following theorem.

Theorem: Convergence of Iterative Methods:

If $\|I - M^{-1}A\| < 1$ for some induced matrix norm, then the sequence produced by Equation **(1.3)** converges to the solution of $A\,x = b$ for any initial vector x^0.

Notes: Let $\delta = \|I - M^{-1}A\|$

- If

  $$M^{-1}A \approx I, \text{ or equivalently } M^{-1} \approx A^{-1},$$

  then δ will become smaller and therefore the iteration converges faster.
- If $\delta = \|I - M^{-1}A\| < 1$, then it is safe to halt the iterative process when $\|x^k - x^{k-1}\|$ is small. Indeed, since

  $$e^k = (I - M^{-1}A)e^{k-1} = (I - M^{-1}A)(e^k + x^k - x^{k-1}),$$

  we obtain $\|e^k\| \le \delta\,(\|e^k\| + \|x^k - x^{k-1}\|)$

  which implies that

  $$\|e^k\| \le \frac{\delta}{1-\delta}\ \|x^k - x^{k-1}\|$$
- The iteration **(1.3)** converges if and only if

  $$\rho(I - M^{-1}A) = \rho(M^{-1}N) < 1$$

Richardson Method: $M = I$

The iterative method is called the Richardson method if M is simply chosen to be the identity matrix. Equation **(1.3)** in this case reads

$$x^{k} = (\mathrm{I} - A)\, x^{k-1} + b \tag{2}$$

which can rewritten as

$$x^{k} = x^{k-1} + r^{k-1} \tag{3}$$

where $r^{k-1} = b - A\, x^{k-1}$.

Maple code

```
Richardson := proc(n, A, b, x, itmax)
    local k, i, r;
    r := Vector(n);
    for k from 1 to itmax do
        for i from 1 to n do
            r[i] := b[i] - add(A[i, j]·x[j], j = 1 ..n);
        end do;
        for i from 1 to n do
            x[i] := x[i] + r[i];
        end do;
        print(`k=`, k, evalf(x));
    end do;
end proc:

n := 3 :
```

$$A := \frac{1}{6} \cdot Matrix([[6, 3, 2], [2, 6, 3], [3, 2, 6]]) = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{3} & 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} & 1 \end{bmatrix}$$

$$b := \frac{1}{6} \cdot Vector([11, 11, 11]) = \begin{bmatrix} \frac{11}{6} \\ \frac{11}{6} \\ \frac{11}{6} \end{bmatrix}$$

$x := Vector([0, 0, 0])$:
$Richardson(n, A, b, x, 10)$

$$k=, 1, \begin{bmatrix} 1.134587986 \\ 1.134587986 \\ 1.134587986 \end{bmatrix}$$

$$k=, 2, \begin{bmatrix} 0.8878433452 \\ 0.8878433452 \\ 0.8878433452 \end{bmatrix}$$

$$k=, 3, \begin{bmatrix} 1.093463879 \\ 1.093463879 \\ 1.093463879 \end{bmatrix}$$

$$k=, 4, \begin{bmatrix} 0.9221134342 \\ 0.9221134342 \\ 0.9221134342 \end{bmatrix}$$

$$k=, 5, \begin{bmatrix} 1.064905472 \\ 1.064905472 \\ 1.064905472 \end{bmatrix}$$

$$k=, 6, \begin{bmatrix} 0.9459121071 \\ 0.9459121071 \\ 0.9459121071 \end{bmatrix}$$

$$k=,\ 7,\ \begin{bmatrix} 1.045073244 \\ 1.045073244 \\ 1.045073244 \end{bmatrix}$$

$$k=,\ 8,\ \begin{bmatrix} 0.9624389632 \\ 0.9624389632 \\ 0.9624389632 \end{bmatrix}$$

$$k=,\ 9,\ \begin{bmatrix} 1.031300864 \\ 1.031300864 \\ 1.031300864 \end{bmatrix}$$

$$k=,\ 10,\ \begin{bmatrix} 0.9739159467 \\ 0.9739159467 \\ 0.9739159467 \end{bmatrix} \quad \textbf{(4.1)}$$

```
with(LinearAlgebra) :
Id := Matrix(3, shape = identity) :
evalf(Eigenvalues(Id − A))
```

$$\begin{bmatrix} -0.8333333333 \\ 0.4166666667 - 0.1443375673\,\mathrm{I} \\ 0.4166666667 + 0.1443375673\,\mathrm{I} \end{bmatrix} \quad \textbf{(4.2)}$$

On the other hand:

```
B := Matrix([[6, 3, 2], [2, 6, 3], [3, 2, 6]]) :
evalf(Eigenvalues(Id − B))
```

$$\begin{bmatrix} -10. \\ -2.500000000 - 0.8660254040\,\mathrm{I} \\ -2.500000000 + 0.8660254040\,\mathrm{I} \end{bmatrix} \quad \textbf{(4.3)}$$

Thus, the Richardson method does not converge for $B\,x = b$.

Relaxation Methods: $A = M - N$

We first express $A = [a_{ij}]$ as the matrix sum

$$A = D - E - F \tag{4}$$

$$= \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{n,n} \end{bmatrix} - \begin{bmatrix} 0 & 0 & \cdots & 0 \\ -a_{21} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ -a_{n,1} & \cdots & -a_{n,n-1} & 0 \end{bmatrix}$$

$$- \begin{bmatrix} 0 & -a_{12} & \cdots & -a_{1,n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -a_{n-1,n} \\ 0 & \cdots & 0 & 0 \end{bmatrix}$$

Then, a relaxation method can be formulated by selecting M and N for the matrix splitting

$$A = M - N \tag{5}$$

Examples are

Relaxation Methods	M	N
Jacobi method	D	$E + F$
Gauss-Seidel method	$D - E$	F
SOR method	$\frac{1}{\omega} D - E, \quad \omega \in (0, 2)$	$\frac{1 - \omega}{\omega} D + F$

* SOR stands for "Successive Over Relaxation."

Jacobi Method:

The Jacobi method is formulated as

$$\mathrm{D}\,x^{k} = (E+F)\,x^{k-1} + b \tag{6}$$

which is the same as choosing

$$M = D,\ N = E + F \tag{7}$$

in the matrix splitting (5). The i-th component of (6) reads

$$a_{ii}x_i^k = b_i + \sum_{j=1}^{i-1}\left(-a_{ij}x_j^{(k-1)}\right) + \sum_{j=i+1}^{n}\left(-a_{ij}x_j^{(k-1)}\right) \tag{8}$$

or, equivalently,

$$x_i^k = \frac{b_i - \left(\sum_{j=1}^{i-1} a_{ij}x_j^{(k-1)}\right) - \left(\sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right)}{a_{ii}} \tag{9}$$

Example: Let

$$A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}: \quad b := \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}:$$

Find x^k, $k = 1, 2, 3$, beginning from $x0 := Vector([1, 1, 1]) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

Solution:

```
x1 := Vector(3) : x2 := Vector(3) : x3 := Vector(3) :
x1[1] := (b[1] − A[1, 2]·x0[2] − A[1, 3]·x0[3])/A[1, 1] :
x1[2] := (b[2] − A[2, 1]·x0[1] − A[2, 3]·x0[3])/A[2, 2] :
x1[3] := (b[3] − A[3, 1]·x0[1] − A[3, 2]·x0[2])/A[3, 3] :
x1
```

$$\begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} \tag{5.1}$$

$x2[1] := (b[1] - A[1, 2]\cdot x1[2] - A[1, 3]\cdot x1[3])/A[1, 1]$:
$x2[2] := (b[2] - A[2, 1]\cdot x1[1] - A[2, 3]\cdot x1[3])/A[2, 2]$:
$x2[3] := (b[3] - A[3, 1]\cdot x1[1] - A[3, 2]\cdot x1[2])/A[3, 3]$:
$x2$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \tag{5.2}$$

$x3[1] := (b[1] - A[1, 2]\cdot x2[2] - A[1, 3]\cdot x2[3])/A[1, 1]$:
$x3[2] := (b[2] - A[2, 1]\cdot x2[1] - A[2, 3]\cdot x2[3])/A[2, 2]$:
$x3[3] := (b[3] - A[3, 1]\cdot x2[1] - A[3, 2]\cdot x2[2])/A[3, 3]$:
$x3$

$$\begin{bmatrix} \frac{3}{2} \\ 2 \\ \frac{7}{2} \end{bmatrix} \tag{5.3}$$

The true solution is $(2, 3, 4)^t$

Maple code: Jacobi Method

```
Jacobi := proc(n, A, b, x, tol, itmax)
    local i, j, k, id1, id2, xx;
    xx := Matrix(n, 2);
    for i from 1 to n do
        xx[i, 1] := x[i];
    end do;
    for k from 1 to itmax do
        id1 := modp(k + 1, 2) + 1;
        id2 := modp(k, 2) + 1;
        for i from 1 to n do
            xx[i, id2] := (b[i] - add(A[i, j]·xx[j, id1], j = 1 ..i - 1)
                                - add(A[i, j]·xx[j, id1], j = i + 1 ..n)) / A[i,
    i];
        end do;
        print(`k=`, k, evalf(xx[1 ..n, id2]));
    end do;
end proc:

Jacobi(3, A, b, x0, tol, 10)
```

$$k=,\ 1,\ \begin{bmatrix} 1. \\ 1. \\ 3. \end{bmatrix}$$

$$k=,\ 2,\ \begin{bmatrix} 1. \\ 2. \\ 3. \end{bmatrix}$$

$$k=,\ 3,\ \begin{bmatrix} 1.500000000 \\ 2. \\ 3.500000000 \end{bmatrix}$$

$$k=,\ 4,\ \begin{bmatrix} 1.500000000 \\ 2.500000000 \\ 3.500000000 \end{bmatrix}$$

$$k=,\ 5,\ \begin{bmatrix} 1.750000000 \\ 2.500000000 \\ 3.750000000 \end{bmatrix}$$

$$k=,\ 6,\ \begin{bmatrix} 1.750000000 \\ 2.750000000 \\ 3.750000000 \end{bmatrix}$$

$$k=,\ 7,\ \begin{bmatrix} 1.875000000 \\ 2.750000000 \\ 3.875000000 \end{bmatrix}$$

$$k=,\ 8,\ \begin{bmatrix} 1.875000000 \\ 2.875000000 \\ 3.875000000 \end{bmatrix}$$

$$k=,\ 9,\ \begin{bmatrix} 1.937500000 \\ 2.875000000 \\ 3.937500000 \end{bmatrix}$$

$$k=,\ 10,\ \begin{bmatrix} 1.937500000 \\ 2.937500000 \\ 3.937500000 \end{bmatrix} \qquad \textbf{(6.1)}$$

Gauss-Seidel Method:

The Gauss-Seidel method is formulated as

$$(\mathrm{D}-E)\,x^{k}=F\,x^{k-1}+b \tag{10}$$

which is the same as choosing

$$M=\mathrm{D}-E,\ N=F \tag{11}$$

in the matrix splitting **(5)**. The i-th component of **(10)** reads

$$a_{ii}\,x_i^k-\left(\sum_{j=1}^{i-1}\left(-a_{ij}\,x_j^k\right)\right)=b_i+\sum_{j=i+1}^{n}\left(-a_{ij}\,x_j^{(k-1)}\right) \tag{12}$$

or, equivalently,

$$x_i^k=\frac{b_i-\left(\sum_{j=1}^{i-1}a_{ij}\,x_j^k\right)-\left(\sum_{j=i+1}^{n}a_{ij}\,x_j^{(k-1)}\right)}{a_{ii}} \tag{13}$$

The difference is that the GS method utilizes updated values.

Maple code: Gauss-Seidel Method

```
GaussSeidel := proc(n, A, b, x, tol, itmax)
    local i, j, k;
    for k from 1 to itmax do
        for i from 1 to n do
            x[i] := (b[i] - add(A[i,j]·x[j], j = 1..i - 1)
                          - add(A[i,j]·x[j], j = i + 1..n))/A[i, i];
        end do;
        print(`k=`, k, evalf(x));
    end do;
end proc:
```

$$A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}:$$

$$b := \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}:$$

$$x0 := Vector([1, 1, 1]) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$GaussSeidel(3, A, b, x0, tol, 10)$$

$$k=, 1, \begin{bmatrix} 1. \\ 1. \\ 3. \end{bmatrix}$$

$$k=, 2, \begin{bmatrix} 1. \\ 2. \\ 3.500000000 \end{bmatrix}$$

$$k=, 3, \begin{bmatrix} 1.500000000 \\ 2.500000000 \\ 3.750000000 \end{bmatrix}$$

$$k=, 4, \begin{bmatrix} 1.750000000 \\ 2.750000000 \\ 3.875000000 \end{bmatrix}$$

$$k=, 5, \begin{bmatrix} 1.875000000 \\ 2.875000000 \\ 3.937500000 \end{bmatrix}$$

$$k=,\ 6,\ \begin{bmatrix} 1.937500000 \\ 2.937500000 \\ 3.968750000 \end{bmatrix}$$

$$k=,\ 7,\ \begin{bmatrix} 1.968750000 \\ 2.968750000 \\ 3.984375000 \end{bmatrix}$$

$$k=,\ 8,\ \begin{bmatrix} 1.984375000 \\ 2.984375000 \\ 3.992187500 \end{bmatrix}$$

$$k=,\ 9,\ \begin{bmatrix} 1.992187500 \\ 2.992187500 \\ 3.996093750 \end{bmatrix}$$

$$k=,\ 10,\ \begin{bmatrix} 1.996093750 \\ 2.996093750 \\ 3.998046875 \end{bmatrix} \qquad \textbf{(7.1)}$$

By comparison with the result in **(6.1)**, we may conclude that Gauss-Seidel method is faster than the Jacobi method.

Successive Over Relaxation (SOR) method:

The SOR method is formulated as

$$(\mathrm{D} - \omega E)\, x^{k} = \left((1-\omega)\,\mathrm{D} + \omega F\right) x^{k-1} + \omega b \tag{14}$$

which is the same as choosing

$$M = \frac{\mathrm{D}}{\omega} - E,\; N = \frac{(1-\omega)\,\mathrm{D}}{\omega} + F \tag{15}$$

Note that **(14)** can be equivalently written as

$$\mathrm{D}\,x^{k} = (1-\omega)\,\mathrm{D}\,x^{k-1} + \omega\left(b + E x^{k} + F x^{k-1}\right) \tag{16}$$

Thus the *i*-th component of **(16)** reads

$$a_{ii} x_i^k = (1-\omega)\, a_{ii} x_i^{(k-1)} + \omega\left(b_i + \sum_{j=1}^{i-1}\left(-a_{ij} x_j^k\right) + \sum_{j=i+1}^{n}\left(-a_{ij} x_j^{(k-1)}\right)\right) \tag{17}$$

or, equivalently,

$$x_{GS,\,i}^{k} = \frac{b_i - \left(\sum_{j=1}^{i-1} a_{ij} x_j^k\right) - \left(\sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)}\right)}{a_{ii}}$$

$$x_i^k = (1-\omega)\; x_i^{(k-1)} + \omega\, x_{GS,\,i}^{k} \tag{18}$$

Note that if $\omega = 1$, SOR turns out to be the Gauss-Seidel method.

Maple code: SOR method

```
SOR := proc(n, A, b, x, w, tol, itmax)
    local i, j, k, xGS;
    for k from 1 to itmax do
        for i from 1 to n do
            xGS := (b[i] - add(A[i, j]·x[j], j = 1 ..i - 1)
                        -add(A[i, j]·x[j], j = i + 1 ..n)) /A[i, i];
            x[i] := (1 - w)·x[i] + w·xGS;
        end do;
        print(`k=`, k, evalf(x));
    end do;
end proc:
```

$$A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}:$$

$$b := \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}:$$

$$x0 := Vector([1, 1, 1]) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$SOR(3, A, b, x0, 1.2, tol, 10)$$

$$k=, 1, \begin{bmatrix} 1.0 \\ 1.000000000 \\ 3.400000000 \end{bmatrix}$$

$$k=,\ 2,\ \begin{bmatrix} 1.000000000 \\ 2.440000000 \\ 3.784000000 \end{bmatrix}$$

$$k=,\ 3,\ \begin{bmatrix} 1.864000000 \\ 2.900800000 \\ 3.983680000 \end{bmatrix}$$

$$k=,\ 4,\ \begin{bmatrix} 1.967680000 \\ 2.990656000 \\ 3.997657600 \end{bmatrix}$$

$$k=,\ 5,\ \begin{bmatrix} 2.000857600 \\ 3.000977920 \\ 4.001055232 \end{bmatrix}$$

$$k=,\ 6,\ \begin{bmatrix} 2.000415232 \\ 3.000686694 \\ 4.000200970 \end{bmatrix}$$

$$k=,\ 7,\ \begin{bmatrix} 2.000328970 \\ 3.000180625 \\ 4.000068180 \end{bmatrix}$$

$$k=,\ 8,\ \begin{bmatrix} 2.000042580 \\ 3.000030331 \\ 4.000004563 \end{bmatrix}$$

$$k=,\ 9,\ \begin{bmatrix} 2.000009683 \\ 3.000002482 \\ 4.000000576 \end{bmatrix}$$

$$k=,\ 10,\ \begin{bmatrix} 1.999999552 \\ 2.999999581 \\ 3.999999633 \end{bmatrix} \tag{8.1}$$

SOR with $\omega = 1.2$ is much faster than Jacobi and Gauss-Seidel methods.

Convergence Theory:

Theorem: For any $x^0 \in \mathbb{R}^n$, the sequence $\{x^k\}_{k=1}^{\infty}$ defined by

$$x^k = Tx^{k-1} + c \tag{19}$$

converges to the unique solution of $x = Tx + c$ if and only if $\rho(T) < 1$. In this case, the iterates satisfy

$$\|x - x^k\| \leq \|T\|^k \, \|x - x^0\|$$

For example,

Relaxation Methods	T (Iteration matrix)
Jacobi method	$T_J = \mathrm{D}^{-1}(E + F)$
Gauss-Seidel method	$T_{GS} = (\mathrm{D} - E)^{-1} F$
SOR method	$T_{SOR} = (\mathrm{D} - \omega E)^{-1}[(1 - \omega)\,\mathrm{D} + \omega F]$

Theorem ([Stein and Rosenberg, 1948]). One and only one of the following mutually exclusive relations is valid:

1. $\rho(T_J) = \rho(T_{GS}) = 0$
2. $0 < \rho(T_{GS}) < \rho(T_J) < 1$
3. $\rho(T_J) = \rho(T_{GS}) = 1$
4. $1 < \rho(T_J) < \rho(T_{GS})$

Theorem: Let A be symmetric. Then,

$$\rho\left(T_{SOR}\right) < 1 \Leftrightarrow A \text{ is positive definite and } 0 < \omega < 2$$

Optimal ω for SOR: For algebraic systems of *good* properties, it is theoretically known that the convergence of SOR can be optimized when

$$\omega = \frac{2}{1 + \sqrt{1 - \rho\left(T_J\right)^2}}$$

However, in most cases you can find a better ω for a given algebraic system.

Graph theory for the estimation of the spectral radius

Definition: For $n \geq 2$, a matrix $A \in \mathbb{C}^{n \times n}$ is **reducible** if there is an $n \times n$ permutation matrix P such that

$$P A P^T = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}$$

where $A_{11} \in \mathbb{C}^{r \times r}$ and $A_{22} \in \mathbb{C}^{(n-r) \times (n-r)}$, submatrices, for which $1 \leq r < n$. If no such permutation matrix exists, then A is **irreducible**.

The geometrical interpretation of the concept of the irreducibility by means of graph theory is useful. Let $A \in \mathbb{C}^{n \times n}$, and consider any distinct points

$$P_1, P_2, \bullet\bullet\bullet, P_n$$

in the plane, which we will call **nodes**. For any nonzero a_{ij} of A, we connect P_i to P_j by a path $\overrightarrow{P_i P_j}$, directed from the node P_i to the node P_j; for a nonzero a_{ii}, the node P_i is joined to itself by a directed loop. In this way, every $A \in \mathbb{C}^{n \times n}$ can be associated with a **directed graph** $G(A)$.

Example: Find the directed graph of $A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$.

Solution:

Definition: A directed graph is **strongly connected**, if , for any ordered pair of nodes (P_i, P_j), there is a directed path of a finite length

$$\overrightarrow{P_i P_{k_1}}, \overrightarrow{P_{k_1} P_{k_2}}, \cdots, \overrightarrow{P_{k_{r-1}} P_{k_r = j}}$$

connecting from P_i to P_j.

Example: The directed graph of the above A is strongly connected. How about for the following matrix?

$$B := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & 0 \\ 0 & -1 & 2 \end{bmatrix}$$

Solution:

Theorem: A matrix $A \in \mathbb{C}^{n \times n}$ is irreducible if and only if its directed graph $G(A)$ is strongly connected.

Definitions:

- A matrix $A \in \mathbb{C}^{n \times n}$ is **diagonally dominant** if

$$|a_{ii}| \geq \sum_{j=1, j \neq i}^{n} |a_{ij}| \equiv \Lambda_i \quad (1 \leq i \leq n)$$

- A matrix $A \in \mathbb{C}^{n \times n}$ is **irreducibly diagonally dominant** if A is irreducible and diagonally dominant, with strict inequality holding in the above for at least one i.

Theorem: Let $A \in \mathbb{C}^{n \times n}$ be strictly or irreducibly diagonally dominant. Then, A is nonsingular. If all the diagonal entries of A are in addition positive real, then real parts of all eigenvalues of A are all positive.

Theorem: Let $A \in \mathbb{C}^{n \times n}$ be irreducible. Then,
(a) ([Gerschgorin, 1931]) All eigenvalues of A lies in the union of the disks in the complex plane

$$\cup_{i=1}^{n} \{ z : |z - a_{ii}| \leq \Lambda_i \}$$

(b) ([Taussky, 1948]) In addition, assume that λ, an eigenvalue of A, is a boundary point of the union of the disks $|z - a_{ii}| \leq \Lambda_i$. Then, all the n circles $|z - a_{ii}| = \Lambda_i$ must pass through the point λ, i.e.,

$$|\lambda - a_{ii}| = \Lambda_i \quad (1 \leq i \leq n)$$

Example: Let $A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$:

Spectral radius of iteration matrices:

Jacobi method:

$M := \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$: $N := M - A$:

$TJ := M^{-1}.N$

$$\begin{bmatrix} 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \end{bmatrix} \quad \textbf{(10.1)}$$

Thus, $-1 < \mathrm{Re}(\lambda_i) < 1$ and therefore $\rho(TJ) < 1$.

$with(LinearAlgebra)$:

$$Eigenvalues(TJ) = \begin{bmatrix} 0 \\ \frac{1}{2}\sqrt{2} \\ -\frac{1}{2}\sqrt{2} \end{bmatrix}$$

$\rho(TJ) = evalf\left(\frac{1}{2}\sqrt{2}\right) = 0.7071067810$

Gauss-Seidel method

$$M := \begin{bmatrix} 2 & 0 & 0 \\ -1 & 2 & 0 \\ 0 & -1 & 2 \end{bmatrix} : \quad N := M - A :$$

$$TGS := M^{-1}.N$$

$$\begin{bmatrix} 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} \\ 0 & \frac{1}{8} & \frac{1}{4} \end{bmatrix} \tag{10.2}$$

Thus, $-\frac{1}{2} < \mathrm{Re}\left(\lambda_i\right) < \frac{3}{4}$ and therefore $\rho(TGS) < \frac{3}{4}$.

$$Eigenvalues(TGS) = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2} \end{bmatrix}$$

$$\rho(TGS) = \frac{1}{2}$$

SOR method

$$M := \begin{bmatrix} 2/w & 0 & 0 \\ -1 & 2/w & 0 \\ 0 & -1 & 2/w \end{bmatrix} : \quad N := M - A :$$

$$TSOR := M^{-1}.N$$

$$\begin{bmatrix} \frac{1}{2} w \left(\frac{2}{w} - 2 \right) & \frac{1}{2} w & 0 \\ \frac{1}{4} w^2 \left(\frac{2}{w} - 2 \right) & \frac{1}{4} w^2 + \frac{1}{2} w \left(\frac{2}{w} - 2 \right) & \frac{1}{2} w \\ \frac{1}{8} w^3 \left(\frac{2}{w} - 2 \right) & \frac{1}{8} w^3 + \frac{1}{4} w^2 \left(\frac{2}{w} - 2 \right) & \frac{1}{4} w^2 + \frac{1}{2} w \left(\frac{2}{w} - 2 \right) \end{bmatrix} \quad \textbf{(10.3)}$$

$$w := \frac{6}{5} :$$

$$TSOR = \begin{bmatrix} -\frac{1}{5} & \frac{3}{5} & 0 \\ -\frac{3}{25} & \frac{4}{25} & \frac{3}{5} \\ -\frac{9}{125} & \frac{12}{125} & \frac{4}{25} \end{bmatrix}$$

$$Eigenvalues(TSOR) = \begin{bmatrix} -\frac{1}{5} \\ \frac{4}{25} - \frac{3}{25} \mathrm{I} \\ \frac{4}{25} + \frac{3}{25} \mathrm{I} \end{bmatrix}$$

$$\rho(TSOR) = \max\left(\mathrm{abs}\left(\frac{4}{25} + \frac{3}{25} \mathrm{I} \right), \frac{1}{5} \right) = \frac{1}{5}$$

Empty Page

7.4. Krylov Subspace Methods

We consider Krylov subspace methods for solving

$$A\,x = b \tag{1}$$

where A is symmetric positive definite (SPD), i.e.,

$$A^T = A, \quad x^T A\,x > 0 \text{ for } x \neq 0$$

Theorem: Let $A \in \mathbb{R}^{n \times n}$. The following are equivalent.

1. A is symmetric positive definite.
2. $A = A^T$ and its eigenvalues are all positive.
3. Each of its leading principal submatrices has a positive determinant.
4. A can be factored into LL^T, where L is a lower triangular matrix with positive diagonal entries.

Example: Show that $A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ is SPD.

Solution: Clearly $A^T = A$. For $x = \left[x_1, x_2, x_3\right]^T$,

$$x^T A\,x = \sum_{i,j=1}^{3} x_i a_{ij} x_j = x_1^2 + \left(x_1 - x_2\right)^2 + \left(x_2 - x_3\right)^2 + x_3^2$$

which is zero only if $x = 0$.

Given an initial guess $x^0 \in \mathbb{R}^n$, Krylov subspace methods find successive approximations of the form

$$x^{k+1} = x^k + \alpha_k p^k \tag{2}$$

where p^k is the **search direction** and $\alpha_k > 0$ is the **step length**. Different methods differ in the choice of the search direction and the step length. In this section, we will consider the steepest descent method (also known as the gradient method, or Richardson's method) and the conjugate gradient method.

Throughout we use the **inner-product** or **dot product** notation for real vectors x and y:

$$\langle x, y \rangle = x \bullet y = x^T y = \sum_{i=1}^{n} x_i y_i$$

Some immediate properties are:

1. $\langle x, y \rangle = \langle y, x \rangle$
2. $\langle \alpha x, y \rangle = \alpha \langle x, y \rangle$, for any constant α
3. $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$
4. $\langle x, Ay \rangle = \langle A^T x, y \rangle$

Theorem: Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite. Then, $\widehat{x}$ is a solution of

$$Ax = b$$

if and only if $\widehat{x}$ is a minimizer of

$$g(x) = \frac{1}{2}\langle x, Ax\rangle - \langle x, b\rangle$$

Proof (sketch). For $0 \neq v \in \mathbb{R}^n$ and $t \in \mathbb{R}$,

$$h(t) \equiv g(x + t\,v) = g(x) - t < v, b - Ax > + \frac{t^2}{2} < v, A\,v >$$

Then, the quadratic function $h(t)$ has a minimum value when $h'(t) = 0$, because $\langle v, A\,v\rangle$ is positive. It follows from

$$h'(t) = -\langle v, b - Ax\rangle + t\,\langle v, A\,v\rangle = 0$$

that

$$\widehat{t} = \frac{\langle v, b - Ax\rangle}{\langle v, A\,v\rangle}$$

Thus,

$$h(\widehat{t}\,) = g(x + \widehat{t}\,v) = g(x) - \frac{1}{2}\frac{\langle v, b - Ax\rangle^2}{\langle v, A\,v\rangle}$$

If $A\widehat{x} = b$, then $\langle v, b - A\widehat{x}\,\rangle = 0$ for all choices of v and therefore $g(x)$ cannot be smaller than $g(\widehat{x}\,)$. Thus, $\widehat{x}$ is a minimizer of $g(x)$. On the other hand, suppose that $\widehat{x}$ is a minimizer of $g(x)$. Then, for any vector $v \neq 0$, we have $g(\widehat{x} + \widehat{t}\,v) \geq g(\widehat{x}\,)$, which is possible only if $\langle v, b - A\widehat{x}\,\rangle = 0$ in the last displaced equation. This implies that $b - A\widehat{x} = 0$, because v is arbitrary.

The Gradient (Steepest Descent) method

We denote the gradient and Hessian of g by g' and g", respectively:

$$g'(x) = A\,x - b, \quad g"(x) = A$$

(Search direction): Given x^{k+1} as in **(2)**, the Taylor's formula says

$$\begin{aligned} g\left(x^{k+1}\right) &= g\left(x^k + \alpha_k p^k\right) \\ &= g\left(x^k\right) + \alpha_k < g'\left(x^k\right), p^k > + \frac{\alpha_k^2}{2} < p^k, g"(\xi) P^k > \\ &= g\left(x^k\right) + \alpha_k < g'\left(x^k\right), p^k > + \mathrm{O}\left(\alpha_k^2\right) \end{aligned}$$

as $\alpha_k \to 0$. The goal is to find p^k and α_k such that

$$g\left(x^{k+1}\right) < g\left(x^k\right) \qquad \textbf{(3)}$$

which can be achieved if

$$\left\langle g'\left(x^k\right), p^k \right\rangle < 0$$

In particular, **(3)** holds if we choose

$$\boxed{p^k = -g'\left(x^k\right) = b - A\,x^k \equiv r^k}$$

when $g'\left(x^k\right) \neq 0$. (Due to the above choice of p^k, the method is called the steepest descent method.)

(Step length): We may determine α_k such that

$$g\left(x^k + \alpha_k p^k\right) = \min_{\alpha} g\left(x^k + \alpha\, p^k\right)$$

in which case α_k is said to be *optimal*. If α_k is optimal, then

$$0 = \frac{\mathrm{d}}{\mathrm{d}\,\alpha}\, g\left(x^k + \alpha\, p^k\right)\Big|_{\alpha = \alpha_k} = \left\langle g'\left(x^k + \alpha_k p^k\right), p^k \right\rangle$$

$$= \left\langle A\left(x^k + \alpha_k p^k\right) - b, p^k \right\rangle = \left\langle A\,x^k - b, p^k \right\rangle + \alpha_k < p^k, A\,p^k >$$

So, we obtain

$$\alpha_k = \frac{\left\langle r^k, p^k \right\rangle}{\left\langle p^k, A\,p^k \right\rangle} = \frac{\left\langle r^k, r^k \right\rangle}{\left\langle r^k, A\,r^k \right\rangle}$$

Convergence of the steepest descent method: For the method, the following is known

$$\left\| x - x^k \right\|_2 \leq \left(1 - \frac{1}{\kappa(A)} \right)^k \left\| x - x^0 \right\|_2$$

The number of iterations required to reduce the error by a factor of ε is in the order of the condition number of A, that is,

$$\kappa(A) \ln\left(\frac{1}{\varepsilon} \right) \leq k \qquad \textbf{(4)}$$

The Conjugate Gradient (CG) method

In this method, the search directions p^k are conjugate, i.e.,

$$\langle p^i, A p^j \rangle = 0, \quad i \neq j$$

and the step length α_k is chosen to be optimal. The following is the original version of the CG method:

(CG.Ver.1)

Select x^0, ε;

$r^0 = b - A x^0$; $p^0 = r^0$;

for k **from** 0 **to** *itmax* **do**

$\alpha_k = \langle r^k, p^k \rangle / \langle p^k, A p^k \rangle$; (CG1)

$x^{k+1} = x^k + \alpha_k p^k$; (CG2)

$r^{k+1} = r^k - \alpha_k A p^k$; (CG3)

if $\| r^{k+1} \|_2 \leq \varepsilon \| r^0 \|_2$, stop;

$\beta_k = -\langle r^{k+1}, A p^k \rangle / \langle p^k, A p^k \rangle$; (CG4)

$p^{k+1} = r^{k+1} + \beta_k p^k$; (CG5)

end do

Remarks, for CG method:

- $r^k = b - A x^k$, by definition. So,

$$\begin{aligned} r^{k+1} &= b - A x^{k+1} = b - A\left(x^k + \alpha_k p^k \right) \\ &= b - A x^k - \alpha_k A p^k = r^k - \alpha_k A p^k \end{aligned}$$

  which is (CG3).

- α_k in (CG1) is computed in order to impose

$$\left\langle r^{k+1}, p^k \right\rangle = 0,$$

  which can be verified using (CG3).

- β_k in (CG4) is determined so as to satisfy

$$\left\langle p^{k+1}, A p^k \right\rangle = 0,$$

  which can be verified using (CG5).

Properties of CG method

- For $m = 0, 1, 2, \cdots,$

$$\text{span}\{p^0, p^1, \cdots, p^m\} = \text{span}\{r^0, r^1, \cdots, r^m\}$$
$$= \text{span}\{r^0, A r^0, \cdots, A^m r^0\}$$

- The search directions and residuals satisfy

$$\langle p^i, A p^j \rangle = 0, \quad \langle r^i, r^j \rangle = 0, \quad i \neq j$$

- The CG method converges in maximum n iterations. That is, $A x^m = b$ for some $m \leq n$. (Originally, the CG method was developed as a direct method.)

- For CG method, the following is known

$$\| x - x^k \|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \| x - x^0 \|_A$$

- The number of iterations required to reduce the error by a factor of ε is reduced to

$$\frac{1}{2} \sqrt{\kappa(A)} \ln\left(\frac{2}{\varepsilon} \right) \leq k \qquad \textbf{(5)}$$

(CG.Ver.2)

Select x^0, ε;

$r^0 = b - A x^0$; $p^0 = r^0$; $\rho_0 = \langle r^0, r^0 \rangle$;

for k **from** 0 **to** *itmax* **do**

$\alpha_k = \rho_k / \langle p^k, A p^k \rangle$;

$x^{k+1} = x^k + \alpha_k p^k$;

$r^{k+1} = r^k - \alpha_k A p^k$;

if $\| r^{k+1} \|_2 \leq \varepsilon \| r^0 \|_2$, stop;

$\rho_{k+1} = \langle r^{k+1}, r^{k+1} \rangle$

$\beta_k = \rho_{k+1} / \rho_k$;

$p^{k+1} = r^{k+1} + \beta_k p^k$;

end do

Note that

$$\langle r^k, p^k \rangle = \langle r^k, r^k + \beta_{k-1} p^{k-1} \rangle = \langle r^k, r^k \rangle \equiv \rho_k$$

$$\beta_k = -\langle r^{k+1}, A p^k \rangle / \langle p^k, A p^k \rangle = -\langle r^{k+1}, A p^k \rangle \frac{\alpha_k}{\rho_k}$$

$$= \langle r^{k+1}, r^{k+1} - r^k \rangle \frac{1}{\rho_k} = \frac{\rho_{k+1}}{\rho_k}$$

Example: Use the CG method to solve $A\,x = b$, where

$$A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}: \qquad b := \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}:$$

starting from $x0 := Vector([1, 1, 1])$:

Solution:

$n := 3$:
$x := x0$: $r := b - A.x$: $p := r$:
$rho0 := r^{\%T}.r$:
for k **from** 1 **to** n **do**
$q := A.p;$
$al := \dfrac{rho0}{p^{\%T}.q};$
$x := x + al \cdot p;$
$r := r - al \cdot q;$
$rho1 := r.r;$
$be := \dfrac{rho1}{rho0};$
$p := r + be \cdot p;$
$rho0 := rho1;$
$print(`k=`, k, x, r);$
if $\left(\mathrm{sqrt}(rho1) < 10^{-8}\right)$ **then break**; **end if**;
end do:

$$k=,\ 1,\ \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix},\ \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$$

$$k=, 2, \begin{bmatrix} 1 \\ \frac{7}{3} \\ \frac{11}{3} \end{bmatrix}, \begin{bmatrix} \frac{4}{3} \\ 0 \\ 0 \end{bmatrix}$$

$$k=, 3, \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \textbf{(1.1)}$$

Preconditioned CG (PCG) method

As shown in Equations **(4)** and **(5)**, the condition number of A is the critical point for the convergence of Krylov subspace methods such as Gradient and CG methods. If we can find a matrix M such that

$$M \approx A \quad \text{and easy to invert,}$$

we may try to apply the CG method to

$$M^{-1} A x = M^{-1} b$$

Then, since

$$\kappa\left(M^{-1}A\right) \ll \kappa(A)$$

the CG iteration will converge much faster.

Common choices of M:

- $M = \text{diag}(A)$
- $M = ILU0(A) \quad \Rightarrow$ PCG-ILU0

In practice, we do not have to multiply M^{-1} to the original algebraic system. The idea can be implemented equivalently as follows:

(PCG)

Select x^0, ε;

$r^0 = b - A x^0$; $M z^0 = r^0$;
$p^0 = z^0$; $\rho_0 = \langle z^0, r^0 \rangle$;

for k **from** 0 **to** *itmax* **do**

$\alpha_k = \rho_k / \langle p^k, A p^k \rangle$;

$x^{k+1} = x^k + \alpha_k p^k$;

$r^{k+1} = r^k - \alpha_k A p^k$;

if $\| r^{k+1} \|_2 \leq \varepsilon \| r^0 \|_2$, stop;

$M z^{k+1} = r^{k+1}$

$\rho_{k+1} = \langle z^{k+1}, r^{k+1} \rangle$

$\beta_k = \rho_{k+1} / \rho_k$;

$p^{k+1} = r^{k+1} + \beta_k p^k$;

end do

Empty Page

Homework:
7. Iterative Algebraic Solvers

#1. a. Verify that the function $\| \cdot \|_1$, defined on $\mathbb{R}^n$ by

$$\| x \|_1 = \sum_{i=1}^{n} | x_i |$$

is a norm on $\mathbb{R}^n$.

b. Prove that $\| x \|_2 \leq \| x \|_1$ for all $x \in \mathbb{R}^n$.

#2. Let $A := \begin{bmatrix} 2 & 1 & -2 \\ 0 & 3 & -1 \\ 4 & 5 & 1 \end{bmatrix}$: Find $\| A \|_1$, $\| A \|_2$, and $\| A \|_\infty$.

#3. Let $A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$:

a. Use Gerschgorin-Taussky theorem to find a range of eigenvalues of A.

b. Is A nonsingular?

#4. When the boundary-value problem

$$\begin{cases} -u_{xx} = -2, & 0 < x < 4 \\ u_x(0) = 0, \; u(4) = 16 \end{cases}$$

is discretized by the second-order finite difference method with $h = 1$, the algebraic system reads

$$A\,x = b$$

where

$$A := \begin{bmatrix} 2 & -2 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}: \qquad b := \begin{bmatrix} -2 \\ -2 \\ -2 \\ 14 \end{bmatrix}:$$

a. Is A irreducibly diagonally dominant?
b. Perform 10 iterations of Jacobi and Gauss-Seidel methods, starting from $x^0 = [0, 0, 0, 0]^T$.
c. The exact solution $x = [0, 1, 4, 9]^T$. Try to find the best ω with which the SOR method converges fastest during the first 10 iterations.
d. Find the spectral radii of the iteration matrices.
e. Which method is the best?

#5. Symmetrize the algebraic system considered in the preceding problem and apply CG algorithm to solve it.

8. Approximation Theory

In This Chapter:

Topics	Applications/Properties
Least Squares (LS) Approximation	Over-determined systems
Linear models	
Weighted LS	
Polynomial models	
Nonlinear models	Linearization
Orthogonal Polynomials and LS Approximation	
Gram-Schmidt process	
Legendre polynomials	
Chebyshev polynomials	
Rational Function Approximation	Pade' rational approximation

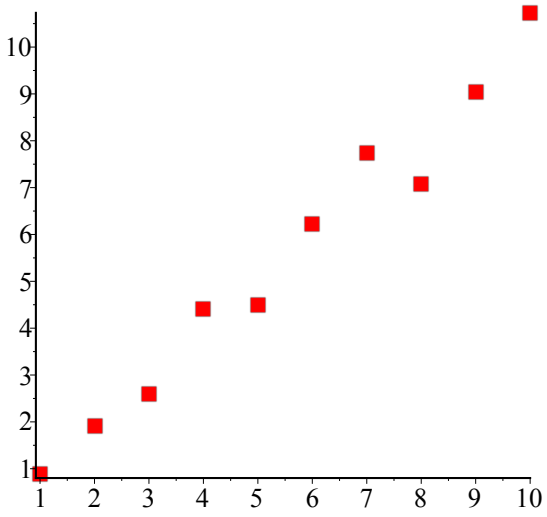
Data representation: $\{(x_i, y_i)\} \rightarrow p(x)$

Interpolation	$p(x)$ passes (interpolates) all data points (Numerical Analysis I)
Approximation	$p(x)$ approximates the data points

8.1. Least Squares Approximation

Example:

```
with(LinearAlgebra) : with(CurveFitting) :
n := 100 :
roll := rand(-n..n) :
m := 10 :
xy := Matrix(m, 2) :
for i to m do
    xy[i, 1] := i;
    xy[i, 2] := i + 1/n·roll( );
end do:
plot(xy, color = red, style = point, symbol = solidbox, symbolsize = 20)
```




```
p := PolynomialInterpolation(xy, x);
```

$$\frac{1507}{12096000}x^9 - \frac{27977}{4032000}x^8 + \frac{36751}{224000}x^7 - \frac{68851}{32000}x^6 + \frac{1975291}{115200}x^5 \qquad \textbf{(1.1)}$$

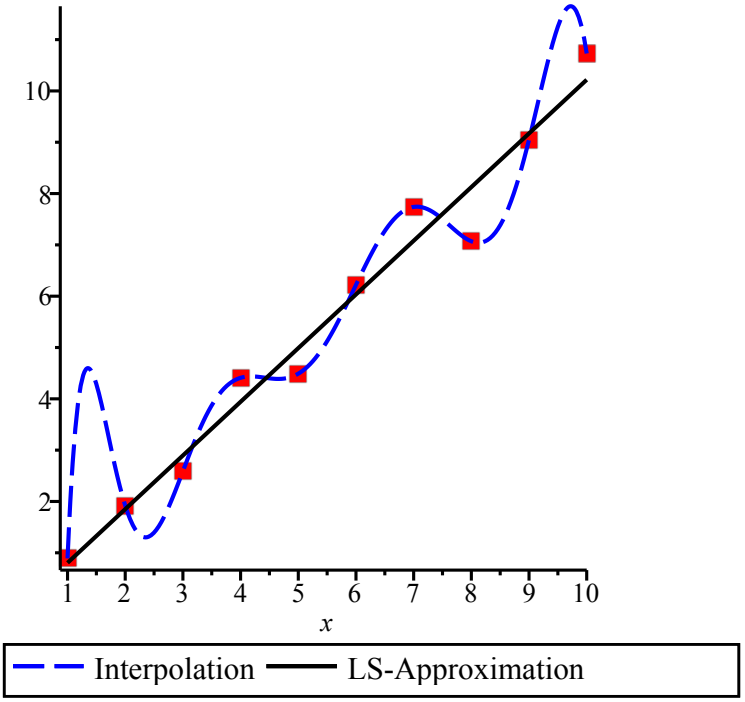
$$-\frac{5471791}{64000}x^4+\frac{199836041}{756000}x^3-\frac{486853651}{1008000}x^2+\frac{39227227}{84000}x$$
$$-\frac{1771}{10}$$

$L := CurveFitting[LeastSquares](xy, x);$

$$-\frac{121}{500}+\frac{523}{500}x \qquad \textbf{(1.2)}$$

▶ **plotting**

$plots[display](pdata, pcurve)$



Linear Least Squares

The least squares approach to this problem involves determining the best approximating line, for which the error is measured in ℓ_2-norm. Let

$$\{(x_i, y_i)\}, \quad 1 \le i \le m,$$

be the data and the best approximating line be expressed as

$$y = a_0 + a_1 x \tag{1}$$

Since the data points may not be on the line, each of the data points may introduce a misfit (error) as follows:

$$\begin{aligned} y_1 &= a_0 + a_1 x_1 + \varepsilon_1 \\ y_2 &= a_0 + a_1 x_2 + \varepsilon_2 \\ &\bullet\bullet\bullet \\ y_m &= a_0 + a_1 x_m + \varepsilon_m \end{aligned} \tag{2}$$

Then, the objective of the **least squares (LS)** method is to determine $\{a_0, a_1\}$ with which the sum of the squares of errors

$$E(a_0, a_1) = \sum_{i=1}^{m} \varepsilon_i^2 = \sum_{i=1}^{m} \left[y_i - (a_0 + a_1 x_i) \right]^2$$

is minimized.

For a minimum to occur, we need

$$\frac{\partial}{\partial a_0} E(a_0, a_1) = 0, \quad \frac{\partial}{\partial a_1} E(a_0, a_1) = 0 \tag{3}$$

that is,

$$0 = \frac{\partial}{\partial a_0} E(a_0, a_1) = 2 \sum_{i=1}^{m} (y_i - a_0 - a_1 x_i)\,(-1)$$

$$0 = \frac{\partial}{\partial a_1} E(a_0, a_1) = 2 \sum_{i=1}^{m} (y_i - a_0 - a_1 x_i)\,(-x_i)$$

These equations are reduced to

$$\sum_{i=1}^{m}\left(a_0 + a_1\, x_i\right) = \sum_{i=1}^{m} y_i$$

$$\sum_{i=1}^{m}\left(a_0 + a_1\, x_i\right) x_i = \sum_{i=1}^{m} x_i\, y_i$$

and finally to the **normal equations**

$$m\, a_0 + \left(\sum_{i=1}^{m} x_i\right) a_1 = \sum_{i=1}^{m} y_i \qquad \textbf{(4)}$$

$$\left(\sum_{i=1}^{m} x_i\right) a_0 + \left(\sum_{i=1}^{m} x_i^2\right) a_1 = \sum_{i=1}^{m} x_i\, y_i \qquad \textbf{(5)}$$

Equations **(4)** and **(5)** equivalently read

$$\begin{bmatrix} m & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i\, y_i \end{bmatrix}$$

Example: For the preceding example, we have

x_i	y_i	x_i^2	$x_i y_i$	$\varepsilon_i = y_i - (a_0 + a_1 x_i)$
1	0.89	1	0.89	0.086
2	1.92	4	3.84	0.070
3	2.59	9	7.77	−0.306
4	4.41	16	17.64	0.468
5	4.49	25	22.45	−0.498
6	6.22	36	37.32	0.186
7	7.74	49	54.18	0.660
8	7.07	64	56.56	−1.056
9	9.05	81	81.45	−0.122
10	10.73	100	107.30	0.512
55	55.11	385	389.40	$\sum \varepsilon_i^2 = 2.43532$

Thus the corresponding system to solve is

$$A \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = b$$

where

$$A := \begin{bmatrix} 10 & 55 \\ 55 & 385 \end{bmatrix}: \quad b := \begin{bmatrix} 55.11 \\ 389.40 \end{bmatrix}:$$

So,

$$\begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = evalf_4\left(A^{-1}.b\right) = \begin{bmatrix} -0.2420 \\ 1.046 \end{bmatrix}.$$

The LS error 2.43532 is a minimum.

The Normal Equation

The normal equation can be derived in a different way. The linear model **(1)** is evaluated at data points $\{(x_i, y_i) : 1 \le i \le m\}$, the resulting equations read

$$\begin{aligned} a_0 + a_1 x_1 &= y_1 \\ a_0 + a_1 x_2 &= y_2 \\ &\bullet\bullet\bullet \\ a_0 + a_1 x_m &= y_m \end{aligned}$$

which can be expressed as an over-determined linear system

$$G \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = y$$

where

$$G = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ : & : \\ 1 & x_m \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_1 \\ : \\ y_m \end{bmatrix}$$

Then the normal equation can be obtained by applying G^T on the linear system:

$$A \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = b$$

It is easy to check that

$$A = G^T G = \begin{bmatrix} m & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{bmatrix} \text{ and } b = G^T y = \begin{bmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \end{bmatrix}$$

The computation of the matrix and the right-hand side:

Note that

$$A = G^T G = \begin{bmatrix} m & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{bmatrix} = \sum_{i=1}^{m} \begin{bmatrix} 1 & x_i \\ x_i & x_i^2 \end{bmatrix}$$

$$= \sum_{i=1}^{m} \begin{bmatrix} 1 \\ x_i \end{bmatrix} \begin{bmatrix} 1 & x_i \end{bmatrix} = \sum_{i=1}^{m} \left(\begin{bmatrix} 1 & x_i \end{bmatrix}^T \begin{bmatrix} 1 & x_i \end{bmatrix} \right)$$

where $\begin{bmatrix} 1 & x_i \end{bmatrix}$ is the i-th row of the matrix G. Thus, the matrix A can be constructed in a point-by-point manner. The column-row multiplication, $\begin{bmatrix} 1 & x_i \end{bmatrix}^T \begin{bmatrix} 1 & x_i \end{bmatrix}$, is called the **outer product**.

Similarly, the right-hand side can be constructed pointwisely as follows.

$$b = G^T y = \begin{bmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \end{bmatrix} = \sum_{i=1}^{m} \begin{bmatrix} y_i \\ x_i y_i \end{bmatrix} = \sum_{i=1}^{m} \left(\begin{bmatrix} 1 & x_i \end{bmatrix}^T y_i \right)$$

Notes:

- The pointwise construction of the normal equation is convenient when either points are first to be searched or weights are applied depending on the point location.
- The idea is applicable for other LS models as well.

Example: Let $x_i = i \cdot h$, for some $h > 0$, and $u_i = u(x_i)$. Find a numerical differentiation formula approximating $u_x(x_i)$.

Solution.

Let us first try to find the least-squares numerical differentiation involving the **nearest two** function values. Then, since

$$u_{i+1} = u_i + h\, u_x + \mathrm{O}(h^2)$$
$$u_{i-1} = u_i - h\, u_x + \mathrm{O}(h^2)$$

the linear system reads

$$\begin{bmatrix} h \\ -h \end{bmatrix} [\alpha] = \begin{bmatrix} u_{i+1} - u_i \\ u_{i-1} - u_i \end{bmatrix}$$

where α denotes an approximation of $u_x(x_i)$. Application of $G^T = [h \quad -h]$ leads to

$$2h^2 \alpha = h\left(u_{i+1} - u_i\right) - h\left(u_{i-1} - u_i\right) \tag{3.1}$$

and therefore

$$\alpha = \frac{1}{2} \frac{u_{i+1} - u_{i-1}}{h} \tag{3.2}$$

which is a second-order approximation of $u_x(x_i)$.

Now, we will find the least-squares numerical differentiation involving the **nearest four** function values. Then, since

$$u_{i+1} = u_i + h\, u_x + \mathrm{O}(h^2)$$
$$u_{i-1} = u_i - h\, u_x + \mathrm{O}(h^2)$$
$$u_{i+2} = u_i + 2h\, u_x + \mathrm{O}(h^2)$$
$$u_{i-2} = u_i - 2h\, u_x + \mathrm{O}(h^2)$$

the linear system reads

$$G[\alpha] = \begin{bmatrix} h \\ -h \\ 2h \\ -2h \end{bmatrix} [\alpha] = \begin{bmatrix} u_{i+1} - u_i \\ u_{i-1} - u_i \\ u_{i+2} - u_i \\ u_{i-2} - u_i \end{bmatrix} = r$$

Now, we apply a weight

$$W = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then, the algebraic system becomes

$$WG[\alpha] = \begin{bmatrix} 2h \\ -2h \\ 2h \\ -2h \end{bmatrix} [\alpha] = \begin{bmatrix} 2(u_{i+1} - u_i) \\ 2(u_{i-1} - u_i) \\ u_{i+2} - u_i \\ u_{i-2} - u_i \end{bmatrix} = Wr$$

Application of $(WG)^T = [2h \;\; -2h \;\; 2h \;\; -2h]$ leads to

$$16h^2 \alpha = 2h(2u_{i+1} - 2u_{i-1} + u_{i+2} - u_{i-2}) \qquad \textbf{(3.3)}$$

which implies that

$$\alpha = \frac{u_{i+2} + 2u_{i+1} - 2u_{i-1} - u_{i-2}}{8h}$$

which is a second-order approximation of $u_x(x_i)$, worse than **(3.2)** though.

Weighted Least Squares

Given data $\{(x_i, y_i)\}$, $1 \le i \le m$, the best-fitting curve can be found by solving an over-determined algebraic system

$$G\theta = y \tag{6}$$

where $G \in \mathbb{R}^{m \times n}$, $m > n$, and θ is the coefficient vector to be calculated. The associated **LS problem** is formulated as

$$\min_{\theta} \| G\theta - y \|_2^2$$

When certain data points are more important or more reliable than the others, one may try to compute the coefficient vector with larger weights on more reliable data points. The **weighted least squares** method is an LS method which involves a weight. The weight is often given as a diagonal matrix

$$W = diag(w_1, w_2, \bullet\bullet\bullet, w_m)$$

When the weight is applied, the system **(6)** can be written as

$$WG\theta = Wy \tag{7}$$

and therefore its normal equation turns out to be

$$G^T W^2 G\theta = G^T W^2 y \tag{8}$$

Example: Given data, find the LS line with and without a weight. When a weight is applied, weigh the first and the last data point by 1/4.

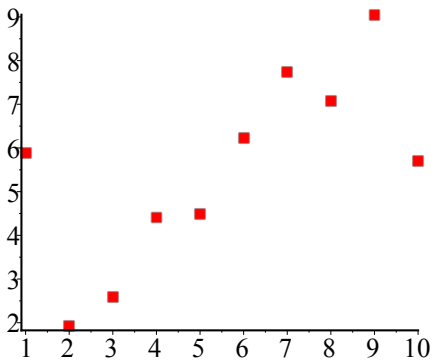
$$\begin{bmatrix} 1. & 2. & 3. & 4. & 5. & 6. & 7. & 8. & 9. & 10. \\ 5.89 & 1.92 & 2.59 & 4.41 & 4.49 & 6.22 & 7.74 & 7.07 & 9.05 & 5.7 \end{bmatrix}$$

Solution:

$$DATA := \begin{bmatrix} 1. & 2. & 3. & 4. & 5. & 6. & 7. & 8. & 9. & 10. \\ 5.89 & 1.92 & 2.59 & 4.41 & 4.49 & 6.22 & 7.74 & 7.07 & 9.05 & 5.7 \end{bmatrix}:$$

$$xy := DATA^{\%T}:$$

$$plot(xy, color = red, style = point, symbol = solidbox, symbolsize = 20)$$



$L := CurveFitting[LeastSquares](xy, x)$

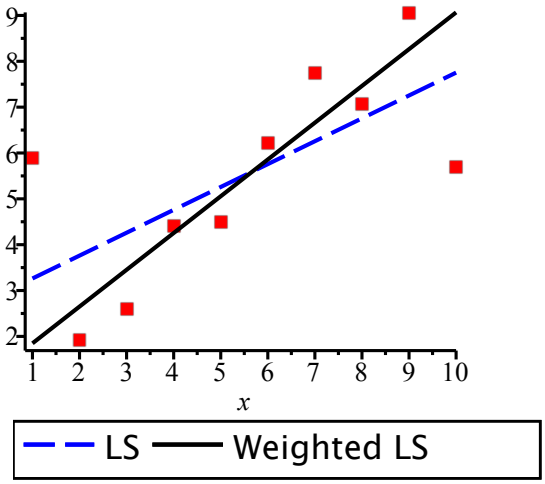
$$2.76400000000000 + 0.498909090909091\,x \quad \textbf{(4.1)}$$

$LW := CurveFitting[LeastSquares](xy, x, weight = [1/4, 1, 1, 1, 1, 1, 1, 1, 1, 1/4])$

$$1.04666948793906 + 0.801942446043165\,x \quad \textbf{(4.2)}$$

▶ plotting

$plots[display](pdata, pcurve)$



Polynomial Least Squares

The general problem of approximating a set of data $\{(x_i, y_i) : i = 1, 2, \cdots, m\}$, with an algebraic polynomial

$$P_n(x) = a_0 + a_1 x + \cdots + a_n x^n = \sum_{j=0}^{n} a_j x^j,$$

of degree $n \leq m - 1$, using the least squares procedure, is handled similarly. We choose the constants $a_0, a_1, \cdots, a_n$ to minimize the LS error $E = E(a_0, a_1, \cdots, a_n)$, where

$$E = \sum_{i=1}^{m} \left(y_i - P_n(x_i)\right)^2 \qquad \textbf{(9)}$$

Some algebra leads us to

$$\begin{aligned} E &= \sum_{i=1}^{m} y_i^2 - 2\sum_{i=1}^{m} P_n(x_i) y_i + \sum_{i=1}^{m} P_n(x_i)^2 \\ &= \sum_{i=1}^{m} y_i^2 - 2\sum_{i=1}^{m} \left(\sum_{j=0}^{n} a_j x_i^j\right) y_i + \sum_{i=1}^{m} \left(\sum_{j=0}^{n} a_j x_i^j\right)^2 \\ &= \sum_{i=1}^{m} y_i^2 - 2\sum_{j=0}^{n} a_j \left(\sum_{i=1}^{m} x_i^j y_i\right) + \sum_{j=0}^{n}\sum_{k=0}^{n} a_j a_k \left(\sum_{i=1}^{m} x_i^{j+k}\right) \end{aligned}$$

As in the linear case, for E to be minimized it is necessary that

$$\frac{\partial}{\partial a_j} E = 0, \text{ for each } j = 0, 1, \cdots, n$$

Indeed,

$$0 = \frac{\partial}{\partial a_j} E = -2\left(\sum_{i=1}^{m} x_i^j y_i\right) + 2\sum_{k=0}^{n} a_k \left(\sum_{i=1}^{m} x_i^{j+k}\right) \qquad (0 \leq j \leq n)$$

which implies that, for $0 \leq j \leq n$,

$$\sum_{k=0}^{n} a_k \left(\sum_{i=1}^{m} x_i^{(j+k)} \right) = \sum_{i=1}^{m} x_i^j y_i \qquad \textbf{(10)}$$

It is helpful to write the equations as follows:

$$\begin{bmatrix} m = \sum_{i=1}^{m} 1 & \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 & \cdots & \sum_{i=1}^{m} x_i^n \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 & \sum_{i=1}^{m} x_i^3 & \cdots & \sum_{i=1}^{m} x_i^{n+1} \\ \cdots & \cdots & & \ddots & \cdots \\ \sum_{i=1}^{m} x_i^n & \sum_{i=1}^{m} x_i^{n+1} & \sum_{i=1}^{m} x_i^{n+2} & \cdots & \sum_{i=1}^{m} x_i^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \\ \vdots \\ \sum_{i=1}^{m} x_i^n y_i \end{bmatrix}$$

Example: Find the best approximating quadratic curve for the data

$$xy := \begin{bmatrix} 1 & 0.89 \\ 2 & 2.61 \\ 3 & 2.17 \\ 4 & 4.79 \\ 5 & 4.55 \\ 6 & 6.70 \\ 7 & 6.82 \\ 8 & 7.24 \\ 9 & 8.89 \\ 10 & 9.92 \end{bmatrix} :$$

Solution:

$m := 10$:

$$A := \begin{bmatrix} \text{add}(1, i = 1..m) & \text{add}(xy[i, 1], i = 1..m) & \text{add}(xy[i, 1]^2, i = 1..m) \\ \text{add}(xy[i, 1], i = 1..m) & \text{add}(xy[i, 1]^2, i = 1..m) & \text{add}(xy[i, 1]^3, i = 1..m) \\ \text{add}(xy[i, 1]^2, i = 1..m) & \text{add}(xy[i, 1]^3, i = 1..m) & \text{add}(xy[i, 1]^4, i = 1..m) \end{bmatrix}$$

$$\begin{bmatrix} 10 & 55 & 385 \\ 55 & 385 & 3025 \\ 385 & 3025 & 25333 \end{bmatrix} \quad \textbf{(6.1)}$$

$$b := \begin{bmatrix} \text{add}(xy[i, 2], i = 1..m) \\ \text{add}(xy[i, 1] \cdot xy[i, 2], i = 1..m) \\ \text{add}(xy[i, 1]^2 \cdot xy[i, 2], i = 1..m) \end{bmatrix} = \begin{bmatrix} 54.58 \\ 379.60 \\ 2972.08 \end{bmatrix}$$

$$c := A^{-1}.b$$

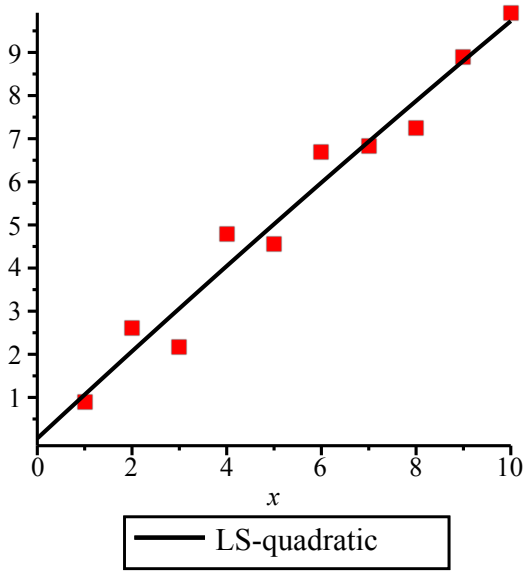
$$\begin{bmatrix} 0.0490000 \\ 1.02004546 \\ -0.005227272 \end{bmatrix} \quad \textbf{(6.2)}$$

$$LS2 := x \rightarrow c[1] + c[2] \cdot x + c[3] \cdot x^2 :$$

$$LS2(x) = 0.0490000 + 1.02004546\, x - 0.005227272\, x^2$$

▶ **plotting**

plots[*display*](*pdata*, *pcurve*)



Note that the algebraic system in **(10)** can be expressed as

$$A\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = G^T G \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = G^T y = b$$

where $G \in \mathbb{R}^{m \times (n+1)}$, $m > n+1$,

$$G = \begin{bmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & & \ddots & \vdots \\ 1 & x_m & \cdots & x_m^n \end{bmatrix}$$

Self study: Check

$$A = G^T G = \sum_{i=1}^{m} \begin{bmatrix} 1 & x_i & \cdots & x_i^n \\ x_i & x_i^2 & \cdots & x_i^{n+1} \\ \vdots & & \ddots & \vdots \\ x_i^n & x_i^{n+1} & \cdots & x_i^{2n} \end{bmatrix}$$

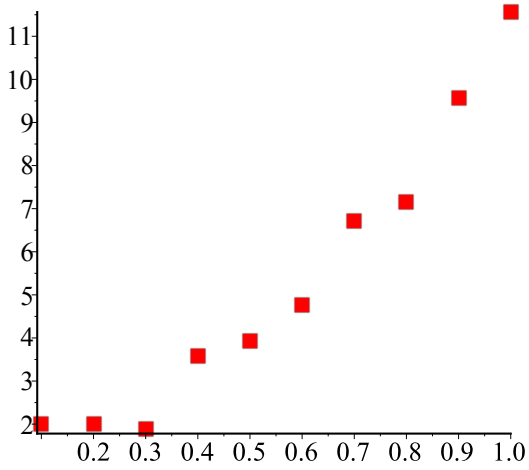
and

$$b = G^T y = \sum_{i=1}^{m} \left(\begin{bmatrix} 1 & x_i & \cdots & x_i^n \end{bmatrix}^T y_i \right)$$

LS for Nonlinear Models

Example: Find the best fitting curve of the form $y = c\, e^{d\,x}$ for the data

0.1	1.9940
0.2	2.0087
0.3	1.8770
0.4	3.5783
0.5	3.9203
0.6	4.7617
0.7	6.7246
0.8	7.1491
0.9	9.5777
1.0	11.5625



Solution:

Applying the natural log (ln) function to $y = c\, e^{d\,x}$ gives

$$\ln(y) = \ln(c) + d\,x$$

Using the change of variables

$$Y = \ln(y), \quad a_0 = \ln(c), \quad a_1 = d, \quad X = x$$

the above equation reads

$$Y = a_0 + a_1 X \tag{8.1}$$

for which one can apply the linear LS procedure. The data to be used is

$$(X_i, Y_i) = (x_i, \ln(y_i)) \tag{8.2}$$

```
xlny := Matrix(m, 2) :
for i to m do
    xlny[i, 1] := xy[i, 1];
    xlny[i, 2] := ln(xy[i, 2]);
end do:
```

$L := CurveFitting[LeastSquares](xlny, x, curve = a + b\,x);$

$$0.295704647799999 + 2.15307406543637\,x \quad \textbf{(8.3)}$$

Thus,

```
> c := exp(0.295704647799999)
```

$$c := 1.344073123 \quad \textbf{(8.4)}$$

```
> d := 2.15307406543637
```

$$d := 2.15307406543637 \quad \textbf{(8.5)}$$

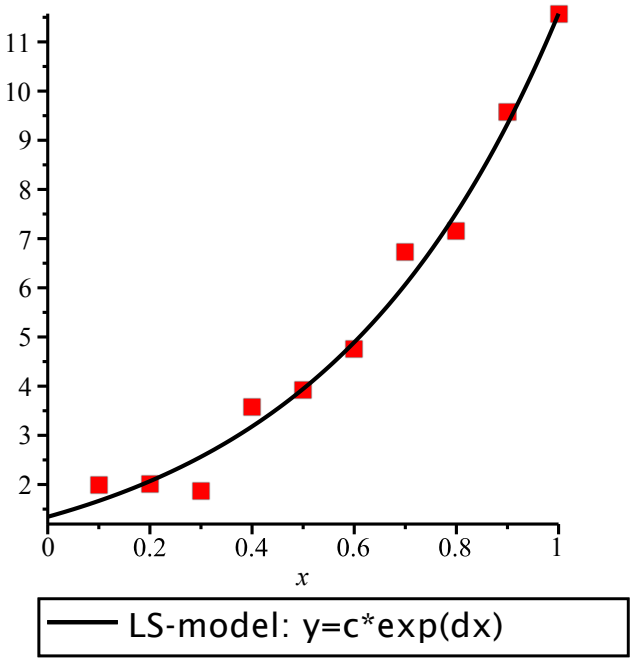
$LSe := x \rightarrow c \cdot \exp(d\,x)$:

$LSe(x)$

$$1.344073123\,e^{2.15307406543637\,x} \quad \textbf{(8.6)}$$

▶ plotting

$plots[display](pdata, pcurve)$



Linearization

Model	Change of Variables	Linear Model
$y = \frac{1}{A + B\,x}$	$Y = \frac{1}{y};\quad X = x$	$Y = A + B\,X$
$y = C\,\mathrm{e}^{B\,x}$	$Y = \ln(y);\ X = x;\ A = \ln(C)$	$Y = A + B\,X$
$y = A + \frac{B}{x} + \frac{C}{x^2}$	$Y = y;\ X = \frac{1}{x}$	$Y = A + B\,X + C\,X^2$
$y = \frac{1}{A + B\ln(x)}$	$Y = \frac{1}{y};\ X = \ln(x)$	$Y = A + B\,X$

The above table contains just a few examples of linearization; for other nonlinear models, use your imagination and creativity.

Empty Page

8.2. Rational Function Approximations

The class of algebraic polynomials:

Advantages

- There are a sufficient number of polynomials to approximate any continuous function on a closed interval to within an arbitrary tolerance.
- Polynomials are easy to evaluate.
- Their derivatives and integrals exist and are easily determined.

Disadvantage

- Tendency for oscillation, which often causes the error bound to significantly exceed the average approximation error.

Definition: A **rational function *r* of degree *N*** has the form

$$r(x) = \frac{p(x)}{q(x)} \tag{1}$$

where $p(x)$ and $q(x)$ are polynomials whose degrees sum to N.

Pade Approximation:

Pade rational function of degree $N = m + n$, approximating $f(x)$, has the form

$$f(x) \approx R_{n,m}(x) = \frac{p(x)}{q(x)} = \frac{p_0 + p_1 x + \cdots + p_n x^n}{1 + q_1 x + \cdots + q_m x^m}$$

Determination of $\{p_0, p_1, \cdots, p_n, q_1, \cdots, q_m\}$: To satisfy

$$D^{(k)}(f)(0) - D^{(k)}\left(R_{n,m}\right)(0) = 0 \tag{2}$$

for $k = 0, 1, \cdots, N$.

Let f has the Maclaurin series expansion

$$f(x) = \sum_{i=0}^{\infty} a_i x^i \tag{3}$$

Then,

$$f(x) - R_{n,m}(x) = \frac{\left(\sum_{i=0}^{\infty} a_i x^i\right)\left(\sum_{i=0}^{m} q_i x^i\right) - \left(\sum_{i=0}^{n} p_i x^i\right)}{q(x)} \tag{4}$$

Our object is to choose the constants $\{p_0, p_1, \cdots, p_n, q_1, \cdots, q_m\}$ so that **(2)** is satisfied, which is equivalent to $(f - R_{m,n})$ having a zero of multiplicity $N+1$ at $x = 0$. As a consequence, we choose the constants so that the numerator on the right side of **(4)**,

$$\left(\sum_{i=0}^{\infty} a_i x^i\right)\left(\sum_{i=0}^{m} q_i x^i\right) - \left(\sum_{i=0}^{n} p_i x^i\right) \tag{5}$$

has no terms of degree $\leq N$.

In order to simplify the notation, we define

$$q_{m+1} = q_{m+2} = \cdots = q_N = 0$$
$$p_{n+1} = p_{n+2} = \cdots = p_N = 0$$

Then the coefficient of x^k in **(5)** can be expresses as

$$\sum_{i=0}^{k} a_i q_{k-i} - p_k \tag{6}$$

Thus the rational function for Pade approximation results from the solution of $N+1$ linear equations

$$\sum_{i=0}^{k} a_i q_{k-i} = p_k \tag{7}$$

for $k = 0, 1, 2, \cdots, N$.

Example: Find the Pade approximation to e^{-x} of degree 5 with $n = 3$ and $m = 2$.

Solution

$T := taylor(e^{-x}, x = 0, 6)$

$$1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 - \frac{1}{120}x^5 + O(x^6) \quad \textbf{(2.1)}$$

$T5 := convert(T, polynom)$

$$1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 - \frac{1}{120}x^5 \quad \textbf{(2.2)}$$

$A := Array(0..5)$:
for i **from** 0 **to** 5 **do**; $A[i] := coeff(T5, x, i)$; **end do**:
$q_0 := 1$:
$q_3 := 0$: $q_4 := 0$: $q_5 := 0$:
$p_4 := 0$: $p_5 := 0$:

$x0 := add(A[i] \cdot q_{0-i}, i = 0..0) = p_0 = 1 = p_0$

$x1 := add(A[i] \cdot q_{1-i}, i = 0..1) = p_1 = -1 + q_1 = p_1$

$x2 := add(A[i] \cdot q_{2-i}, i = 0..2) = p_2 = \frac{1}{2} + q_2 - q_1 = p_2$

$x3 := add(A[i] \cdot q_{3-i}, i = 0..3) = p_3 = -\frac{1}{6} - q_2 + \frac{1}{2}q_1 = p_3$

$x4 := add(A[i] \cdot q_{4-i}, i = 0..4) = p_4 = \frac{1}{24} + \frac{1}{2}q_2 - \frac{1}{6}q_1 = 0$

$x5 := add(A[i] \cdot q_{5-i}, i = 0..5) = p_5 = -\frac{1}{120} - \frac{1}{6}q_2 + \frac{1}{24}q_1 = 0$

$solve(\{x0, x1, x2, x3, x4, x5\}, \{q_1, q_2, p_0, p_1, p_2, p_3\})$

$$\left\{p_0 = 1, p_1 = -\frac{3}{5}, p_2 = \frac{3}{20}, p_3 = -\frac{1}{60}, q_1 = \frac{2}{5}, q_2 = \frac{1}{20}\right\} \quad \textbf{(2.3)}$$

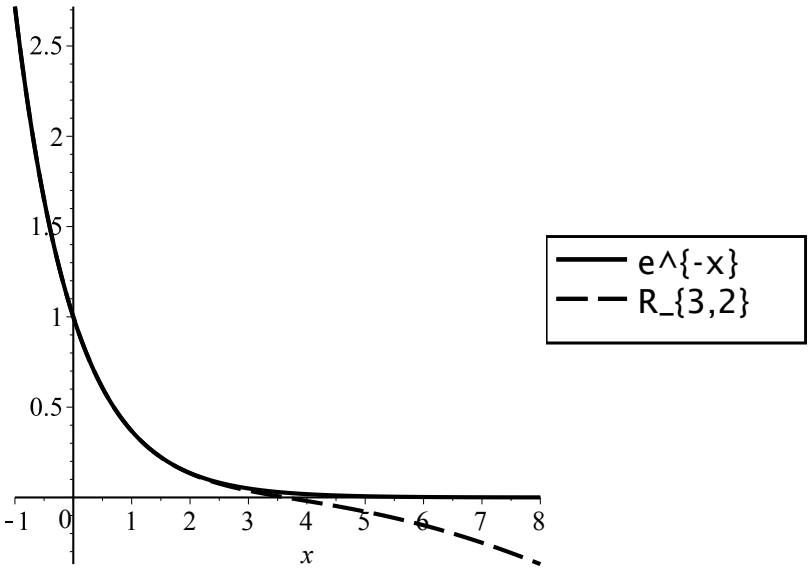
So the Pade approximation of e^{-x} is

$$R_{3,2}(x) = \frac{1 - \frac{3}{5}x + \frac{3}{20}x^2 - \frac{1}{60}x^3}{1 + \frac{2}{5}x + \frac{1}{20}x^2} \tag{2.4}$$

Use of built-in command:
convert(*T*, *ratpoly*, 3, 2)

$$\frac{1 - \frac{3}{5}x + \frac{3}{20}x^2 - \frac{1}{60}x^3}{1 + \frac{2}{5}x + \frac{1}{20}x^2} \tag{2.5}$$

▶ **plotting**



 # Maple code:

```
# Pade Rational Approximation

PadeRatApprox := proc(f, n, m)
    local i, j, N, p, q, eq, PQ, T, a, R;
    N := m + n;
    T := convert(taylor(f, x = 0, N + 1), polynom);
    for i from 0 to N do a[i] := coeff(T, x, i); end do;
    q[0] := 1;
    for i from m + 1 to N do q[i] := 0; end do;
    for i from n + 1 to N do p[i] := 0; end do;
    for i from 0 to N do
        eq[i] := add(a[j]*q[i - j], j = 0 .. i) = p[i];
    end do;
    PQ := solve({seq(eq[i], i = 0 .. N)}, {seq(p[i], i = 0 .. n), seq(q[i], i = 1
    .. m)});
    R := x -> add(rhs(PQ[1 + i])*x^i, i = 0 .. n)/(q[0] + add(rhs(PQ[n + 1 + i])*x^i, i = 1 .. m));
    return R;
end proc:

R32 := PadeRatApprox(exp(-x), 3, 2):
> R32(x)
```

$$\frac{1 - \frac{3}{5}x + \frac{3}{20}x^2 - \frac{1}{60}x^3}{1 + \frac{2}{5}x + \frac{1}{20}x^2} \tag{8}$$

Empty Page

Homework:
8. Approximation Theory

#1. Given data

x_i	0.2	0.4	0.6	0.8	1.	1.2	1.4	1.6	1.8	2.
y_i	1.88	2.13	1.76	2.78	3.23	3.82	6.13	7.22	6.66	9.07

a. Plot the data (scattered point plot)
b. Decide which curve fits the data best.
c. Implement an LS code to find the curve.
d. Plot the curve superposed over the point plot.

#2. Determine Pade approximation of degree 6 for $f(x) = \sin(x)$, and compare the results at $x_i = 0.2i,\ i = 0, 1, \cdots 5$, with $f(x)$ and with its sixth Maclaurin polynomial

$$convert(taylor(\sin(x), x = 0, 7), polynom) = x - \frac{1}{6}x^3 + \frac{1}{120}x^5$$

a. with $n = 2,\ m = 4$
b. with $n = 3,\ m = 3$
c. with $n = 4,\ m = 2$

(You may use Maple functions; however, you should not use any built-in functions which produce the results immediately.)

Empty Page

9. Approximating Eigenvalues

In This Chapter:

Topics	Applications/Properties
Linear Algebra and Eigenvalues	
Orthogonal Matrices and Similarity Transformations	
The Power Method	
Symmetric power method	Symmetric matrices
The inverse power method	
Householder's Method	
QR Factorization	
Singular Value Decomposition	

9.1. The Power Method

To apply the Power method, we assume that $A \in \mathbb{R}^{n \times n}$ has
• n eigenvalues $\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$ and
• n associated eigenvectors $\{v^1, v^2, \cdots, v^n\}$ which are linearly independent.
Moreover, we assume that A has precisely one eigenvalue, λ_1, that is largest in magnitude, so that

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$$

The Power method approximate the largest eigenvalue λ_1 and its associated eigenvector v^1.

For any vector $x \in \mathbb{R}^n$, the fact that $\{v^1, v^2, \cdots, v^n\}$ is linearly independent implies that constants $\{\beta_1, \beta_2, \cdots, \beta_n\}$ exist with

$$x = \sum_{j=1}^{n} \beta_j v^j \tag{1}$$

Multiplying both sides of Equation (1) by A, A^2, $\cdots$ gives

$$A x = A\left(\sum_{j=1}^{n} \beta_j v^j\right) = \sum_{j=1}^{n} \beta_j A v^j = \sum_{j=1}^{n} \beta_j \lambda_j v^j$$

$$A^2 x = A\left(\sum_{j=1}^{n} \beta_j \lambda_j v^j\right) = \sum_{j=1}^{n} \beta_j \lambda_j A v^j = \sum_{j=1}^{n} \beta_j \lambda_j^2 v^j$$

$$\cdots$$

In general,

$$A^k x = \sum_{j=1}^{n} \beta_j \lambda_j^k v^j \tag{2}$$

which gives

$$A^k x = \lambda_1^k \left(\sum_{j=1}^{n} \beta_j \left(\frac{\lambda_j}{\lambda_1} \right)^k v^j \right) \tag{3}$$

Since $|\lambda_1| > |\lambda_j|, \quad 2 \le j \le n,$ we have limit $\lim_{k \to \infty} |\lambda_j / \lambda_1|^k = 0$, and

$$\lim_{k \to \infty} A^k x = \lim_{k \to \infty} \lambda_1^k \beta_1 v^1 \tag{4}$$

Note that the sequence in **(4)** converges to 0 if $|\lambda_1| < 1$ and diverges if $|\lambda_1| > 1$ provided that $\beta_1 \neq 0$. Thus the entries of $A^k x$ will grow with k if $|\lambda_1| > 1$ and will go to 0 if $|\lambda_1| < 1$; in either case, it is hard to decide the largest eigenvalue λ_1 and its associated eigenvector v^1. To take care of that possibility, we scale $A^k x$ in an appropriate manner to ensure that the limit in Equation **(4)** is finite and nonzero.

The Power Method:

Given $x \neq 0$, let $x^0 = x / \|x\|_\infty$

$$y^1 = A x^0 \qquad \mu_1 = \|y^1\|_\infty \qquad x^1 = y^1 / \mu_1$$

$$y^2 = A x^1 \qquad \mu_2 = \|y^2\|_\infty \qquad x^2 = y^2 / \mu_2$$

$$\cdots$$

$$y^k = A x^{k-1} \qquad \mu_k = \|y^k\|_\infty \qquad x^k = y^k / \mu_k$$

$$\cdots$$

Properties:

First of all,

$$\left\| x^k \right\|_\infty = 1, \text{ for all } k \geq 0$$

Then, it follows from Equation **(4)** that for k sufficiently large,

$$x^k = \left(\prod_{j=1}^{k} \mu_j \right)^{-1} A^k x^0 \approx \left(\prod_{j=1}^{k} \mu_j \right)^{-1} \lambda_1^k \beta_1 v^1$$

and

$$x^{k+1} = \left(\prod_{j=1}^{k+1} \mu_j \right)^{-1} A^{k+1} x^0 \approx \left(\prod_{j=1}^{k+1} \mu_j \right)^{-1} \lambda_1^{k+1} \beta_1 v^1$$

Note that $\left\| x^k \right\|_\infty = \left\| x^{k+1} \right\|_\infty = 1$. Comparison of the above two equations reads

$$\left(\frac{\lambda_1}{\mu_{k+1}} \right) \approx 1 \qquad \textbf{(2.1)}$$

Claim: Let $\left\{x^k, \mu_k\right\}$ be sequences produced by the Power method. Then,

$$x^k \to v^1, \quad \mu_k \to \lambda_1, \quad \text{as} \quad k \to \infty,$$

with the rate of convergence given by

$$O\left(\left|\lambda_2/\lambda_1\right|^k\right), \quad \text{as} \quad k \to \infty$$

Example: The matrix

$$A := \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}:$$

has eigenvalues and eigenvectors as follows

$$LinearAlgebra[Eigenvectors](A) = \begin{bmatrix} 6 \\ 3 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 & -2 & 0 \\ -1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Verify that the sequences produced by the power method converge to the largest eigenvalue and its associated eigenvector.

Solution:

```
> x0 := ⟨1, 0, 0⟩
```

$$x0 := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.1}$$

```
for k from 1 to 10 do
    if (k = 1) then x := x0; end if;
    y := A x;
    muk := LinearAlgebra[VectorNorm](y, infinity);
    x := 1/muk y;
```

```
print( `k=`, k, `muk=`, evalf[8](muk), evalf[4](x^%T), evalf[4](abs(6
- muk)) );
end do:
```

```
k=, 1, muk=, 4., [ 1.  -0.2500  0.2500 ], 2.
k=, 2, muk=, 4.5000000, [ 1.  -0.5000  0.5000 ], 1.500
k=, 3, muk=, 5., [ 1.  -0.7000  0.7000 ], 1.
k=, 4, muk=, 5.4000000, [ 1.  -0.8333  0.8333 ], 0.6000
k=, 5, muk=, 5.6666667, [ 1.  -0.9118  0.9118 ], 0.3333
k=, 6, muk=, 5.8235294, [ 1.  -0.9545  0.9545 ], 0.1765
k=, 7, muk=, 5.9090909, [ 1.  -0.9769  0.9769 ], 0.09091
k=, 8, muk=, 5.9538462, [ 1.  -0.9884  0.9884 ], 0.04615
k=, 9, muk=, 5.9767442, [ 1.  -0.9942  0.9942 ], 0.02326
k=, 10, muk=, 5.9883268, [ 1.  -0.9971  0.9971 ], 0.01167
```

(3.2)

Notice that $\left|\lambda_1 - \mu_{k+1}\right| \approx \frac{1}{2}\left|\lambda_1 - \mu_k\right|$, where $\left|\lambda_2/\lambda_1\right| = 1/2.$

Symmetric Power Method

When $A \in \mathbb{R}^{n \times n}$ is symmetric, the scaling step in the Power method can be carried out by the ℓ_2 norm instead of the ℓ_∞ norm, which makes the method converge more rapidly, more precisely,

$$O\left(\left|\lambda_2/\lambda_1\right|^{2k}\right), \text{ as } k \to \infty$$

In this case, since $x^k \approx v^1$ with $\left\|x^k\right\|_2 = 1$ for large k,

$$\mu_k = x^k \bullet y^{k+1} = x^k \bullet \left(A\,x^k\right) \approx v^1 \bullet \left(\lambda_1\, v^1\right) = \lambda_1$$

That is, $\mu_k = x^k \bullet y^{k+1}$ is an approximation converging to λ_1.

Example: Symmetric Power Method

$$A := \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}:$$

```
> x0 := ⟨1, 0, 0⟩
```

$$x0 := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \textbf{(4.1)}$$

```
for k from 1 to 10 do
    if (k = 1) then x := x0; end if;
    y := A x;
    muk := x^%T.y;
    ynorm := LinearAlgebra[VectorNorm](y, 2);
    x := 1/ynorm y;
    print(`k=`, k, `muk=`, evalf[8](muk), evalf[4](x^%T), evalf[4](abs(6
```

$-muk))\Big);$

end do:

$k=, 1, muk=, 4., \left[\begin{array}{ccc} 0.9427 & -0.2357 & 0.2357 \end{array}\right], 2.$

$k=, 2, muk=, 5., \left[\begin{array}{ccc} 0.8163 & -0.4082 & 0.4082 \end{array}\right], 1.$

$k=, 3, muk=, 5.6666667, \left[\begin{array}{ccc} 0.7105 & -0.4974 & 0.4974 \end{array}\right], 0.3333$

$k=, 4, muk=, 5.9090909, \left[\begin{array}{ccc} 0.6471 & -0.5393 & 0.5393 \end{array}\right], 0.09091$

$k=, 5, muk=, 5.9767442, \left[\begin{array}{ccc} 0.6127 & -0.5587 & 0.5587 \end{array}\right], 0.02326$

$k=, 6, muk=, 5.9941520, \left[\begin{array}{ccc} 0.5950 & -0.5679 & 0.5679 \end{array}\right], 0.005848$

$k=, 7, muk=, 5.9985359, \left[\begin{array}{ccc} 0.5863 & -0.5728 & 0.5728 \end{array}\right], 0.001464$

$k=, 8, muk=, 5.9996338, \left[\begin{array}{ccc} 0.5819 & -0.5751 & 0.5751 \end{array}\right], 0.0003662$

$k=, 9, muk=, 5.9999084, \left[\begin{array}{ccc} 0.5792 & -0.5760 & 0.5760 \end{array}\right], 0.00009155$

$k=, 10, muk=, 5.9999771, \left[\begin{array}{ccc} 0.5781 & -0.5764 & 0.5764 \end{array}\right], 0.00002289$ **(4.2)**

Now, $\left|\lambda_1 - \mu_{k+1}\right| \approx \frac{1}{4}\left|\lambda_1 - \mu_k\right|$, where $\left|\lambda_2/\lambda_1\right| = 1/2$

Accuracy Analysis

Let μ be be a real number that approximates an eigenvalue of a symmetric matrix A and x an associated eigenvector. then $Ax - \mu x$ is approximately the zero vector. The following theorem shows a relation between the norm of the vector and the accuracy of μ to the eigenvalue.

Theorem: Suppose $A \in \mathbb{R}^{n \times n}$ is symmetric, having eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$. Then, for $\|x\|_2 = 1$ and $\mu \in \mathbb{R}$,

$$\min_{1 \le i \le n} |\lambda_i - \mu| \le \|Ax - \mu x\|_2$$

Proof. Let $\{v_1, v_2, ..., v_n\}$ form an orthonormal set of eigenvectors of A associated with the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$. Then there is a unique set of constants $\{c_1, c_2, ..., c_n\}$ such that

$$x = \sum_{i=1}^{n} c_i v_i$$

Thus,

$$\|Ax - \mu x\|_2^2 = \left\| \sum_{i=1}^{n} c_i (Av_i - \mu v_i) \right\|_2^2 = \left\| \sum_{i=1}^{n} c_i (\lambda_i - \mu) v_i \right\|_2^2$$

$$= \sum_{i=1}^{n} c_i^2 (\lambda_i - \mu)^2 \ge \min_{1 \le i \le n} |\lambda_i - \mu|^2 \sum_{i=1}^{n} c_i^2$$

But, $\sum_{i=1}^{n} c_i^2 = \|x\|_2^2 = 1$, which completes the proof.

Note: The above theorem implies that the approximated eigenvalue represents one of the eigenvalues of A, with accuracy that is in the same order as the stopping tolerance.

Inverse Power Method:

Some applications require to find an eigenvalue of A near a prescribed value q. Inverse Power method is a variant of the Power method to solve such a problem.

We begin with the eigenvalues and eigenvectors of $(A - q\,I)^{-1}$. When

$$A\,v^i = \lambda_i\,v^i \tag{5}$$

it is easy to see that

$$(A - q\,I)\,v^i = \left(\lambda_i - q\right)v^i$$

Thus, we obtain

$$(A - q\,I)^{-1}\,v^i = \frac{1}{\lambda_i - q}\,v^i$$

That is, the eigenvalues of $(A - q\,I)^{-1}$, $q \neq \lambda_i$, are

$$\frac{1}{\lambda_1 - q}, \frac{1}{\lambda_2 - q}, \bullet\bullet\bullet, \frac{1}{\lambda_n - q}$$

with these same eigenvectors $\{v^1, v^2, \bullet\bullet\bullet, v^n\}$.

Applying the Power method to $(A - q\,I)^{-1}$ gives

$$y^k = (A - q\,I)^{-1}\,x^{k-1}$$
$$ynorm = \left\| y^k \right\|_\infty$$
$$\lambda = \frac{1}{ynorm} + q$$
$$x^k = \frac{y^k}{ynorm} \tag{6}$$

Example: Inverse Power Method

$$A := \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}:$$

```
> x0 := <1, 0, 0>
```

$$x0 := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \textbf{(5.1)}$$

```
> q := 5/2
```

$$q := \frac{5}{2} \quad \textbf{(5.2)}$$

```
Id := Matrix(3, 3, shape = identity):

for k from 1 to 10 do
    if (k = 1) then x := x0; end if;
    y := (A - q Id)^(-1) . x;
    ynorm := LinearAlgebra[VectorNorm](y, infinity);
    x := 1/ynorm y;
    muk := 1/ynorm + q;
    print(`k=`, k, `muk=`, evalf[8](muk), evalf[4](x^%T), evalf[8](abs(3
    - muk)));
end do:
```

```
k=, 1, muk=, 3.2000000, [ 1.   0.4000   -0.4000 ], 0.20000000
k=, 2, muk=, 3.0303030, [ 1.   0.4848   -0.4848 ], 0.030303030
```

$$k=, 3, muk=, 3.0043668, \begin{bmatrix} 1. & 0.4978 & -0.4978 \end{bmatrix}, 0.0043668122$$

$$k=, 4, muk=, 3.0006246, \begin{bmatrix} 1. & 0.4997 & -0.4997 \end{bmatrix}, 0.00062460962$$

$$k=, 5, muk=, 3.0000892, \begin{bmatrix} 1. & 0.5000 & -0.5000 \end{bmatrix}, 0.000089245872$$

$$k=, 6, muk=, 3.0000127, \begin{bmatrix} 1. & 0.5000 & -0.5000 \end{bmatrix}, 0.000012749735$$

$$k=, 7, muk=, 3.0000018, \begin{bmatrix} 1. & 0.5000 & -0.5000 \end{bmatrix}, 0.0000018213974$$

$$k=, 8, muk=, 3.0000003, \begin{bmatrix} 1. & 0.5000 & -0.5000 \end{bmatrix}, 2.6019977\ 10^{-7}$$

$$k=, 9, muk=, 3.0000000, \begin{bmatrix} 1. & 0.5000 & -0.5000 \end{bmatrix}, 3.7171398\ 10^{-8}$$

$$k=, 10, muk=, 3.0000000, \begin{bmatrix} 1. & 0.5000 & -0.5000 \end{bmatrix}, 5.3101998\ 10^{-9} \qquad \textbf{(5.3)}$$

Symmetric Inverse Power method

for k **from** 1 **to** 10 **do**

if $(k = 1)$ **then** $x := x0$; **end if**;

$y := (A - q\ Id)^{-1}.x$;

$lam := x^{\%T}.y$;

$ynorm := LinearAlgebra[VectorNorm](y, 2)$;

$x := \frac{1}{ynorm}\ y$;

$muk := \frac{1}{lam} + q$;

$print\left(\text{`k=`}, k, \text{`muk=`}, evalf_8(muk), evalf_4\left(x^{\%T}\right), evalf_{10}(\text{abs}(3 - muk))\right)$;

end do:

$k=, 1, muk=, 3.2000000, \left[\begin{array}{ccc} 0.8704 & 0.3482 & -0.3482 \end{array}\right], 0.2000000000$

$k=, 2, muk=, 3.0043668, \left[\begin{array}{ccc} 0.8245 & 0.3997 & -0.3997 \end{array}\right], 0.004366812227$

$k=, 3, muk=, 3.0000892, \left[\begin{array}{ccc} 0.8178 & 0.4071 & -0.4071 \end{array}\right], 0.00008924587238$

$k=, 4, muk=, 3.0000018, \left[\begin{array}{ccc} 0.8164 & 0.4080 & -0.4080 \end{array}\right], 0.000001821397413$

$k=, 5, muk=, 3.0000000, \left[\begin{array}{ccc} 0.8167 & 0.4083 & -0.4083 \end{array}\right], 3.717139787\ 10^{-8}$

$k=, 6, muk=, 3.0000000, \left[\begin{array}{ccc} 0.8159 & 0.4080 & -0.4080 \end{array}\right], 7.585999658\ 10^{-10}$

$k=, 7, muk=, 3.0000000, \left[\begin{array}{ccc} 0.8166 & 0.4084 & -0.4084 \end{array}\right], 1.548163196\ 10^{-11}$

$k=, 8, muk=, 3.0000000, \left[\begin{array}{ccc} 0.8167 & 0.4082 & -0.4082 \end{array}\right], 3.159516726\ 10^{-13}$

$k=, 9, muk=, 3.0000000, \left[\begin{array}{ccc} 0.8166 & 0.4083 & -0.4083 \end{array}\right], 6.447993319\ 10^{-15}$

$k=, 10, muk=, 3.0000000, \left[\begin{array}{ccc} 0.8171 & 0.4084 & -0.4084 \end{array}\right], 1.315917004\ 10^{-16}$ **(5.4)**

9.2. QR Factorization

Power methods are not in general suitable for calculating all the eigenvalues of a matrix because of the growth of round-off error. In this section, we will consider the so-called QR factorization. The method can be utilized to find all the eigenvalues simultaneously. However, it is useful for a wide range of applications. Thus here we will derive the QR algorithm for general purpose.

Square matrix:

Let $A \in \mathbb{R}^{n \times n}$, a square matrix. Then, it may be decomposed as

$$A = Q R \tag{1}$$

where Q is an **orthogonal matrix** (its columns are orthogonal unit vectors meaning $Q^T Q = I$) and R is an upper triangular matrix (also called right triangular matrix)

Rectangular matrix:

More generally, for $A \in \mathbb{R}^{m \times n}$, $m \geq n$, we can factor it as

$$A = Q R = \left[Q_1, Q_2 \right] \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1$$

where $Q \in \mathbb{R}^{m \times m}$, an orthogonal matrix, $R \in \mathbb{R}^{m \times n}$, an upper triangular matrix, and

$$Q_1 \in \mathbb{R}^{m \times n},\ Q \in \mathbb{R}^{m \times (m-n)},\ R_1 \in \mathbb{R}^{n \times n}$$

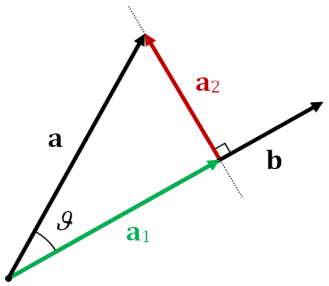
The decomposition $A = Q_1 R_1$ is called the **reduced/thin QR factorization**.

Note: If A is of full rank n and we require that the diagonal elements of R_1 are positive, then R_1 and Q_1 are unique, but in general Q_2 is not.

Definitions

Projection (of ***a*** onto ***b***):

$$proj_{\boldsymbol{b}}\,\boldsymbol{a} = \frac{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}{\langle \boldsymbol{b}, \boldsymbol{b} \rangle}\boldsymbol{b} = \frac{\boldsymbol{a}\cdot\boldsymbol{b}}{\boldsymbol{b}\cdot\boldsymbol{b}}\boldsymbol{b}$$



In the figure, $proj_{\boldsymbol{b}}\,\boldsymbol{a} = \boldsymbol{a}_1$ and $\|a_1\| = \|a\| \cos\theta$, and $\boldsymbol{a} = \boldsymbol{a}_1 + \boldsymbol{a}_2$ is called an orthogonal decomposition of ***a***.

Householder reflection (or, Householder transformation, Elementary reflector)
is a transformation that takes a vector and reflects it about some plane or hyperplane.
Let u be a unit vector which is orthogonal to the hyperplane. Then, **the reflection of a point *x* about this hyperplane** is:

$$x - 2\,proj_u\,x = x - 2 < x, u > u$$

Figure:

Note: $x - 2 < x, u > u = x - 2\,u(u^T x) = (I - 2\,u\,u^T)\,x$, and

$$Q = \mathrm{I} - 2\,u\,u^T \qquad \textbf{(1.1)}$$

is called a **Householder matrix (or, Householder reflector)**

Computation of the QR Factorization

There have been several methods for the computation of the QR factorization, such as by means of

$$\begin{cases} \text{Gram–Schmidt process} \\ \text{Householder reflections} \end{cases}$$

Using the Gram–Schmidt process:

Consider the Gram–Schmidt process applied to the columns of the full column rank matrix $A = \left[a_1, a_2, ..., a_n\right]$.

$$\begin{array}{ll} v_1 = a_1 & u_1 = \dfrac{v_1}{\|v_1\|} \\ v_2 = a_2 - proj_{u_1} a_2 & u_2 = \dfrac{v_2}{\|v_2\|} \\ v_3 = a_3 - proj_{u_1} a_3 - proj_{u_2} a_3 & u_3 = \dfrac{v_3}{\|v_3\|} \\ \vdots & \vdots \\ v_k = a_k - \displaystyle\sum_{j=1}^{k-1} proj_{u_j} a_k & u_k = \dfrac{v_k}{\|v_k\|} \end{array}$$

Since $\{u_1, u_2, ..., u_n\}$ becomes an orthonormal basis, we can rearrange the above equations as

$$\begin{aligned}
a_1 &= \langle u_1, a_1 \rangle u_1 \\
a_2 &= \langle u_1, a_2 \rangle u_1 + \langle u_2, a_2 \rangle u_2 \\
a_3 &= \langle u_1, a_3 \rangle u_1 + \langle u_2, a_3 \rangle u_2 + \langle u_3, a_3 \rangle u_3 \\
&\vdots \\
a_k &= \sum_{j=1}^{k} < u_j, a_k > u_j
\end{aligned}$$

where $\langle u_j, a_j \rangle = \| v_j \|$ and the above can be rewritten in the form

$$A = Q R \tag{2}$$

where

$$Q = \left[u_1, u_2, ..., u_n \right], \quad R = \begin{bmatrix} < u_1, a_1 > & < u_1, a_2 > & < u_1, a_3 > & \dots \\ 0 & < u_2, a_2 > & < u_2, a_3 > & \dots \\ 0 & 0 & < u_3, a_3 > & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Example

```
restart
with(LinearAlgebra) :
A := Matrix([[3, 1, 3], [1, 6, 4], [6, 7, 8], [3, 3, 7]])
```

$$\begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \\ 3 & 3 & 7 \end{bmatrix} \tag{2.1}$$

```
a[1] := Column(A, 1) :   v[1] := a[1] :   u[1] := v[1]/norm(v[1], 2) :
for k from 2 to 3 do
    a[k] := Column(A, k);
    v[k] := a[k] - add(a[k].u[j] u[j], j = 1 ..k - 1);
    u[k] := v[k]/norm(v[k], 2);
end do:

R := Matrix(3) :
for i from 1 to 3 do
   for j from i to 3 do
       R[i, j] := u[i].a[j];
   end do;
end do:
```

$Q := \langle u_1 | u_2 | u_3 \rangle$

$$\begin{bmatrix} \frac{3}{55}\sqrt{55} & -\frac{5}{143}\sqrt{143} & -\frac{23}{12090}\sqrt{4030} \\ \frac{1}{55}\sqrt{55} & \frac{54}{715}\sqrt{143} & \frac{1}{2015}\sqrt{4030} \\ \frac{6}{55}\sqrt{55} & \frac{1}{143}\sqrt{143} & -\frac{38}{6045}\sqrt{4030} \\ \frac{3}{55}\sqrt{55} & -\frac{3}{715}\sqrt{143} & \frac{173}{12090}\sqrt{4030} \end{bmatrix} \tag{2.2}$$

R

$$\begin{bmatrix} \sqrt{55} & \frac{12}{11}\sqrt{55} & \frac{82}{55}\sqrt{55} \\ 0 & \frac{5}{11}\sqrt{143} & \frac{32}{143}\sqrt{143} \\ 0 & 0 & \frac{3}{65}\sqrt{4030} \end{bmatrix} \tag{2.3}$$

$Q.R = A$

$$\begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \\ 3 & 3 & 7 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \\ 3 & 3 & 7 \end{bmatrix} \tag{2.4}$$

$Q^{\%T}.Q$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Using a Maple built-in function:

$$q, r := QRDecomposition(A)$$

$$\left[\begin{bmatrix} \frac{3}{55}\sqrt{55} & -\frac{5}{143}\sqrt{143} & -\frac{23}{12090}\sqrt{4030} \\ \frac{1}{55}\sqrt{55} & \frac{54}{715}\sqrt{143} & \frac{1}{2015}\sqrt{4030} \\ \frac{6}{55}\sqrt{55} & \frac{1}{143}\sqrt{143} & -\frac{38}{6045}\sqrt{4030} \\ \frac{3}{55}\sqrt{55} & -\frac{3}{715}\sqrt{143} & \frac{173}{12090}\sqrt{4030} \end{bmatrix}, \begin{bmatrix} \sqrt{55} & \frac{12}{11}\sqrt{55} & \frac{82}{55}\sqrt{55} \\ 0 & \frac{5}{11}\sqrt{143} & \frac{32}{143}\sqrt{143} \\ 0 & 0 & \frac{3}{65}\sqrt{4030} \end{bmatrix}\right] \quad \textbf{(2.6)}$$

Example

```
restart
with(LinearAlgebra) :
B := Matrix([[2,-1, 0], [-1, 2,-1], [0,-1, 2]])
```

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \tag{3.1}$$

```
Eigenvectors(B)
```

$$\begin{bmatrix} 2+\sqrt{2} \\ 2-\sqrt{2} \\ 2 \end{bmatrix}, \begin{bmatrix} 1 & 1 & -1 \\ -\sqrt{2} & \sqrt{2} & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

```
Determinant(B)
```

$$4 \tag{3.2}$$

```
Q, R := QRDecomposition(B)
```

$$\begin{bmatrix} \frac{2}{5}\sqrt{5} & \frac{3}{70}\sqrt{70} & \frac{1}{14}\sqrt{14} \\ -\frac{1}{5}\sqrt{5} & \frac{3}{35}\sqrt{70} & \frac{1}{7}\sqrt{14} \\ 0 & -\frac{1}{14}\sqrt{70} & \frac{3}{14}\sqrt{14} \end{bmatrix}, \begin{bmatrix} \sqrt{5} & -\frac{4}{5}\sqrt{5} & \frac{1}{5}\sqrt{5} \\ 0 & \frac{1}{5}\sqrt{70} & -\frac{8}{35}\sqrt{70} \\ 0 & 0 & \frac{2}{7}\sqrt{14} \end{bmatrix} \tag{3.3}$$

```
R[1, 1]·R[2, 2]·R[3, 3] = 2/35*sqrt(5)*sqrt(70)*sqrt(14)
simplify(%)
```

$$4 \tag{3.4}$$

In general,

$$|det(A)| = |det(R)| = \left| \prod_{k=1}^{n} r_{ii} \right|$$

Theory on QR Decomposition

Theorem: Let $A \in \mathbb{R}^{n \times n}$. Then there exist an orthogonal matrix Q and an upper triangular matrix R such that $A = Q\,R$.

Theorem: Let $A \in \mathbb{R}^{n \times n}$ be nonsingular. Then there exist **unique** $Q, R \in \mathbb{R}^{n \times n}$ such that Q is orthogonal, R is upper triangular with positive main diagonal entries, and $A = Q\,R$.

Householder Reflectors

Theorems on Reflectors

1. Let $u \in \mathbb{R}^n$ with $\|u\|_2 = 1$, and define $P \in \mathbb{R}^{n \times n}$ by $P = u\,u^T$. Then

   (a) $P\,u = u$

   (b) $P\,v = 0$ if $\langle u, v\rangle = 0$

   (c) $P^2 = P$

   (d) $P^T = P$

2. Let $u \in \mathbb{R}^n$ with $\|u\|_2 = 1$, and define $Q \in \mathbb{R}^{n \times n}$ by $Q = I - 2\,u\,u^T$. Then

   (a) $Q\,u = -u$

   (b) $Q\,v = v$ if $\langle u, v\rangle = 0$

   (c) $Q = Q^T$ (Q is symmetric)

   (d) $Q^T = Q^{-1}$ (Q is orthogonal)

   (e) $Q^{-1} = Q$ (Q is an involution)

3. Let $x, y \in \mathbb{R}^n$ with $x \neq y$ but $\|x\|_2 = \|y\|_2$. Then there is a unique reflector Q such that $Q\,x = y$. (The reflector Q can be constructed, following the same arguments presented in two pages down.)

Corollary 1: Let $u \in \mathbb{R}^n$ be a nonzero vector and $\gamma = 2 / \| u \|_2^2$. Define $Q \in \mathbb{R}^{n \times n}$ by $Q = I - \gamma u u^T$. Then

(a) $Q u = -u$

(b) $Q v = v$ if $\langle u, v \rangle = 0$

Corollary 2: Let $x \in \mathbb{R}^n$ be any nonzero vector. Then there is a reflector Q such that

$$Q x = Q \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} * \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Proof: Let $y = (-\tau, 0, ..., 0)^T$ with $\tau = \pm \| x \|_2$. By choosing the sign appropriately, we can guarantee that $x \neq y$. Clearly $\| x \|_2 = \| y \|_2$. Thus, the corollary follows from Theorem 3.

Construction of $Q: x \to y = (-\tau, 0, ..., 0)^T$, where $\tau = \pm \|x\|_2$

Let

$$\boldsymbol{Q = I - \gamma u u^T} \qquad \textbf{(5.1)}$$

where

$$\boldsymbol{u = x - y = [x_1 + \tau, x_2, ..., x_n]; \quad \gamma = \frac{2}{\|u\|_2^2};}$$

and the sign of τ is such that $x_1 + \tau \neq 0$, e.g., $\boldsymbol{sign(x_1) = sign(\tau)}$ for $x_1 \neq 0$.

Let's check if $\boldsymbol{Q x = y = (-\tau, 0, ..., 0)^T}$

Rewrite x as

$$x = \frac{1}{2}(x - y) + \frac{1}{2}(x + y)$$

Note that by Corollary 1(a),

$$Q(x - y) = Q u = -u = (y - x)$$

Since $\langle x - y, x + y \rangle = \|x\|_2^2 - \|y\|_2^2 = 0$, it follows from Corollary 1(b) that

$$Q(x + y) = (x + y).$$

Thus

$$Q x = Q\left[\frac{1}{2}(x - y) + \frac{1}{2}(x + y)\right] = \frac{1}{2}(y - x) + \frac{1}{2}(x + y) = y$$

Note: Any nonzero multiple of the above u will generate the same reflector.

Example: $Q = I - \gamma\, u\, u^T$: $x \to y$, where $u = x - y$ and $\gamma = 2 / \|u\|_2^2$

$$x := \langle 2, 2, -1 \rangle = \begin{bmatrix} 2 \\ 2 \\ -1 \end{bmatrix}$$

$$y := \langle -norm(x, 2), 0, 0 \rangle = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}$$

$$u := x - y = \begin{bmatrix} 5 \\ 2 \\ -1 \end{bmatrix}$$

$$Q := Matrix(3, shape = identity) - \frac{2}{norm(u, 2)^2}\, u.u^{\%T}$$

$$\begin{bmatrix} -\frac{2}{3} & -\frac{2}{3} & \frac{1}{3} \\ -\frac{2}{3} & \frac{11}{15} & \frac{2}{15} \\ \frac{1}{3} & \frac{2}{15} & \frac{14}{15} \end{bmatrix} \quad \textbf{(6.1)}$$

$Q.x$

$$\begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix} \quad \textbf{(6.2)}$$

$Q^{\%T}.Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ **Note**: Q is symmetric, orthogonal, and involution matrix.

QR Decomposition by Reflectors

Theorem (revisited): Let $A \in \mathbb{R}^{n \times n}$. Then there exist an orthogonal matrix Q and an upper triangular matrix R such that $A = Q\,R$.

Proof

The proof is by induction on n.
When $n = 1$, let $Q = [1]$ and $R = \begin{bmatrix} a_{11} \end{bmatrix}$ to get $A = Q\,R$.

For $n \geq 2$, assume that QR decomposition exists for $(n-1) \times (n-1)$ matrices. Let Q_1 be a reflector that creates zeros in the first column of A:

$$Q_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} = \begin{bmatrix} -\tau_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \text{ where } \tau_1 = \pm \left\| a_1 \right\|_2$$

Thus,

$$Q_1^T A = Q_1 A = \begin{bmatrix} -\tau_1 & \widehat{a}_{12} & \cdots & \widehat{a}_{1n} \\ 0 & & & \\ \vdots & & \widehat{A}_2 & \\ 0 & & & \end{bmatrix}$$

By the induction hypothesis, $\widehat{A}_2 \in \mathbb{R}^{(n-1) \times (n-1)}$ has a QR decomposition:

$$\widehat{A}_2 = \widehat{Q}_2 \widehat{R}_2$$

Define $Q_2 \in \mathbb{R}^{n \times n}$ by

$$Q_2 = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & \widehat{Q}_2 & \\ 0 & & & \end{bmatrix}$$

Then Q_2 is obviously orthogonal and

$$Q_2^T Q_1^T A = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & \widehat{Q}_2 & \\ 0 & & & \end{bmatrix} \begin{bmatrix} -\tau_1 & \widehat{a}_{12} & \dots & \widehat{a}_{1n} \\ 0 & & & \\ \vdots & & \widehat{A}_2 & \\ 0 & & & \end{bmatrix} = \begin{bmatrix} -\tau_1 & \widehat{a}_{12} & \dots & \widehat{a}_{1n} \\ 0 & & & \\ \vdots & & \widehat{R}_2 & \\ 0 & & & \end{bmatrix}$$

The matrix is upper triangular; let us call it R. Let $Q := Q_1 Q_2$. Then $A = Q R$, clearly.

Remarks:

1. For $Q = I - \gamma u u^T : x \to y = (-\tau, 0, ..., 0)^T$, since any multiple of $u = x - y$ will generate the same reflector, you may scale u so that its first entry is 1. That is,

$$u = \frac{x - y}{x_1 + \tau} = \begin{bmatrix} 1 \\ x_2/(x_1 + \tau) \\ : \\ x_n/(x_1 + \tau) \end{bmatrix}$$

In this case, the first entry of u does not need to be saved. Furthermore

$$\| u \|_2^2 = \frac{(x_1 + \tau)^2 + x_2^2 + ... + x_n^2}{(x_1 + \tau)^2} = \frac{\tau^2 + 2\tau x_1 + \|x\|_2^2}{(x_1 + \tau)^2}$$

Since $\tau^2 = \| x \|_2^2$,

$$\| u \|_2^2 = \frac{2\tau^2 + 2\tau x_1}{(x_1 + \tau)^2} = \frac{2\tau(\tau + x_1)}{(x_1 + \tau)^2} = \frac{2\tau}{\tau + x_1}$$

Thus,

$$\gamma = \frac{2}{\| u \|_2^2} = \frac{\tau + x_1}{\tau}$$

2. The the proof suggests an algorithm for constructing Q and R.

$$Q_1 = I - \gamma_1 u_1 u_1^T, \text{ where } u_1 = \begin{bmatrix} 1 \\ a_{21}/(a_{11} + \tau_1) \\ \vdots \\ a_{n1}/(a_{11} + \tau_1) \end{bmatrix}$$

and

$$Q_2 = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & & & \\ \vdots & & I - \gamma_2 u_2 u_2^T & \\ 0 & & & \end{bmatrix}$$

where $u_2 \in \mathbb{R}^{n-1}$ is determined from $\widehat{A}_2$. In general,

$$Q_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & I - \gamma_k u_k u_k^T \end{bmatrix}$$

Then, if $Q = Q_1 Q_2 \ldots Q_{n-1}$, then $Q^T A = R$ is a upper triangular matrix.

3. We do not need form Q_1, Q_2, ... explicitly. For example, for

$$Q_1 = I - \gamma_1 u_1 u_1^T$$

we store only $-\tau_1, \gamma_1, u_1$. (The construction of Q_1 costs O(n) flops.)
Then, for each a_k (or columns of A), $k = 2 .. n$,

$$Q_1^T a_k = Q_1 a_k = \left(I - \gamma_1 u_1 u_1^T\right) a_k = a_k - \gamma_1 \left(u_1^T a_k\right) u_1$$

the last of which requires about 2 n flops.

4. The flop count for $Q_1^T A$:

$$\text{cost}\left(Q_1^T A\right) \approx 2\, n^2$$

Algorithm: The QR Decomposition by Reflectors

$A \in \mathbb{R}^{n \times n}$

For $k = 1 \,..\, (n-1)$

i. Determine $Q_k = I - \gamma_k u_k u_k^T$ such that $Q_k \left[a_{kk} ... a_{nk}\right]^T = \left[-\tau_k \; 0 \; ... 0\right]^T$

ii. Store u_k over $A[k:n,\, k]$

iii. Save $A[k,\, k] := -\tau_k$

iv. $A[k:n,\, k+1:n] \leftarrow Q_k A[k:n,\, k+1:n]$

v. Save $G[k] := \gamma_k$

End

$G[n] := A[n,\, n]$

Notes

1. The output (Q and R) is saved over A.

2. Recall that the flop count for $k = 1$ is about $2n^2$. For $k = 2$, it is about $2(n-1)^2$; for $k = 3$, it is about $2(n-2)^2$; and so on. Thus the total flop count for QR Decomposition by Reflectors is about

$$2n^2 + 2(n-1)^2 + ... \approx \frac{2n^3}{3}$$

which is twice that of an LU-factorization.

3. Although each of Q_k is symmetric and an involution, $Q = Q_1 Q_2 ... Q_{n-1}$ may not be symmetric nor an involution. However, it is still orthogonal.

4. The matrix A is singular if at least one entry of G (particularly, $G[n]$) is zero.

Example: QR by Reflectors

```
restart
with(LinearAlgebra) :
m := 3 :  n := 3 :
```

$$A := Matrix([[3, 1, 3], [1, 6, 4], [6, 7, 8]]) = \begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \end{bmatrix}$$

```
############### k=1 ##########################################
a[1] := Column(A, 1) :
τ[1] := norm(a[1], 2) :
 if a[1][1] < 0 then τ[1] := -τ[1]; end if;
 u[1] := a[1] : scale := a[1][1] + τ[1] :
 u[1][1] := 1 :
 for i from 2 to m do u[1][i] := u[1][i]/scale; end do:
  g[1] := scale/τ[1] :
##
```

$$u_1 = \begin{bmatrix} 1 \\ \frac{1}{3+\sqrt{46}} \\ \frac{6}{3+\sqrt{46}} \end{bmatrix}$$

```
Q1 := Matrix(m, shape = identity) - g[1]·u[1].u[1]^%T;
```

$$\begin{bmatrix} 1-\frac{1}{46}\left(3+\sqrt{46}\right)\sqrt{46} & -\frac{1}{46}\sqrt{46} & -\frac{3}{23}\sqrt{46} \\ -\frac{1}{46}\sqrt{46} & 1-\frac{1}{46}\frac{\sqrt{46}}{3+\sqrt{46}} & -\frac{3}{23}\frac{\sqrt{46}}{3+\sqrt{46}} \\ -\frac{3}{23}\sqrt{46} & -\frac{3}{23}\frac{\sqrt{46}}{3+\sqrt{46}} & 1-\frac{18}{23}\frac{\sqrt{46}}{3+\sqrt{46}} \end{bmatrix} \quad \textbf{(10.1)}$$

$Q1A := simplify\left(Q1^{\%T}.A\right) :\ \ evalf\left(Q1A\right)$

$$\begin{bmatrix} -6.782329983 & -7.519539763 & -8.993959329 \\ 0. & 5.129088899 & 2.773915892 \\ 0. & 1.774533402 & 0.6434953557 \end{bmatrix} \quad \textbf{(10.2)}$$

############### k=2 ##

$A2 := Q1A[2..m, 2..n] :$

$a_2 := Column(A2, 1) :$

$\tau_2 := norm\left(a_2, 2\right) :$

if $evalf\left(a_2[1]\right) < 0$ **then** $\tau_2 := -\tau_2$; **end if**;

$u_2 := a_2 :\ scale := a_2[1] + \tau_2 :$

$u_2[1] := 1 :$

for i **from** 2 **to** $m-1$ **do** $u_2[i] := u_2[i]/scale$; **end do**:

$g_2 := scale/\tau_2 :$

##

$$simplify\left(u_2\right) = \begin{bmatrix} 1 \\ \frac{2\left(8\sqrt{46}+345\right)}{225\sqrt{46}+782+3\sqrt{46}\sqrt{1355}+46\sqrt{1355}} \end{bmatrix}$$

$Q2 := Matrix(m) :$

$Q2[1, 1] := 1 :$

$Q2sub := simplify\left(Matrix(m-1, shape = identity) - g_2 \cdot u_2.u_2^{\%T}\right) :$

$evalf(Q2sub)$

$$\begin{bmatrix} -0.9450384856 & -0.3269591131 \\ -0.3269591131 & 0.9450384849 \end{bmatrix} \tag{10.3}$$

for i **from** 2 **to** m **do**
for j **from** 2 **to** m **do**
$Q2[i, j] := Q2sub[i-1, j-1];$
end do:
end do:
$evalf(Q2)$

$$\begin{bmatrix} 1. & 0. & 0. \\ 0. & -0.9450384856 & -0.3269591131 \\ 0. & -0.3269591131 & 0.9450384849 \end{bmatrix} \tag{10.4}$$

############## Q & R ################################
$Q := simplify(Q1.Q2) : \ evalf(Q)$

$$\begin{bmatrix} -0.4423258684 & 0.4285832703 & -0.7878224459 \\ -0.1474419561 & -0.9012265027 & -0.4074943688 \\ -0.8846517369 & -0.06408721798 & 0.4618269510 \end{bmatrix} \tag{10.5}$$

$R := simplify\left(Q2^{\%T}.Q1A\right) : \ \ evalf(R)$

$$\begin{bmatrix} -6.782329983 & -7.519539763 & -8.993959329 \\ 0. & -5.427386271 & -2.831853944 \\ 0. & 0. & -0.2988292036 \end{bmatrix} \tag{10.6}$$

$simplify(Q.R)$

$$\begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \end{bmatrix} \tag{10.7}$$

On the other hand

$q, r := QRDecomposition(A)$

$$\left[\begin{bmatrix} \frac{3}{46}\sqrt{46} & -\frac{107}{62330}\sqrt{62330} & \frac{29}{1355}\sqrt{1355} \\ \frac{1}{46}\sqrt{46} & \frac{45}{12466}\sqrt{62330} & \frac{3}{271}\sqrt{1355} \\ \frac{3}{23}\sqrt{46} & \frac{8}{31165}\sqrt{62330} & -\frac{17}{1355}\sqrt{1355} \end{bmatrix}, \begin{bmatrix} \sqrt{46} & \frac{51}{46}\sqrt{46} & \frac{61}{46}\sqrt{46} \\ 0 & \frac{1}{46}\sqrt{62330} & \frac{707}{62330}\sqrt{62330} \\ 0 & 0 & \frac{11}{1355}\sqrt{1355} \end{bmatrix}\right] \quad \textbf{(10.8)}$$

$evalf(q)$

$$\begin{bmatrix} 0.4423258684 & -0.4285832703 & 0.7878224461 \\ 0.1474419561 & 0.9012265027 & 0.4074943687 \\ 0.8846517369 & 0.06408721798 & -0.4618269512 \end{bmatrix} \quad \textbf{(10.9)}$$

$evalf(r)$

$$\begin{bmatrix} 6.782329983 & 7.519539763 & 8.993959329 \\ 0. & 5.427386271 & 2.831853944 \\ 0. & 0. & 0.2988292037 \end{bmatrix} \quad \textbf{(10.10)}$$

Empty Page

9.3. Singular Value Decomposition (SVD)

Let $A \in \mathbb{R}^{m \times n}$ for arbitrary integers $m, n \geq 1$. In this section, we consider the factorization of A into what is called the Singular Value Decomposition. The decomposition takes the form

$$A = U \Sigma V^T \tag{1}$$

where

U	an $m \times m$ orthogonal matrix
Σ	an $m \times n$ diagonal matrix of singular values
V	an $n \times n$ orthogonal matrix

Note: In most applications, $m \gg n$. So, we assume that $m > n$ for simplicity, although the algorithms to be presented would work for general cases.

Definitions

The **singular values** of $A \in \mathbb{R}^{m \times n}$ are the nonnegative square roots of eigenvalues of $A^T A \in \mathbb{R}^{n \times n}$.
The **rank** of A, denoted by $Rank(A)$, is the number of linearly independent rows in A.
The **nullity** of A, denoted by $Nul(A)$, is $n - Rank(A)$.

Basic properties, for $A \in \mathbb{R}^{m \times n}$

1. $Rank(A) = Rank\left(A^T\right)$
2. The matrices $A^T A$ and AA^T are symmetric.
3. $Rank(A) = Rank\left(A^T A\right)$ and therefore $Nul(A) = Nul\left(A^T A\right)$
4. Eigenvalues of $A^T A$ and AA^T are real and nonnegative.
5. The nonzero eigenvalues of $A^T A$ and AA^T are the same.

Diagonalizable Matrices

Theorem 1: A square matrix $A \in \mathbb{R}^{n \times n}$ is similar to a diagonal matrix D if and only if A has n linearly independent eigenvectors. In this case,

$$A = V \, \mathrm{D} \, V^{-1}$$

where the columns of V consist of the eigenvectors, and the i-th diagonal entry of D is the eigenvalue of A that corresponds to the i-th column of V.

Theorem 2: Let v_1 and v_2 be eigenvectors of a **symmetric** $n \times n$ matrix corresponding to two distinct eigenvalues $\lambda_1 \neq \lambda_2$. Then $v_1 \bullet v_2 = 0$.

Proof:

$$\left(\lambda_1 - \lambda_2\right) v_1^T v_2 = \left(\lambda_1 v_1\right)^T v_2 - v_1^T \left(\lambda_2 v_2\right) = \left(A v_1\right)^T v_2 - v_1^T \left(A v_2\right) = 0.$$

Since $\lambda_1 - \lambda_2 \neq 0$, we conclude $v_1^T v_2 = 0$.

Theorem 3: An $n \times n$ matrix A is symmetric if and only if there is a diagonal matrix D and an orthogonal matrix Q such that

$$A = Q \, \mathrm{D} \, Q^{-1} = Q \, \mathrm{D} \, Q^T$$

where Q consists of *orthonormal* eigenvectors of A.

Corollary: The matrices $A^T A$ and $A A^T$ are diagonalizable. That is, there are orthogonal matrices $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ such that

$$A A^T = U \, S_m \, U^T; \quad A^T A = V \, S_n \, V^T$$

where S_m and S_n are diagonal matrices having identical nonzero diagonal entries.

Singular Value Decomposition (SVD)

$$A = U \Sigma V^T \tag{2}$$

where

$U \in \mathbb{R}^{m \times m}$	left-singular vectors orthonormal eigenvectors of $A\,A^T$
$\Sigma \in \mathbb{R}^{m \times n}$	diagonal matrix of singular values
$V \in \mathbb{R}^{n \times n}$	right-singular vectors orthonormal eigenvectors of $A^T A$

Construction of SVD

1. Construct $\Sigma \leftarrow$ nonnegative square roots of eigenvalues of $A^T A$
2. Construct $V \leftarrow$ orthonormal eigenvectors of $A^T A$
3. Construct $U \leftarrow$ orthonormal eigenvectors of AA^T

Let's see the details.

1. Construction of Σ

Let λ_i, $i = 1 \cdots n$, be the nonnegative eigenvalues of $A^T A$. Without loss of generality, we may assume

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k > \lambda_{k+1} = \cdots = \lambda_n = 0$$

($Rank(A^T A) = k \leq n$.)

Define the **singular values** as

$$s_i = \sqrt{\lambda_i}, \quad i = 1 \cdots n$$

Then,

$$\Sigma = \begin{bmatrix} s_1 & 0 & \cdots & 0 \\ 0 & s_2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & s_n \\ 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}$$

where $s_{k+1} = \cdots = s_n = 0$

Example

```
restart
with(LinearAlgebra) :
```

$$A := \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} :$$

```
ATA := A^%T . A
```

$$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 2 \end{bmatrix} \quad \textbf{(4.1)}$$

```
p := x -> CharacteristicPolynomial(ATA, x) :
p(x)
```

$$x^3 - 8x^2 + 17x - 10 \quad \textbf{(4.2)}$$

```
Lam := sort( [solve(p(x) = 0, x)], `>` )
```

$$[5, 2, 1] \quad \textbf{(4.3)}$$

```
Σ := Matrix(5, 3) : s := Vector(3) :
for i from 1 to 3 do
    s[i] := sqrt(Lam[i]);
    Σ[i, i] := s[i];
end do:
'Σ' = Σ
```

$$\Sigma = \begin{bmatrix} \sqrt{5} & 0 & 0 \\ 0 & \sqrt{2} & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \textbf{(4.4)}$$

2. Construction of V

Since $A^T A$ is symmetric, it is diagonalizable as

$$A^T A = V \, \mathrm{D} \, V^T$$

where D is a diagonal matrix of eigenvalues of $A^T A$ and V is an orthogonal matrix of eigenvectors. Note that D is unique, but V may not be.

Example

$ATA := A^{\%T}.A$

$$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{5.1}$$

$e, G := Eigenvectors(ATA)$

$$\begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & -1 \\ 2 & -1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{5.2}$$

Note that G is not orthogonal; it can be transformed to be orthogonal by applying the Gram-Schmidt process.

$a_1 := Column(G, 1) : \quad v_1 := a_1 : \quad u_1 := v_1 / norm(v_1, 2) :$

for k **from** 2 **to** 3 **do**

$a_k := Column(G, k);$

$v_k := a_k - add(a_k.u_j \, u_j, j = 1 \,..k - 1);$

$u_k := v_k / norm(v_k, 2);$

end do:

$V := \langle u_1 | u_2 | u_3 \rangle :$

$'V' = V$

$$V = \begin{bmatrix} \frac{1}{6}\sqrt{6} & \frac{1}{3}\sqrt{3} & -\frac{1}{2}\sqrt{2} \\ \frac{1}{3}\sqrt{6} & -\frac{1}{3}\sqrt{3} & 0 \\ \frac{1}{6}\sqrt{6} & \frac{1}{3}\sqrt{3} & \frac{1}{2}\sqrt{2} \end{bmatrix} \quad \textbf{(5.3)}$$

Check:

##========

```
D3 := Matrix(3) :
for i from 1 to 3 do D3[i, i] := e[i]; end do:
'D'= D3
```

$$\mathrm{D} = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(5.4)}$$

```
'VD V^T' = V.D3.V^%T
```

$$VD\,V^T = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 2 \end{bmatrix} \quad \textbf{(5.5)}$$

which is the same as $A^T A$.

3. Construction of U

Example

$AAT := A.A^{\%T}$

$$\begin{bmatrix} 2 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 2 \end{bmatrix} \quad \textbf{(6.1)}$$

$e, G := Eigenvectors(AAT)$

$$\begin{bmatrix} 5 \\ 2 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \frac{2}{3} & -2 & 0 & -1 & 0 \\ \frac{2}{3} & 1 & 0 & -2 & -1 \\ 1 & 0 & -1 & 1 & 0 \\ \frac{2}{3} & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \quad \textbf{(6.2)}$$

Again, here, G is not orthogonal; let's use the Gram-Schmidt process:

$a_1 := Column(G, 1) : \quad v_1 := a_1 : \quad u_1 := v_1 / norm(v_1, 2) :$

for k **from** 2 **to** 5 **do**

$a_k := -Column(G, k);$ ## **(-) is extra**

$v_k := a_k - add(a_k.u_j\, u_j, j = 1..k-1);$

$u_k := v_k / norm(v_k, 2);$

end do:

$U := \langle u_1 | u_2 | u_3 | u_4 | u_5 \rangle :$

$'U' = U$

$$U = \begin{bmatrix} \frac{1}{15}\sqrt{30} & \frac{1}{3}\sqrt{6} & 0 & \frac{1}{7}\sqrt{7} & -\frac{1}{35}\sqrt{70} \\ \frac{1}{15}\sqrt{30} & -\frac{1}{6}\sqrt{6} & 0 & \frac{2}{7}\sqrt{7} & \frac{3}{70}\sqrt{70} \\ \frac{1}{10}\sqrt{30} & 0 & \frac{1}{2}\sqrt{2} & -\frac{1}{7}\sqrt{7} & \frac{1}{35}\sqrt{70} \\ \frac{1}{15}\sqrt{30} & -\frac{1}{6}\sqrt{6} & 0 & 0 & -\frac{1}{10}\sqrt{70} \\ \frac{1}{10}\sqrt{30} & 0 & -\frac{1}{2}\sqrt{2} & -\frac{1}{7}\sqrt{7} & \frac{1}{35}\sqrt{70} \end{bmatrix} \quad \textbf{(6.3)}$$

$simplify(U.\Sigma.V^{\%T})$

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \quad \textbf{(6.4)}$$

Using the built-in command:

$U1, S1, V1 := SingularValues(A, output = ['U','S','Vt'])$:

$U1$

$$\begin{bmatrix} -0.365148371670111 & -0.816496580927726 & 5.45631686657392\ 10^{-16} & 0.0900653319565860 & -0.438050494782907 \\ -0.365148371670111 & 0.408248290463863 & -6.50157889960984\ 10^{-17} & -0.602553314882261 & -0.580456288383888 \\ -0.547722557505166 & 3.02058725254462\ 10^{-16} & 0.707106781186548 & -0.0900653319565859 & 0.438050494782907 \\ -0.365148371670111 & 0.408248290463863 & -1.76038091458614\ 10^{-16} & 0.782683978795432 & -0.295644701181927 \\ -0.547722557505166 & -2.97288235695525\ 10^{-16} & -0.707106781186548 & -0.0900653319565860 & 0.438050494782907 \end{bmatrix} \quad \textbf{(6.5)}$$

$S1$

$$\begin{bmatrix} 2.23606797749979 \\ 1.41421356237310 \\ 1.00000000000000 \\ 0. \\ 0. \end{bmatrix} \tag{6.6}$$

V1

$$\begin{bmatrix} -0.408248290463863 & -0.816496580927726 & -0.408248290463863 \\ -0.577350269189626 & 0.577350269189626 & -0.577350269189626 \\ -0.707106781186547 & -3.11176331828133\ 10^{-16} & 0.707106781186548 \end{bmatrix} \tag{6.7}$$

S2 := *Matrix*(5, 3) :
for *i* **from** 1 **to** 3 **do** *S2*[*i*, *i*] := *S1*[*i*]; **end do**:
U1.*S2*.*V1*

$$\begin{bmatrix} 1. & -3.33066907387547\ 10^{-16} & 1. \\ -3.98116104566728\ 10^{-16} & 1.00000000000000 & -4.59731052833349\ 10^{-17} \\ -4.44089209850063\ 10^{-16} & 1. & 1.00000000000000 \\ -3.19611481632539\ 10^{-16} & 1.00000000000000 & -1.24477728217524\ 10^{-16} \\ 1.00000000000000 & 1. & -3.33066907387547\ 10^{-16} \end{bmatrix} \tag{6.8}$$

3'. An Alternative for finding *U*

First, consider the nonzero singular values $s_1 \geq s_2 \geq \cdots \geq s_k > 0$ and the corresponding columns in V given by $v_1, v_2, \cdots, v_k$. Define

$$u_i = \frac{1}{s_i} A v_i, \quad i = 1, 2, ..., k$$

Then we can prove that u_i, $i = 1, 2, ..., k$, are eigenvectors of AA^T and

$$u_i^T u_j = \delta_{ij} \tag{3}$$

See a homework problem for **(3)**. Here we will prove that u_i, $i = 1, 2, ..., k$, are eigenvectors of AA^T.

$$AA^T u_i = \frac{1}{s_i} AA^T A v_i = \frac{1}{s_i} A\left(A^T A v_i\right) = \frac{1}{s_i} A\, s_i^2 v_i = s_i^2 \frac{1}{s_i} A v_i = s_i^2 u_i$$

To determine the remaining columns of U, we first need to choose $(m - k)$ vectors $\{x_{k+1}, ..., x_m\}$ such that

$$\{u_1, ..., u_k, x_{k+1}, ..., x_m\}$$

is linear independent. Then we apply the Gram-Schmidt process to obtain $\{u_{k+1}, ..., u_m\}$ so that

$$\{u_1, ..., u_k, u_{k+1}, ..., u_m\}$$

is orthonormal.

Example

$$A=\begin{bmatrix}1&0&1\\0&1&0\\0&1&1\\0&1&0\\1&1&0\end{bmatrix},\ s=\begin{bmatrix}\sqrt{5}\\\sqrt{2}\\1\end{bmatrix},\ \text{and}\ V=\begin{bmatrix}\frac{1}{6}\sqrt{6}&\frac{1}{3}\sqrt{3}&-\frac{1}{2}\sqrt{2}\\\frac{1}{3}\sqrt{6}&-\frac{1}{3}\sqrt{3}&0\\\frac{1}{6}\sqrt{6}&\frac{1}{3}\sqrt{3}&\frac{1}{2}\sqrt{2}\end{bmatrix}$$

for i **from** 1 **to** 3 **do**

$$u_i := \frac{1}{s[i]}\ A.Column(V,\ i);$$

end do:

$\langle u_1 | u_2 | u_3 \rangle$

$$\begin{bmatrix}\frac{1}{15}\sqrt{5}\sqrt{6}&\frac{1}{3}\sqrt{2}\sqrt{3}&0\\\frac{1}{15}\sqrt{5}\sqrt{6}&-\frac{1}{6}\sqrt{2}\sqrt{3}&0\\\frac{1}{10}\sqrt{5}\sqrt{6}&0&\frac{1}{2}\sqrt{2}\\\frac{1}{15}\sqrt{5}\sqrt{6}&-\frac{1}{6}\sqrt{2}\sqrt{3}&0\\\frac{1}{10}\sqrt{5}\sqrt{6}&0&-\frac{1}{2}\sqrt{2}\end{bmatrix} \qquad \textbf{(7.1)}$$

The two remaining vector may simply be selected:

$x4 := \langle 1, 1, 1, 0, 0 \rangle$:

$x5 := \langle 0, 0, 0, 1, 1 \rangle$:

$G := \langle u_1 | u_2 | u_3 | x4 | x5 \rangle$

$$\begin{bmatrix} \frac{1}{15}\sqrt{5}\sqrt{6} & \frac{1}{3}\sqrt{2}\sqrt{3} & 0 & 1 & 0 \\ \frac{1}{15}\sqrt{5}\sqrt{6} & -\frac{1}{6}\sqrt{2}\sqrt{3} & 0 & 1 & 0 \\ \frac{1}{10}\sqrt{5}\sqrt{6} & 0 & \frac{1}{2}\sqrt{2} & 1 & 0 \\ \frac{1}{15}\sqrt{5}\sqrt{6} & -\frac{1}{6}\sqrt{2}\sqrt{3} & 0 & 0 & 1 \\ \frac{1}{10}\sqrt{5}\sqrt{6} & 0 & -\frac{1}{2}\sqrt{2} & 0 & 1 \end{bmatrix} \quad (7.2)$$

$a_1 := Column(G, 1) : \quad v_1 := a_1 : \quad u_1 := v_1 / norm(v_1, 2) :$

for k **from** 2 **to** 5 **do**

$a_k := Column(G, k);$

$v_k := a_k - add(a_k.u_j\, u_j, j = 1..k-1);$

$u_k := v_k / norm(v_k, 2);$

end do:

$U := \langle u_1 | u_2 | u_3 | u_4 | u_5 \rangle$

$$\begin{bmatrix} \frac{1}{15}\sqrt{5}\sqrt{6} & \frac{1}{3}\sqrt{2}\sqrt{3} & 0 & \frac{1}{35}\sqrt{70} & \frac{1}{7}\sqrt{7} \\ \frac{1}{15}\sqrt{5}\sqrt{6} & -\frac{1}{6}\sqrt{2}\sqrt{3} & 0 & \frac{1}{10}\sqrt{70} & 0 \\ \frac{1}{10}\sqrt{5}\sqrt{6} & 0 & \frac{1}{2}\sqrt{2} & -\frac{1}{35}\sqrt{70} & -\frac{1}{7}\sqrt{7} \\ \frac{1}{15}\sqrt{5}\sqrt{6} & -\frac{1}{6}\sqrt{2}\sqrt{3} & 0 & -\frac{3}{70}\sqrt{70} & \frac{2}{7}\sqrt{7} \\ \frac{1}{10}\sqrt{5}\sqrt{6} & 0 & -\frac{1}{2}\sqrt{2} & -\frac{1}{35}\sqrt{70} & -\frac{1}{7}\sqrt{7} \end{bmatrix} \quad (7.3)$$

Check:

$U.\Sigma.V^{\%T}$

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \tag{7.4}$$

Note: V and U may not be unique. U must be constructed, with care, corresponding to V.

Verification of the decomposition: $A = U \Sigma V^T$

It can be verified by showing

$$A V = U \Sigma \tag{4}$$

Since

$$A v_i = \begin{cases} s_i u_i & i = 1 \ldots k \\ 0 & i = k+1 \ldots n \end{cases}$$

we can obtain

$$A V = A \left[v_1, \ldots, v_k, v_{k+1}, \ldots, v_n \right] = \left[A v_1, \ldots, A v_k, A v_{k+1}, \ldots, A v_n \right]$$
$$= \left[s_1 u_1, \ldots, s_k u_k, 0, \ldots, 0 \right]$$
$$= \left[u_1, \ldots, u_k, 0, \ldots, 0 \right] \begin{bmatrix} s_1 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \vdots & \vdots & & \vdots \\ \vdots & \ddots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \ldots & 0 & s_k & 0 & \ldots & 0 \\ 0 & \ldots & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & & & \vdots & \vdots & & \vdots \\ 0 & \ldots & \ldots & 0 & 0 & \ldots & 0 \end{bmatrix} = U \Sigma$$

Note: No matter how the orthonormal vectors $\left[u_{k+1}, \ldots, u_m \right]$ are chosen, $U \Sigma$ remains the same and therefore it holds $A = U \Sigma V^T$. This implies that all columns of U do not have to be eigenvectors of AA^T.

A Remark on the Construction of U

Let the nonzero singular values be $s_1 \geq s_2 \geq \cdots \geq s_k > 0$ and the corresponding columns in V given by $v_1, v_2, \cdots, v_k$. Then, we may define

$$u_i = \frac{1}{s_i} A v_i, \quad i = 1, 2, ..., k$$

To determine the remaining columns of U, we first need to choose $(m - k)$ vectors $\{x_{k+1}, ..., x_m\}$ such that

$$\{u_1, ..., u_k, x_{k+1}, ..., x_m\}$$

is linear independent.

These remaining vectors can be chosen to be eigenvectors of AA^T, which correspond to the eigenvalue zero. That is, $\{x_{k+1}, ..., x_m\}$ can be found as solutions of

$$AA^T x = 0 \tag{5}$$

A way of solving the problem is to transform AA^T into the **reduced echelon form**.

Example

```
restart
with(LinearAlgebra) :
```

$$A := \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} : \quad s := \begin{bmatrix} \sqrt{5} \\ \sqrt{2} \\ 1 \end{bmatrix} : \quad V := \begin{bmatrix} \frac{1}{6}\sqrt{6} & \frac{1}{3}\sqrt{3} & -\frac{1}{2}\sqrt{2} \\ \frac{1}{3}\sqrt{6} & -\frac{1}{3}\sqrt{3} & 0 \\ \frac{1}{6}\sqrt{6} & \frac{1}{3}\sqrt{3} & \frac{1}{2}\sqrt{2} \end{bmatrix} :$$

```
Σ := Matrix(5, 3) :
```

for i **from** 1 **to** 3 **do** $\Sigma[i, i] := s[i]$; **end do**:

for i **from** 1 **to** 3 **do**

$$u_i := \frac{1}{s[i]}\, A.Column(V, i);$$

end do:

$$B := A.A^{\%T}$$

$$\begin{bmatrix} 2 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 2 \end{bmatrix} \quad \textbf{(8.1)}$$

$$R := ReducedRowEchelonForm(B)$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \textbf{(8.2)}$$

Note: $B\,x = 0$ is equivalent to $R\,x = 0$.

$$x := Vector(5, symbol = t) : \quad x^{\%T}$$

$$\begin{bmatrix} t_1 & t_2 & t_3 & t_4 & t_5 \end{bmatrix} \quad \textbf{(8.3)}$$

$$R.x = 0$$

$$\begin{bmatrix} t_1 + t_5 \\ t_2 + t_4 + 2\,t_5 \\ t_3 - t_5 \\ 0 \\ 0 \end{bmatrix} = 0 \qquad \textbf{(8.4)}$$

Equation **(8.4)** can be rewritten as

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{bmatrix} = \begin{bmatrix} -t_5 \\ -t_4 - 2\,t_5 \\ t_5 \\ t_4 \\ t_5 \end{bmatrix} = t_4 \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \\ 0 \end{bmatrix} + t_5 \begin{bmatrix} -1 \\ -2 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Define

$$x_4 := \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \\ 0 \end{bmatrix} : \quad x_5 := \begin{bmatrix} -1 \\ -2 \\ 1 \\ 0 \\ 1 \end{bmatrix} :$$

These two vectors are eigenvectors of AA^T, corresponding to the zero eigenvalue.

$$G := \langle u_1 | u_2 | u_3 | x_4 | x_5 \rangle$$

$$\begin{bmatrix} \frac{1}{15}\sqrt{5}\sqrt{6} & \frac{1}{3}\sqrt{2}\sqrt{3} & 0 & 0 & -1 \\ \frac{1}{15}\sqrt{5}\sqrt{6} & -\frac{1}{6}\sqrt{2}\sqrt{3} & 0 & -1 & -2 \\ \frac{1}{10}\sqrt{5}\sqrt{6} & 0 & \frac{1}{2}\sqrt{2} & 0 & 1 \\ \frac{1}{15}\sqrt{5}\sqrt{6} & -\frac{1}{6}\sqrt{2}\sqrt{3} & 0 & 1 & 0 \\ \frac{1}{10}\sqrt{5}\sqrt{6} & 0 & -\frac{1}{2}\sqrt{2} & 0 & 1 \end{bmatrix} \quad \textbf{(8.5)}$$

Now, apply Gram-Schmidt process:

$a_1 := Column(G, 1) : \quad v_1 := a_1 : \quad u_1 := v_1 / norm(v_1, 2) :$

for k **from** 2 **to** 5 **do**

$a_k := Column(G, k);$

$v_k := a_k - add(a_k.u_j\, u_j, j = 1 ..k - 1);$

$u_k := v_k / norm(v_k, 2);$

end do:

$U := \langle u_1 | u_2 | u_3 | u_4 | u_5 \rangle$

$$\begin{bmatrix} \frac{1}{15}\sqrt{5}\sqrt{6} & \frac{1}{3}\sqrt{2}\sqrt{3} & 0 & 0 & -\frac{1}{5}\sqrt{5} \\ \frac{1}{15}\sqrt{5}\sqrt{6} & -\frac{1}{6}\sqrt{2}\sqrt{3} & 0 & -\frac{1}{2}\sqrt{2} & -\frac{1}{5}\sqrt{5} \\ \frac{1}{10}\sqrt{5}\sqrt{6} & 0 & \frac{1}{2}\sqrt{2} & 0 & \frac{1}{5}\sqrt{5} \\ \frac{1}{15}\sqrt{5}\sqrt{6} & -\frac{1}{6}\sqrt{2}\sqrt{3} & 0 & \frac{1}{2}\sqrt{2} & -\frac{1}{5}\sqrt{5} \\ \frac{1}{10}\sqrt{5}\sqrt{6} & 0 & -\frac{1}{2}\sqrt{2} & 0 & \frac{1}{5}\sqrt{5} \end{bmatrix} \tag{8.6}$$

Now, U is a collection of all linearly independent eigenvectors of AA^T.

$U.\Sigma.V^{\%T}$

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \tag{8.7}$$

Application of SVD for LS Approximation

Let $A \in \mathbb{R}^{m \times n}$, $m > n$, and $b \in \mathbb{R}^m$. Then, the least squares objective is to find a vector $x \in \mathbb{R}^n$ that minimizes

$$\| Ax - b \|_2$$

Suppose that the SVD of A is given, that is,

$$A = U \Sigma V^T \tag{6}$$

where Σ is assumed to have k nonzero singular values.

Since U and V are ℓ_2-norm preserving, we have

$$\| Ax - b \|_2 = \left\| U \Sigma V^T x - b \right\|_2 = \left\| \Sigma V^T x - U^T b \right\|_2$$

Define $z = V^T x$ and $\boldsymbol{c = U^T b}$. Then

$$\| Ax - b \|_2 = \left(\sum_{i=1}^{k} \left(s_i z_i - c_i \right)^2 + \sum_{i=k+1}^{m} c_i^2 \right)^{1/2}$$

Thus the norm is minimized when z is chosen with

$$z_i = \begin{cases} c_i / s_i & \text{when } i \le k \\ \text{arbitrary} & \text{when } k+1 \le i \le n \end{cases}$$

After determining z, one can find the solution as

$$\boldsymbol{x = Vz} \tag{7}$$

Then the least-squares error reads

$$\min_x \| Ax - b \|_2^2 = \sum_{i=k+1}^{m} c_i^2$$

Example

```
with(LinearAlgebra) : with(CurveFitting) :
n := 100 :
roll := rand(-n..n) :
m := 10 :
xy := Matrix(m, 2) :
```

```
for i to m do
    xy[i, 1] := i;
    xy[i, 2] := i + 1/n · roll( );
end do:


n := 2 :
A := Matrix(m, n) : b := Vector(m) :
for i from 1 to m do
    A[i, 1] := 1;  A[i, 2] := xy[i, 1];  b[i] := xy[i, 2];
end do:

U, S, Vt := SingularValues(A, output = ['U','S','Vt']) :
S^%T
```

$$\begin{bmatrix} 19.8217110839779 & 1.44905821253278 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \end{bmatrix} \quad \textbf{(9.1)}$$

```
c := U^%T.b
```

$$\begin{bmatrix} 19.8408478534228 \\ -0.562615727942738 \\ -0.332748118823707 \\ 0.445541486561571 \\ -0.516168908053152 \\ 0.172120697332124 \\ 0.650410302717399 \\ -1.06130009189732 \\ -0.123010486512046 \\ 0.515279118873232 \end{bmatrix} \quad \textbf{(9.2)}$$

```
z := Vector(n) :
z[1] := c[1]/S[1] :   z[2] := c[2]/S[2] :
Sol := Vt^%T.z
```

$$\begin{bmatrix} -0.242000000000000 \\ 1.04600000000000 \end{bmatrix} \quad \textbf{(9.3)}$$

The least-squares error becomes

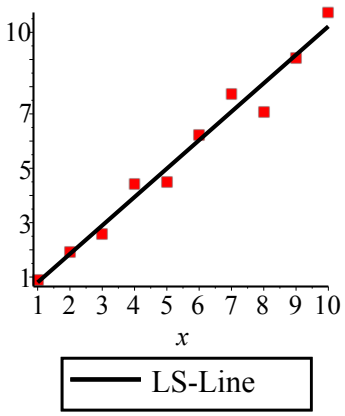
$E2 := add\left(c[i]^2,\ i = 3\ ..10\right)$

$$2.43532000000000 \quad \textbf{(9.4)}$$

See Page 6 of Section 8.1

▶ **plotting**

$plots[display](pdata,\ pcurve)$



Empty Page

Homework:
9. Approximating Eigenvalues

#1. Let $A := \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & 0 & -1 \\ 0 & 0 & 4 & -2 \\ 0 & -1 & -2 & 4 \end{bmatrix}$:

Use indicated methods to approximate eigenvalues and their associated eigenvectors of A within to 10^{-10} accuracy.

a. **Power method,** the largest eigenvalue.
b. **Inverse Power method,** the smallest eigenvalue.
c. **Inverse Power method,** an eigenvalue near 3.
d. Repeat the above with their symmetric versions.

2. Find a reflector Q that maps the vector $x = [4, -3, 1, 3, 1]^T$ to a vector of the form $[-\tau, 0, 0, 0, 0]^T$. Write Q in two ways: (a) in the form of $I - \gamma u u^T$ and (b) as a completely assembled matrix.

#3. Let

$$A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}:$$

Find QR decompositions for A by using (a) the Gram-Schmidt process and (b) reflectors. (You have to show your solutions **step-by-step** in detail.)

#4. This problem revisits the first homework problem of Chapter 8. Consider the same data

x_i	0.2	0.4	0.6	0.8	1.	1.2	1.4	1.6	1.8	2.
y_i	1.88	2.13	1.76	2.78	3.23	3.82	6.13	7.22	6.66	9.07

a. Plot the data (scattered point plot) and decide which curve fits the data best.
b. Construct an algebraic system of the form $G\theta = y$, where $G \in \mathbb{R}^{m \times n}$, $n < m = 10$.
c. Use the **LS code** (you have implemented for HW for Chapter 8) to find the curve.
d. Implement a code for **QR Decomposition** to find θ and then find the curve.
e. Plot the curves superposed over the point plot, and compre them. Are they the same?

#5. For the overdetermined system in the previous problem, use the SVD to find the least-squares solution. Your solution must be the same as the one found in the previous problem.
For the SVD, you may use *SingularValues* (in Maple) or *svd* (in Matlab).

#6. Prove equation (3) on page 9.3.11 (page 11 of Section 9.3).

10. Numerical Solutions of Nonlinear Systems of Equations

In This Chapter:

Topics	Applications/Properties
Fixed Points for Functions of Several Variables	
Fixed-point iteration	NA I; we will do it again
Newton's Method for Systems	(a little more generally)
Quasi-Newton Methods	
Sherman-Morrison formula	
Broyden's method	
Steepest Descent Methods	
An application: Mesh Optimization	Convex mesh optimization

The Problem

Consider a system of nonlinear equations of the form

$$\begin{aligned} f_1(x_1, x_2, ..., x_m) &= 0 \\ f_2(x_1, x_2, ..., x_m) &= 0 \\ &\bullet\bullet\bullet \\ f_m(x_1, x_2, ..., x_m) &= 0 \end{aligned}$$

which can be written as

$$F(x) = 0 \tag{1}$$

where

$$x = (x_1, x_2, ..., x_m)^T \text{ and } F(x) = (f_1(x), f_2(x), \bullet\bullet\bullet, f_m(x))^T$$

Objective: For such systems of nonlinear equations, the objective is to find solutions, that are real-valued vectors $x \in \mathbb{R}^m$ such that $F(x) = 0$.

Applications

Zero-finding problems:

$$\text{Find } X \in \mathbb{R}^m \text{ such that } F(X) = 0$$

Optimization:

$$\min_{(x, y)} K(x, y)$$

where the objective function K can be minimized by solving

$$\nabla K = \begin{bmatrix} K_x \\ K_y \end{bmatrix} = 0$$

10.1. Fixed Points for Functions of Several Variables

Definition: A function G from $D \subset \mathbb{R}^m$ to $\mathbb{R}^m$ has a **fixed point** at $p \in \mathbb{R}^m$ if $G(p) = p$

Fixed-Point Theorem:

Let $D = \left\{ (x_1, x_2, ..., x_m)^T : a_i \le x_i \le b_i,\ i = 1, 2, \bullet\bullet\bullet, m \right\}$.

(i) Suppose G is a continuous function from $D \subset \mathbb{R}^m$ to $\mathbb{R}^m$ with the property that $\boldsymbol{G(x) \in D}$ whenever $x \in D$. Then, G has a fixed point in D.
(ii) Moreover, suppose that all the component functions of G have continuous partial derivatives and **a constant $K < 1$ exists with**

$$\left| \frac{\partial}{\partial x_j} g_i(x) \right| \le \frac{K}{m}, \quad x \in D, \text{ for all } 1 \le i, j \le m$$

Then the sequence x^k defined by an arbitrarily selected $x^0 \in D$ and generated by

$$x^k = G(x^{k-1}) \tag{2.1}$$

converges to the unique fixed point $p \in D$ and

$$\| x^k - p \|_\infty \le K^k \| x^0 - p \|_\infty$$

$$\| x^k - p \|_\infty \le \frac{K^k}{1 - K} \| x^1 - x^0 \|_\infty$$

Example: Consider the nonlinear system of equations

$$3\,x_1 - \cos(x_2\,x_3) - \frac{1}{2} = 0$$

$$x_1^2 - 81\left(x_2 + \frac{1}{10}\right)^2 + \sin(x_3) + \frac{53}{50} = 0$$

$$e^{-x_1\,x_2} + 20\,x_3 + \frac{10\,\pi - 3}{3} = 0$$

1. Place the system in a fixed-point form $x = G(x)$ by solving the i-th equation for x_i.
2. Show that there is a unique solution on $D = [-1,\ 1]^3$.
3. Find the fixed point by iterating, starting from $x^0 = (0.1,\ 0.1, -0.1)^T$ until accuracy within 10^{-5} in ℓ_∞ norm is obtained.

Solution:

(1) **Formulation of $x = G(x)$:**

$$solve\left(3\,x_1 - \cos(x_2\,x_3) - \frac{1}{2} = 0,\ x_1\right)$$

$$\frac{1}{6} + \frac{1}{3}\cos(x_2\,x_3) \qquad \textbf{(3.1)}$$

$$solve\left(x_1^2 - 81\left(x_2 + \frac{1}{10}\right)^2 + \sin(x_3) + \frac{53}{50} = 0,\ x_2\right)$$

$$-\frac{1}{10} + \frac{1}{90}\sqrt{106 + 100\,x_1^2 + 100\sin(x_3)},\ -\frac{1}{10} - \frac{1}{90}\sqrt{106 + 100\,x_1^2 + 100\sin(x_3)} \qquad \textbf{(3.2)}$$

$$solve\left(e^{-x_1\,x_2} + 20\,x_3 + \frac{10\,\pi - 3}{3} = 0,\ x_3\right)$$

$$\frac{1}{20} - \frac{1}{20}\, e^{-x_1 x_2} - \frac{1}{6}\,\pi \tag{3.3}$$

Thus

$$g1 := \frac{1}{6} + \frac{1}{3}\cos\left(x_2\, x_3\right) :$$

$$g2 := -\frac{1}{10} + \frac{1}{90}\sqrt{106 + 100\, x_1^2 + 100\sin\left(x_3\right)} :$$

$$g3 := \frac{1}{20} - \frac{1}{20}\, e^{-x_1 x_2} - \frac{1}{6}\,\pi :$$

(2) **Checking $G(x) \in D$ for $x \in D$:**

For $(x_1, x_2, x_3) \in [-1, 1]^3$,

$$|g1| \le \frac{1}{3}\left|\cos(x_2 x_3)\right| + \frac{1}{6} \le \frac{1}{2}$$

$$|g2| \le \frac{1}{90}\sqrt{106 + 100 + 100\sin(1)} - \frac{1}{10} < 0.09$$

$$|g3| = \frac{1}{20}\,e^{-x_1 x_2} + \frac{10\pi - 3}{60} \le \frac{e}{20} + \frac{10\pi - 3}{60} < 0.61$$

which proves $G(x) \in D$ for $x \in D$.

Finding bounds for partial derivatives on D:

Partial derivatives	Bounds
$diff(g1, x_1) = diff(g2, x_2) = diff(g3, x_3) = 0$	0
$diff(g1, x_2) = -\frac{1}{3}\sin(x_2 x_3)\, x_3$	$evalf\left(\frac{1}{3}\sin(1)\right) = 0.2804903282$
$diff(g1, x_3) = -\frac{1}{3}\sin(x_2 x_3)\, x_2$	$evalf\left(\frac{1}{3}\sin(1)\right) = 0.2804903282$
$diff(g2, x_1) = \frac{10}{9}\,\frac{x_1}{\sqrt{106 + 100 x_1^2 + 100\sin(x_3)}}$	$evalf\left(\frac{10}{9}\,\frac{1}{\sqrt{106 + 100\sin(-1)}}\right) = 0.2376856356$
$diff(g2, x_3) = \frac{5}{9}\,\frac{\cos(x_3)}{\sqrt{106 + 100 x_1^2 + 100\sin(x_3)}}$	$evalf\left(\frac{5}{9}\,\frac{1}{\sqrt{106 + 100\sin(-1)}}\right) = 0.1188428178$
$diff(g3, x_1) = \frac{1}{20}\, x_2\, e^{-x_1 x_2}$	$evalf\left(\frac{1}{20}\, e\right) = 0.1359140914$

$$diff\left(g3, x_2\right) = \frac{1}{20}\, x_1\, e^{-x_1 x_2} \qquad evalf\left(\frac{1}{20}\, e\right) = 0.1359140914$$

Thus

$$\left|\frac{\partial}{\partial x_j}\, g_i(x)\right| \leq 0.281, \text{ for all } 1 \leq i, j \leq 3$$

and therefore the condition in the second part of the Fixed-Point Theorem holds with

$$K = 3 \cdot 0.281 = 0.843 < 1$$

(3) Finding the fixed point:

```
FixedPointSYS := proc(G, X0, m, TOL, itmax)
    local i, k, id1, id2, X, T, maxerr;
    with(LinearAlgebra) :
    X := Matrix(m, 2);
    T := Vector(m);
    for i from 1 to m do
       X[i, 1] := X0[i];
    end do;
    if (m = 3) then
       printf(" k            x_1                  x_2                  x_3\n");
       printf("-------------------------------------------------------------------\n");
       printf("%3d   %15.10f   %15.10f    %15.10f\n", 0, X0[1], X0[2], X0[3]);
    end if;

    for k to itmax do
       id1 := modp(k + 1, 2) + 1;
       id2 := modp(k, 2) + 1;
       T := evalf[15](eval(G, [x[1] = X[1, id1], x[2] = X[2, id1], x[3] = X[3, id1]]));
       for i from 1 to m do
          X[i, id2] := T[i];
       end do;
       printf("%3d   %15.10f   %15.10f    %15.10f\n", k, X[1, id2], X[2, id2],
    X[3, id2]);
       maxerr := 0;
       for i from 1 to m do
          maxerr := max(maxerr, abs(X[i, id2] - X[i, id1]));
       end do;
       if (maxerr < TOL) then
          print(`k=`, k, `maxerr=`, maxerr);
          break;
       end if;
    end do;
end proc:
```

$$m := 3 :$$

$$g1 := \frac{1}{6} + \frac{1}{3}\cos\left(x_2\, x_3\right) :$$

$$g2 := -\frac{1}{10} + \frac{1}{90}\sqrt{106 + 100\, x_1^2 + 100\sin\left(x_3\right)} :$$

$$g3 := \frac{1}{20} - \frac{1}{20}\, e^{-x_1\, x_2} - \frac{1}{6}\,\pi :$$

$$G := [g1, g2, g3] :$$

$$X0 := \langle 0.1,\ 0.1, -0.1\rangle :$$

$$TOL := 10^{-5} :\quad itmax := 100 :$$

$$FixedPointSYS(G, X0, m, TOL, itmax) :$$

k	x_1	x_2	x_3
0	0.1000000000	0.1000000000	-0.1000000000
1	0.4999833335	0.0094411496	-0.5231012673
2	0.4999959349	0.0000255677	-0.5233633109
3	0.5000000000	0.0000123367	-0.5235981364
4	0.5000000000	0.0000000342	-0.5235984672
5	0.5000000000	0.0000000165	-0.5235987747

$$k=,\ 5,\ \ maxerr=,\ 3.075\ 10^{-7}$$

(3.4)

Empty Page

10.2. Newton's Method for Nonlinear Systems

We continue considering a system of nonlinear equations of the form

$$\begin{aligned} f_1(x_1, x_2, ..., x_m) &= 0 \\ f_2(x_1, x_2, ..., x_m) &= 0 \\ &\bullet\bullet\bullet \\ f_m(x_1, x_2, ..., x_m) &= 0 \end{aligned}$$

which can be written as

$$F(x) = 0 \qquad \textbf{(1)}$$

where

$$x = (x_1, x_2, ..., x_m)^T \text{ and } F(x) = (f_1(x), f_2(x), \bullet\bullet\bullet, f_m(x))^T$$

Objective: For such systems of nonlinear equations, the objective is to find solutions, that are real-valued vectors $p \in \mathbb{R}^m$ such that $F(p) = 0$.

Review: Solutions of Nonlinear Equations in One Variable

For equations of the form

$$f(x) = 0 \tag{2}$$

the objective is to find solutions, that are real numbers $p \in [a, b]$ such that $f(p) = 0$.

Let p be a zero of $f(x) = 0$ and p_0 an approximation of p and

$$p = p_0 + h \tag{3}$$

Our momentary concern is how to find the correction h.

If f" exists and is continues, then by Taylor's Theorem

$$0 = f(p) = f(p_0 + h) = f(p_0) + (p - p_0) f'(p_o) + \frac{(p - p_0)^2}{2} f''(\xi),$$

where ξ lies between p and p_0. If $|p - p_0|$ is small, it is reasonable to ignore the last term and solve for $p - p_0$:

$$h = p - p_0 \approx -f(p_0)/f'(p_0).$$

Then,

$$p_1 := p_0 - f(p_0)/f'(p_0)$$

may be a better approximation of p than p_0.

This has motivated the **Newton's Method**:

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \qquad n \geq 1$$

Graphical Interpretation:

Consider the tangent line passing $\left(p_0, f\left(p_0\right)\right)$:

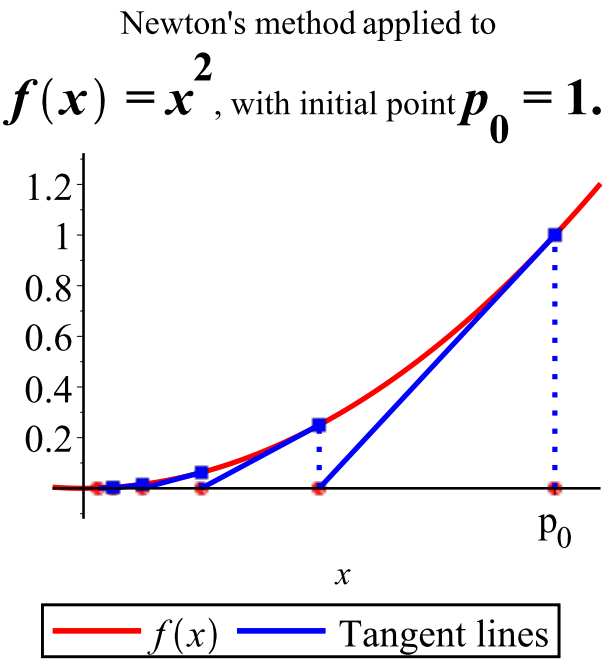
$$\ell_0(x) = f'\left(p_0\right)\left(x - p_0\right) + f\left(p_0\right).$$

Let $\ell_0(x) = 0$. Then,

$$x = p_0 - f\left(p_0\right)/f'\left(p_0\right)$$

which is p_1, the x-intercept of the tangent line $y = \ell_0(x)$.

```
with(Student[NumericalAnalysis]) :
f := x^2 :
Newton(f, x = 1, output = plot, stoppingcriterion = function_value)
```



Example of Nonconvergence:

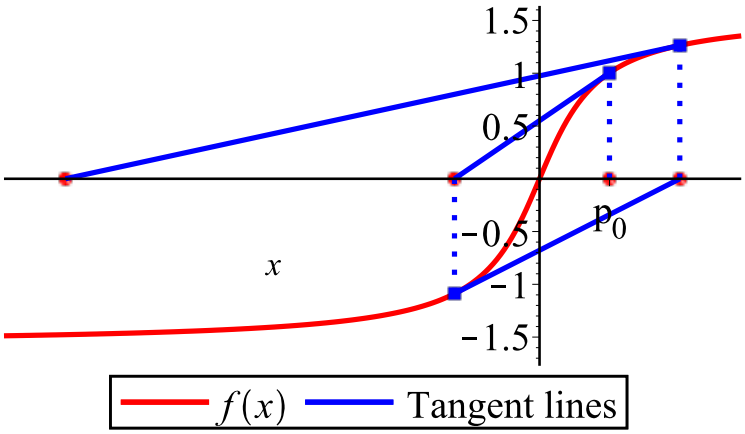
$g := \arctan(x)$:

$Newton(g, x = \pi/2, output = plot, maxiterations = 3)$

3 iterations of Newton's method applied to

$\boldsymbol{f(x) = \arctan(x)}$, with initial point

$\boldsymbol{p_0 = \pi/2}$



Notes:

• Newton's method can be interpreted as a fixed-point iteration:

$$p_n = g(p_{n-1}) := p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}.$$

• It cannot be continued if $f'(p_{n-1}) = 0$ for some n. As a matter of fact, Newton's method is most effective when f' is bounded away from zero near p.

Convergence Analysis:

Let $e_n = p_n - p$. Then,

$$e_n = p_n - p = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} - p = \frac{e_{n-1} f'(p_{n-1}) - f(p_{n-1})}{f'(p_{n-1})}.$$

By Taylor's Theorem, we have

$$0 = f(p) = f(p_{n-1} - e_{n-1}) = f(p_{n-1}) - e_{n-1} f'(p_{n-1}) + \frac{1}{2} e_{n-1}^2 f''(\xi_{n-1})$$

.

Thus,

$$e_n = \frac{1}{2} \frac{f''(\xi_{n-1})}{f'(p_{n-1})} e_{n-1}^2 \approx \frac{1}{2} \frac{f''(p)}{f'(p)} e_{n-1}^2 = C e_{n-1}^2$$

Theorem: Convergence of Newton's Method

Let $f \in C^2[a, b]$ and $p \in (a, b)$ is such that $f(p) = 0$ and $f'(p) \neq 0$. Then, there is a neighborhood of p such that if Newton's method is started p_0 in that neighborhood, it generates a convergence sequence p_n satisfying

$$|p_n - p| \leq C|p_{n-1} - p|^2,$$

for a positive constant C.

Example: Use a Maple built-in function to find the solution of $\arctan(x) = 0$, beginning from $p_0 = \pi/5$.

Solution:

```
with(Student[NumericalAnalysis]):
Newton(arctan(x), x = π/5, output = sequence, maxiterations = 5)
```

$$0.6283185308,\ -0.1541304479,\ 0.0024295539,\ -9.562\,10^{-9},\ 0.,\ 0. \qquad (4.1)$$

Example: Use Newton's method to find the square root of a positive number Q.

Solution:

Let $x = \sqrt{Q}$. Then x is a root of $x^2 - Q = 0$.
Set $f(x) = x^2 - Q$ and $f'(x) = 2x$.
The Newton's method reads

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} = p_{n-1} - \frac{p_{n-1}^2 - Q}{2p_{n-1}} = \frac{1}{2}\left(p_{n-1} + \frac{Q}{p_{n-1}}\right)$$

```
NR := proc(Q, p0, N0)
    local p, n;
    p := p0;
    for n to N0 do
       p := (p + Q/p)/2;
       print(n, evalf[14](p));
    end do;
end proc:
Q := 16 : p0 := 1 : N0 := 8 :
NR(Q, p0, N0)
```

```
1, 8.5000000000000
2, 5.1911764705882
3, 4.1366647225462
4, 4.0022575247985
5, 4.0000006366929
6, 4.0000000000000
7, 4.0000000000000
```

8, 4.0000000000000 **(5.1)**

```
p0 := -1 :
NR(Q, p0, N0)
```

1, -8.5000000000000
2, -5.1911764705882
3, -4.1366647225462
4, -4.0022575247985
5, -4.0000006366929
6, -4.0000000000000
7, -4.0000000000000
8, -4.0000000000000 **(5.2)**

Systems of Nonlinear Equations:

Newton's method for systems of nonlinear equations follows the same strategy that was used for single equation. That is, we (1) **linearize**, (2) **solve for corrections**, and (3) **update the solution**, repeating the steps as often as necessary. For an illustration, we begin with a pair of equations:

$$\begin{cases} \mathrm{f}_1\left(x_1, x_2\right) = 0 \\ \mathrm{f}_2\left(x_1, x_2\right) = 0 \end{cases}$$

Supposing that $\left(x_1, x_2\right)$ is an approximate solution of the system, let us computer corrections $\left(h_1, h_2\right)$ so that $\left(x_1 + h_1, x_2 + h_2\right)$ will be a better approximate solution.

$$\begin{cases} 0 = \mathrm{f}_1\left(x_1 + h_1, x_2 + h_2\right) \approx \mathrm{f}_1\left(x_1, x_2\right) + h_1 \dfrac{\partial f_1}{\partial x_1} + h_2 \dfrac{\partial f_1}{\partial x_2} \\ 0 = \mathrm{f}_2\left(x_1 + h_1, x_2 + h_2\right) \approx \mathrm{f}_2\left(x_1, x_2\right) + h_1 \dfrac{\partial f_2}{\partial x_1} + h_2 \dfrac{\partial f_2}{\partial x_2} \end{cases}$$

The above can be rewritten as

$$0 \approx \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} + \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$$

This implies that the correction vector $\left(h_1, h_2\right)^T$ can be found by

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = -J\left(x_1, x_2\right)^{-1} \begin{bmatrix} f_1\left(x_1, x_2\right) \\ f_2\left(x_1, x_2\right) \end{bmatrix}$$

where J is the **Jacobian** of $\left(f_1, f_2\right)^T$ at $\left(x_1, x_2\right)$:

$$J\left(x_1, x_2\right) = \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \end{bmatrix} \left(x_1, x_2\right).$$

Hence, **Newton's method for two nonlinear equations in two variables** is

$$\begin{bmatrix} x_1^n \\ x_2^n \end{bmatrix} = \begin{bmatrix} x_1^{n-1} \\ x_2^{n-1} \end{bmatrix} + \begin{bmatrix} h_1^{n-1} \\ h_2^{n-1} \end{bmatrix},$$

where

$$\begin{bmatrix} h_1^{n-1} \\ h_2^{n-1} \end{bmatrix} = -J\left(x_1^{n-1}, x_2^{n-1}\right)^{-1} \begin{bmatrix} f_1\left(x_1^{n-1}, x_2^{n-1}\right) \\ f_2\left(x_1^{n-1}, x_2^{n-1}\right) \end{bmatrix}$$

In general, the system of m nonlinear equations,

$$f_i(x_1, x_2, \cdots, x_m) = 0, \quad 1 \le i \le m,$$

can be expressed as in Equation **(1)**:

$$F(x) = 0 \tag{4}$$

where $x = (x_1, x_2, \cdots, x_m)^T$ and $F = (f_1, f_2, \cdots, f_m)^T$.

Then

$$0 = F(x + h) \approx F(x) + F'(x) \cdot h,$$

where $h = (h_1, h_2, \cdots, h_m)^T$ and $F'(x)$ is the Jacobian of F at x:

$$J(x) = F'(x) = \left[\frac{\partial f_i}{\partial x_j}\right](x).$$

The correction vector h is obtained as

$$\mathrm{h} = -F'(x)^{-1} F(x).$$

Hence,

Newton's method for m nonlinear equations in m variables is given by

$$x^n = x^{n-1} + h^{n-1} \tag{5}$$

where h^{n-1} is the solution of the linear system:

$$F'\left(x^{n-1}\right) h^{n-1} = -F\left(x^{n-1}\right).$$

Example: Starting with $(1, 1, 1)^T$, carry out 6 iterations of Newton's method to find a root of the nonlinear system

$$\begin{cases} x\,y = z^2 + 1 \\ x\,y\,z + y^2 = x^2 + 2 \\ e^x + z = e^y + 3 \end{cases}$$

Solution:

```
NewtonRaphsonSYS := proc(X, F, X0, TOL, itmax)
   local Xn, H, FX, J, i, m, n, Err;

    m  := LinearAlgebra[Dimension](Vector(X) );
    Xn := Vector(m);
    H  := Vector(m);
    FX := Vector(m);
    J  := Matrix(m, m);
    Xn := X0;

    for n to itmax do
       FX := eval(F, [seq(X[i] = Xn[i], i = 1 ..m) ]);
       J  := evalf15(VectorCalculus[Jacobian](F, X = convert(Xn, list)) );
       H  := -J^(-1).Vector(FX);
       Xn := Xn + H;

       printf(" %3d    %.8f ", n, Xn[1]);
       for i from 2 to m do; printf("  %.8f ", Xn[i]); end do;
       for i from 1 to m do; printf("  %.3g ", H[i]); end do;
       printf("\n");
       if (LinearAlgebra[VectorNorm](H, 2) < TOL) then break end if:
   end do;
end proc:

F := [x*y - z^2 - 1, x*y*z + y^2 - x^2 - 2, e^x + z - e^y - 3]:
X := [x, y, z]:
X0 := <1, 1, 1>:
TOL := 10^(-8): itmax := 10:

NewtonRaphsonSYS(X, F, X0, TOL, itmax):
  1   2.18932610   1.59847516   1.39390063   1.19   0.598   0.394
  2   1.85058965   1.44425142   1.27822400   -0.339   -0.154   -0.116
```

3	1.78016120	1.42443598	1.23929244	-0.0704	-0.0198	-0.0389
4	1.77767471	1.42396093	1.23747382	-0.00249	-0.000475	-0.00182
5	1.77767192	1.42396060	1.23747112	-2.79e-006	-3.28e-007	-2.7e-006
6	1.77767192	1.42396060	1.23747112	-3.14e-012	-4.22e-014	-4.41e-012

Note: The convergence rate is

$$|x^n - p| \leq C\ |x^{n-1} - p|^2$$

for x^0 near to p.

Using Maple for Initial Approximation

$F := [x\,y - z^2 - 1,\ x\,y\,z + y^2 - x^2 - 2,\ e^x + z - e^y - 3]$:

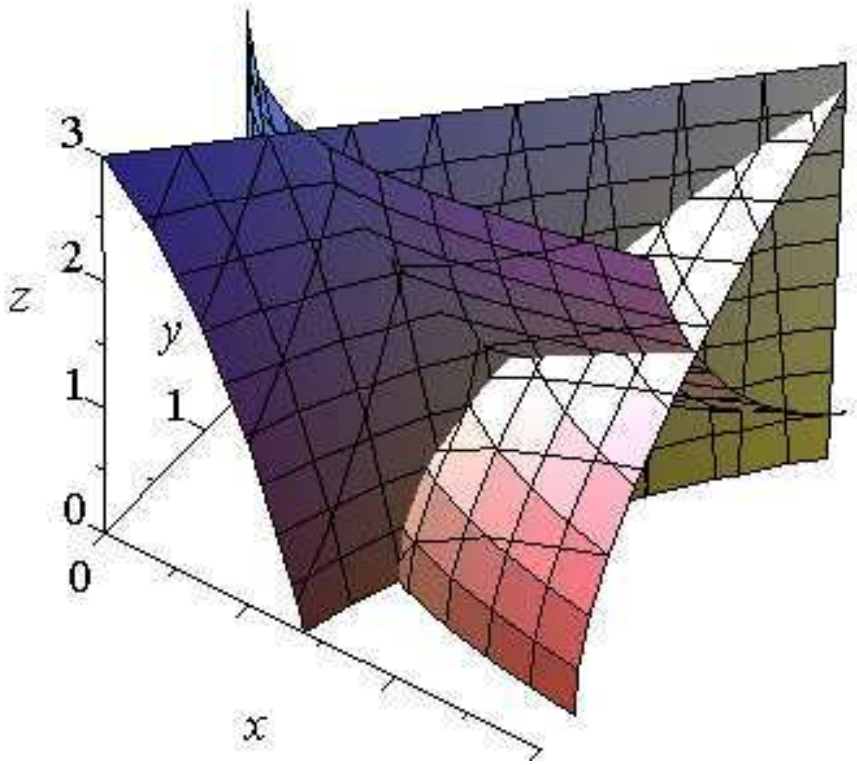
$eq1 := x\,y - z^2 - 1 = 0$:

$eq2 := x\,y\,z + y^2 - x^2 - 2 = 0$:

$eq3 := e^x + z - e^y - 3 = 0$:

$with(plots)$:

$implicitplot3d(\{eq1, eq2, eq3\}, x = 0..3, y = 0..3, z = 0..3, axes = normal, axesfont = [helbetica, 15], labelfont = [helbetica, 15], orientation = [-55, 45, 0], viewpoint = [\text{"circleleft"}, frames = 30])$;



Applications to Optimization

Consider the optimization problem of the form

$$\min_{x,\,y} K(x,\,y) \tag{6}$$

When we assume that the objective function K is differentiable, the minimizer $(x,\,y)^T$ would satisfy

$$\nabla K(x,\,y) \;=\; \begin{bmatrix} K_x \\ K_y \end{bmatrix}(x,\,y) \;=\; 0$$

which may be viewed as a zero-finding problem for a system of two nonlinear equations.

Thus, the corresponding Newton's method is formulated as

$$\begin{bmatrix} x^{n+1} \\ y^{n+1} \end{bmatrix} = \begin{bmatrix} x^n \\ y^n \end{bmatrix} + \begin{bmatrix} h_1^n \\ h_2^n \end{bmatrix}$$

where the correction vector is the solution of the linear system:

$$J^n \begin{bmatrix} h_1^n \\ h_2^n \end{bmatrix} = -\begin{bmatrix} K_x\left(x^n,\,y^n\right) \\ K_y\left(x^n,\,y^n\right) \end{bmatrix}$$

Here J^n is call the Hessian matrix of K evaluated at $\left(x^n,\,y^n\right)$ defined by

$$J^n := \begin{bmatrix} K_{xx} & K_{xy} \\ K_{yx} & K_{yy} \end{bmatrix}\left(x^n,\,y^n\right)$$

The objective function $K(x, y)$

In many applications, the objective function K is quite complicated; an example can be found in mesh optimization. In this case, it would be cumbersome and time-consuming to explicitly compute the derivatives of K. The derivatives can be approximated by using finite difference method:

For example,

$$K_x(x, y) = \frac{K(x+h, y) - K(x-h, y)}{2h} - \frac{h^2}{6} K_{xxx}(x, y) + O(h^4)$$

$$K_{xx}(x, y) = \frac{K(x-h, y) - 2K(x, y) + K(x+h, y)}{h^2} - \frac{h^2}{12} K_{xxxx}(x, y) + O(h^4)$$

$$K_{xy}(x, y) = \frac{K_{NE} + K_{SW} - K_{NW} - K_{SE}}{4h^2} + O(h^2)$$

We will deal with details in Section 11.2 below.

10.3. Quasi-Newton Methods

Nonlinear equation of single variable:

$$f(x) = 0 \tag{1}$$

Newton's Method: Given an initial estimate x^0

$$x^n = x^{n-1} - \left[f'\left(x^{n-1}\right)\right]^{-1} f\left(x^{n-1}\right) \qquad (n \geq 1)$$

• Convergence:

$$\left|e^n\right| \leq C\left|e^{n-1}\right|^2 \tag{1.1}$$

• Evaluation of f' may not be convenient.

Secant method: For initial estimates x^0 and x^1,

$$x^n = x^{n-1} - \left[\frac{f\left(x^{n-1}\right) - f\left(x^{n-2}\right)}{x^{n-1} - x^{n-2}}\right]^{-1} f\left(x^{n-1}\right) \qquad (n \geq 2)$$

• It requires only one new evaluation of f per step.
• Convergence:

$$\left|e^n\right| \leq C\left|e^{n-1}\right|^{1.62} \tag{1.2}$$

Nonlinear equation of muitiple variables:

$$F(x) = 0 \tag{2}$$

Newton's Method: Given an initial estimate x^0

$$x^n = x^{n-1} - \left[J\left(x^{n-1}\right)\right]^{-1} F\left(x^{n-1}\right) \qquad (n \geq 1)$$

- Convergence:

$$\left|e^n\right| \leq C \left|e^{n-1}\right|^2 \tag{2.1}$$

- Evaluation of the Jacobian $J(x)$ may not be convenient.
- Roundoff error: self-correcting
- The inversion of Jacobian requires $\mathrm{O}\left(m^3\right)$ operations.

Quasi-Newton (Broyden's) method: For initial estimates x^0 and x^1,

$$x^n = x^{n-1} - \left[A_{n-1}\right]^{-1} F\left(x^{n-1}\right) \qquad (n \geq 2)$$

- It does not need to compute partial derivatives, and Jacobian
- Each iteration requires $\mathrm{O}\left(m^2\right)$ operations
- Convergence: superlinear, i.e.,

$$\frac{\left|e^n\right|}{\left|e^{n-1}\right|} \to 0 \quad \text{as } n \to \infty$$

- Roundoff error: not self-correcting
- For many applications, Quasi-Newton method is more convenient and faster.

Details of Quasi-Newton Method:

Given an initial estimate x^0,

Computation of x^1:

Use the Newton's method:

$$x^1 = x^0 - \left[J(x^0)\right]^{-1} F(x^0)$$

If it is inconvenient to determine $J(x^0)$ exactly, use finite difference approximations to the partial derivatives

$$J(x^0) \equiv \left[\frac{\partial}{\partial x_j} f_i(x^0)\right] \approx \left[\frac{f_i(x^0 + h\, e_j) - f_i(x^0)}{h}\right] \equiv A_0$$

for $i, j = 1, 2, \bullet\bullet\bullet, m$.

Computation of x^2:

$$x^2 = x^1 - \left[A_1\right]^{-1} F(x^1)$$

where $A_1 \in \mathbb{R}^{m \times m}$ satisfies

$$A_1(x^1 - x^0) = F(x^1) - F(x^0)$$

$$A_1 z = A_0 z \text{ whenever } (x^1 - x^0)^T z = 0$$

A matrix A_1 satisfying the above two conditions can be uniquely determined as

$$A_1 = A_0 + \frac{\left[F(x^1) - F(x^0) - A_0(x^1 - x^0)\right](x^1 - x^0)^T}{\|x^1 - x^0\|_2^2}$$

Note that A_1 is an update of A_0 which satisfies the two weird conditions. However, they are not weird as a matter of fact. The first condition reminds us of the secant method for equations of one variable:

$$A_1 = \frac{F(x^1) - F(x^0)}{x^1 - x^0} \approx F'(x^1)$$

The second condition enforces the new matrix A_1 to be updated with no change in a direction orthogonal to $(x^1 - x^0)$.

Computation of x^{n+1}:

$$x^{n+1} = x^n - [A_n]^{-1} F(x^n)$$

where $A_n \in \mathbb{R}^{m \times m}$ reads

$$A_n = A_{n-1} + \frac{1}{\|\Delta x^n\|_2^2} \left(\Delta F^n - A_{n-1}\Delta x^n\right)\left(\Delta x^n\right)^T$$

where

$$\Delta F^n = F(x^n) - F(x^{n-1}), \quad \Delta x^n = x^n - x^{n-1}$$

Note the inversion of A_n requires $O(m^3)$ operations. However a considerable improvement can be incorporated by employing a matrix inversion formula of Sherman and Morrison.

Sherman-Morrison Formula:

Suppose A is nonsingular matrix and x and y are vectors with $y^T A^{-1} x \neq -1$. Then $A + x\,y^T$ is nonsingular and

$$(A + x\,y^T)^{-1} = A^{-1} - \frac{A^{-1} x\, y^T A^{-1}}{1 + y^T A^{-1} x}$$

Letting $x = \left(\Delta F^{n} - A_{n-1}\Delta x^{n}\right) / \left\|\Delta x^{n}\right\|_2^2$ and $y = \Delta x^{n}$,

$$\left(A_n\right)^{-1} = \left(A_{n-1} + \frac{1}{\left\|\Delta x^{n}\right\|_2^2}\left(\Delta F^{n} - A_{n-1}\Delta x^{n}\right)\left(\Delta x^{n}\right)^{T}\right)^{-1}$$

$$= \left(A_{n-1}\right)^{-1} - \frac{\left(A_{n-1}\right)^{-1}\left(\Delta F^{n} - A_{n-1}\Delta x^{n}\right)\left(\Delta x^{n}\right)^{T}\left(A_{n-1}\right)^{-1} / \left\|\Delta x^{n}\right\|_2^2}{1 + \left(\Delta x^{n}\right)^{T}\left(A_{n-1}\right)^{-1}\left(\Delta F^{n} - A_{n-1}\Delta x^{n}\right) / \left\|\Delta x^{n}\right\|_2^2}$$

Thus we have

$$\left(A_n\right)^{-1} = \left(A_{n-1}\right)^{-1} + \frac{\left(\Delta x^{n} - \left(A_{n-1}\right)^{-1}\Delta F^{n}\right)\left(\Delta x^{n}\right)^{T}\left(A_{n-1}\right)^{-1}}{\left(\Delta x^{n}\right)^{T}\left(A_{n-1}\right)^{-1}\Delta F^{n}}$$

which requires $\mathrm{O}\left(m^2\right)$ operations.

Example: Starting with $(1,\ 1,\ 1)^{T}$, carry out five iterations of Quasi-Newton method to approximate a root of the nonlinear system

$$\begin{cases} x\,y = z^2 + 1 \\ x\,y\,z + y^2 = x^2 + 2 \\ \mathrm{e}^{x} + z = \mathrm{e}^{y} + 3 \end{cases}$$

Solution

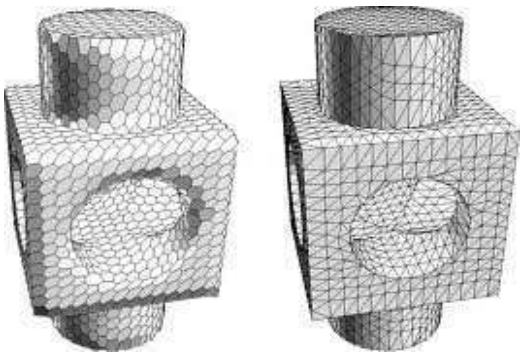
Homework

Empty Page

10.4. Mesh Optimization

What is meshing?

Given an input domain, partition it into simple cells:
triangles, quadrilaterals, tetrahedra, cuboids, ...



What is the “right” quality measure?

Not fully settled, varies by meshing applications

Avoid sharp angles? Allow sharp angles but avoid obtuse angles?

Use non-directional quality measures?
Align mesh elements with domain boundaries?
Concentrate quality near boundary, allow worse elements in interior?

Typical mesh generation stages

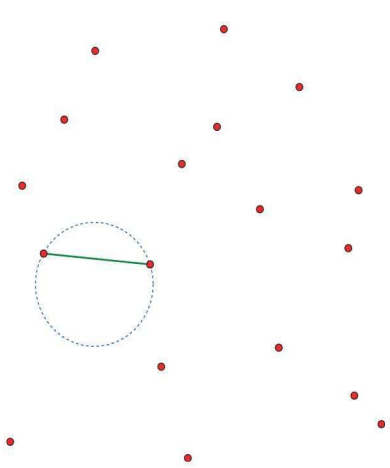
Generate initial point placement
- Well spaced, other quality considerations

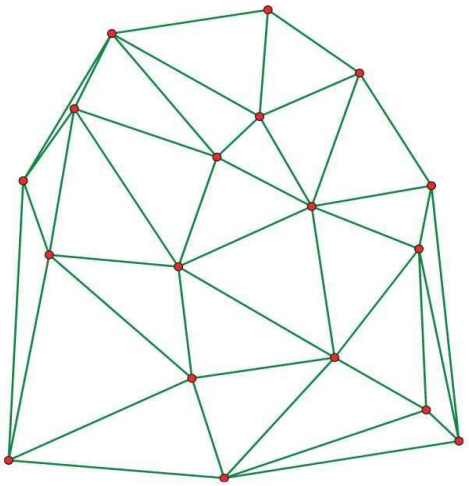
Determine mesh connectivity
- Usually, Delaunay triangulation

Iterate mesh improvement stages
- Laplacian or optimization-based smoothing
- Flips and other connectivity changes

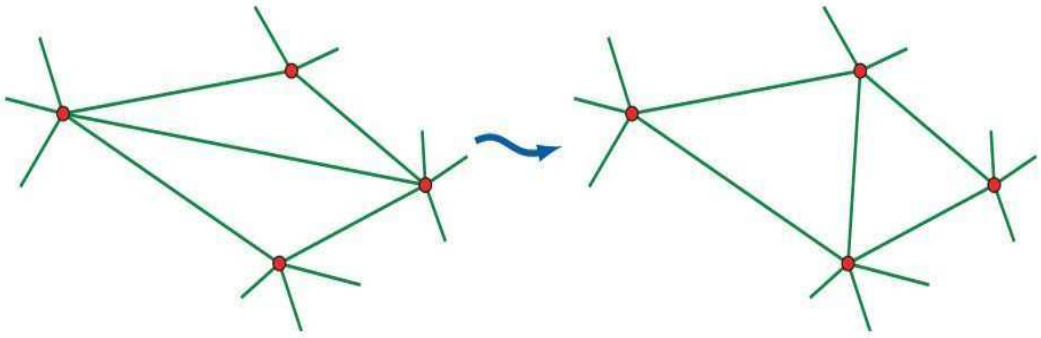
Delaunay Triangulation:

1. Given planar point set, connect two points by edge if some circle exists with them on boundary, empty interior
2. Collection of all such edges for points in general position





Flipping (beginning of mesh optimization)



Mesh Optimization

Mesh optimization can be carried out by (1) defining a quality measure for triangles, (2) forming an objective function, and (3) maximizing/minimizing the objective function. Here the goal is to move vertices so that the resulting mesh includes as uniform triangles as possible.

Quality Measure for Triangles

Let T a triangle in the physical space whose vertices are given by

$$X_k = \left(x_k, y_k\right)^T, \quad k = 0,\ 1,\ 2$$

Let T_R be the reference triangle with vertices

$$u_0 = (0,\ 0)^T, \quad u_1 = (1,\ 0)^T, \quad u_2 = (0,\ 1)^T$$

Then, if we choose X_0 as the translation vector, the affine map that takes T_R to T is

$$X = Au + X_0$$

where A is the Jacobian matrix of the affine map, referenced to node X_0, and expressed as

$$A = \begin{bmatrix} X_1 - X_0 & X_2 - X_0 \end{bmatrix} = \begin{bmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{bmatrix}$$

Let now T_I be an equilateral triangle with all its edges of length one and vertices located at

$$v_0 = (0, 0)^T, \quad v_1 = (1, 0)^T, \quad v_2 = \left(1/2, \sqrt{3}/2\right)^T$$

Let $v = W u$ be the linear map that takes T_R to T_I; its Jacobian matrix

$$W := \begin{bmatrix} 1 & 1/2 \\ 0 & \sqrt{3}/2 \end{bmatrix} :$$

Then the affine map that takes T_I to T is given by

$$X = S\, v + X_0 \qquad \textbf{(1)}$$

where the Jacobian matrix is

$$S = A\, W^{-1}$$

Remark: The matrix S is independent of the node chosen as reference; it is said to be **node invariant.**

Definition: A **quality measure** of the triangle T can be defined as

$$q = \frac{2\,\sigma}{|S|^2} \tag{2}$$

where $|S|$ is the Frobenius norm of S and σ denotes the determinant of S:

$$|S| = \sqrt{tr\left(S^T S\right)}, \qquad \sigma = det(S)$$

Example

Consider a triangle of vertices

$$X_0 := \langle 0, 0\rangle = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad X_1 := \langle 2, 0\rangle = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \quad X_2 := \left\langle 1, \frac{1}{2}\right\rangle = \begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix}$$

Then, $A := \left\langle X_1 - X_0 \middle| X_2 - X_0 \right\rangle = \begin{bmatrix} 2 & 1 \\ 0 & \frac{1}{2} \end{bmatrix}$.

Since $W := \begin{bmatrix} 1 & 1/2 \\ 0 & \sqrt{3}/2 \end{bmatrix}$:

$S := A.W^{-1}$

$$\begin{bmatrix} 2 & 0 \\ 0 & \frac{1}{3}\sqrt{3} \end{bmatrix} \tag{1.1}$$

$$Snorm := \sqrt{LinearAlgebra[Trace]\left(S^{\%T}.S\right)} = \frac{1}{3}\sqrt{39}$$

$$\sigma := LinearAlgebra[Determinant](S) = \frac{2}{3}\sqrt{3}$$

Thus, the triangle quality is

```
> q := 2*σ/Snorm^2; evalf(q);
```

$$q := \frac{4}{13}\sqrt{3}$$

$$0.5329387102 \tag{1.2}$$

Another Example

Consider the ideal triangle with vertices being oriented in the opposite way

$$X_0 := \langle 0, 0\rangle = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad X_1 := \left\langle \frac{1}{2}, \frac{\sqrt{3}}{2} \right\rangle = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2}\sqrt{3} \end{bmatrix} \quad X_2 := \langle 1, 0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Then, $A := \langle X_1 - X_0 | X_2 - X_0 \rangle = \begin{bmatrix} \frac{1}{2} & 1 \\ \frac{1}{2}\sqrt{3} & 0 \end{bmatrix}$.

Since $W := \begin{bmatrix} 1 & 1/2 \\ 0 & \sqrt{3}/2 \end{bmatrix}$:

$S := A.W^{-1}$

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{2}\sqrt{3} \\ \frac{1}{2}\sqrt{3} & -\frac{1}{2} \end{bmatrix} \tag{2.1}$$

$Snorm := \sqrt{LinearAlgebra[Trace]\left(S^{\%T}.S\right)} = \sqrt{2}$

$\sigma := LinearAlgebra[Determinant](S) = -1$

Thus, the triangle quality is

```
> q := 2 σ / Snorm^2;
```

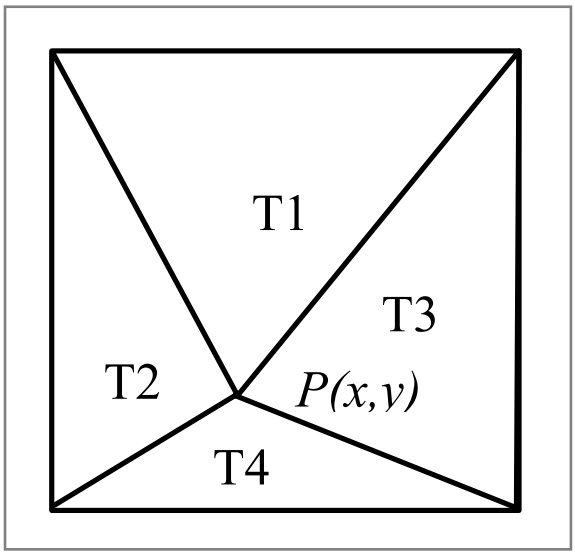
$$q := -1 \tag{2.2}$$

Remarks:

1. The quality measure ranges in $[-1, 1]$, i.e., $q \in [-1, 1]$.
2. For triangles with vertices positively oriented, $0 < q \leq 1$.

Objective Functions

For simplicity, we will deal with a simple example, illustrated in the figure below.



Our objective is to relocate the vertex $P(x, y)$ so that the quality measure of every triangle is maximized (equivalently, the reciprocal of the quality measure is minimized).

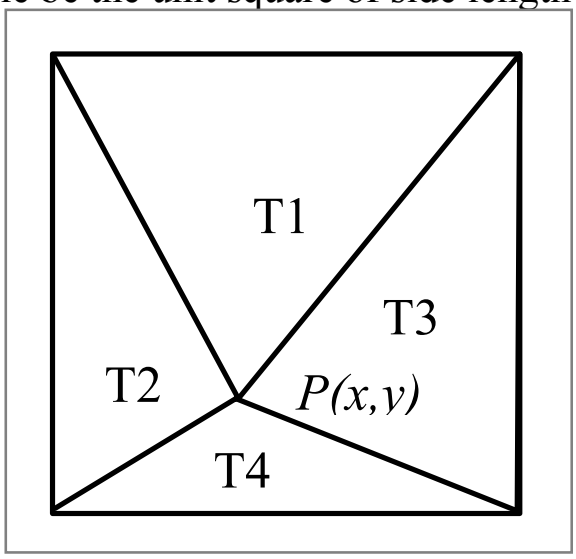
Let $\frac{|S_m|^2}{2\sigma_m}$ be the objective function (**distortion measure**) associated to m-th triangle. Then **the objective function for the local mesh** (submesh) is defined as

$$K(x, y) = \left[\sum_{m=1}^{M} \left(\frac{|S_m|^2}{2\sigma_m} \right)^p (x, y) \right]^{1/p}$$

where p is an integer, typically $p = 1$ or $p = 2$.

Example

Let the outer rectangle be the unit square of side length 1.



Triangle 1:

restart

$$X_0 := \langle x, y\rangle = \begin{bmatrix} x \\ y \end{bmatrix} \quad X_1 := \langle 1, 1\rangle = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad X_2 := \langle 0, 1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Then, $A := \left\langle X_1 - X_0 \middle| X_2 - X_0 \right\rangle = \begin{bmatrix} 1-x & -x \\ 1-y & 1-y \end{bmatrix}$.

Since $W := \begin{bmatrix} 1 & 1/2 \\ 0 & \sqrt{3}/2 \end{bmatrix}$:

$S := A.W^{-1}$

$$\begin{bmatrix} 1-x & -\frac{1}{3}(1-x)\sqrt{3} - \frac{2}{3}x\sqrt{3} \\ 1-y & \frac{1}{3}(1-y)\sqrt{3} \end{bmatrix} \tag{3.1.1}$$

$Snorm := \sqrt{LinearAlgebra[Trace]\left(S^{\%T}.S\right)}$

$$\frac{1}{3}\sqrt{9(1-x)^2 + 12(1-y)^2 + 9\left(-\frac{1}{3}(1-x)\sqrt{3} - \frac{2}{3}x\sqrt{3}\right)^2} \tag{3.1.2}$$

$\sigma := LinearAlgebra[Determinant](S)$

$$\frac{2}{3}\sqrt{3} - \frac{2}{3}\sqrt{3}\,y \tag{3.1.3}$$

Thus, the triangle quality is

$> q1 := \frac{2\,\sigma}{Snorm^2};$

$$q1 := \frac{18\left(\frac{2}{3}\sqrt{3} - \frac{2}{3}\sqrt{3}\,y\right)}{9\,(1-x)^2 + 12\,(1-y)^2 + 9\left(-\frac{1}{3}\,(1-x)\sqrt{3} - \frac{2}{3}\,x\sqrt{3}\right)^2} \tag{3.1.4}$$

$d1 := \frac{1}{q1}$

$$\frac{1}{18}\,\frac{9\,(1-x)^2 + 12\,(1-y)^2 + 9\left(-\frac{1}{3}\,(1-x)\sqrt{3} - \frac{2}{3}\,x\sqrt{3}\right)^2}{\frac{2}{3}\sqrt{3} - \frac{2}{3}\sqrt{3}\,y} \tag{3.1}$$

Triangle 2:

$X_0 := \langle x, y\rangle = \begin{bmatrix} x \\ y \end{bmatrix}$ $X_1 := \langle 0, 1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ $X_2 := \langle 0, 0\rangle = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Then, $A := \left\langle X_1 - X_0 \middle| X_2 - X_0 \right\rangle = \begin{bmatrix} -x & -x \\ 1-y & -y \end{bmatrix}$.

Since $W := \begin{bmatrix} 1 & 1/2 \\ 0 & \sqrt{3}/2 \end{bmatrix}$:

$S := A.W^{-1}$

$$\begin{bmatrix} -x & -\frac{1}{3}\,x\sqrt{3} \\ 1-y & -\frac{1}{3}\,(1-y)\sqrt{3}-\frac{2}{3}\sqrt{3}\,y \end{bmatrix} \quad \textbf{(3.2.1)}$$

$$Snorm := \sqrt{LinearAlgebra[Trace]\left(S^{\%T}.S\right)}$$

$$\frac{1}{3}\sqrt{12\,x^2+9\,(1-y)^2+9\left(-\frac{1}{3}\,(1-y)\sqrt{3}-\frac{2}{3}\sqrt{3}\,y\right)^2} \quad \textbf{(3.2.2)}$$

$$\sigma := LinearAlgebra[Determinant](S)$$

$$\frac{2}{3}\,x\sqrt{3} \quad \textbf{(3.2.3)}$$

Thus, the triangle quality is

$$> q2 := \frac{2\,\sigma}{Snorm^2};$$

$$q2 := \frac{12\,x\sqrt{3}}{12\,x^2+9\,(1-y)^2+9\left(-\frac{1}{3}\,(1-y)\sqrt{3}-\frac{2}{3}\sqrt{3}\,y\right)^2} \quad \textbf{(3.2.4)}$$

$$d2 := \frac{1}{q2}$$

$$\frac{1}{36}\,\frac{\sqrt{3}\left(12\,x^2+9\,(1-y)^2+9\left(-\frac{1}{3}\,(1-y)\sqrt{3}-\frac{2}{3}\sqrt{3}\,y\right)^2\right)}{x} \quad \textbf{(3.2)}$$

Triangle 3:

$$X_0 := \langle x, y\rangle = \begin{bmatrix} x \\ y \end{bmatrix} \quad X_1 := \langle 1, 0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad X_2 := \langle 1, 1\rangle = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Then, $A := \left\langle X_1 - X_0 \middle| X_2 - X_0 \right\rangle = \begin{bmatrix} 1-x & 1-x \\ -y & 1-y \end{bmatrix}$.

Since $W := \begin{bmatrix} 1 & 1/2 \\ 0 & \sqrt{3}/2 \end{bmatrix}$:

$S := A.W^{-1}$

$$\begin{bmatrix} 1-x & \frac{1}{3}(1-x)\sqrt{3} \\ -y & \frac{1}{3}\sqrt{3}\,y + \frac{2}{3}(1-y)\sqrt{3} \end{bmatrix} \quad \textbf{(3.3.1)}$$

$Snorm := \sqrt{LinearAlgebra[Trace]\left(S^{\%T}.S\right)}$

$$\frac{1}{3}\sqrt{12(1-x)^2 + 9y^2 + 9\left(\frac{1}{3}\sqrt{3}\,y + \frac{2}{3}(1-y)\sqrt{3}\right)^2} \quad \textbf{(3.3.2)}$$

$\sigma := LinearAlgebra[Determinant](S)$

$$\frac{2}{3}\sqrt{3} - \frac{2}{3}x\sqrt{3} \quad \textbf{(3.3.3)}$$

Thus, the triangle quality is

> $q3 := \dfrac{2\sigma}{Snorm^2}$;

$$q3 := \frac{18\left(\frac{2}{3}\sqrt{3} - \frac{2}{3}x\sqrt{3}\right)}{12(1-x)^2 + 9y^2 + 9\left(\frac{1}{3}\sqrt{3}\,y + \frac{2}{3}(1-y)\sqrt{3}\right)^2} \quad \textbf{(3.3.4)}$$

$d3 := \dfrac{1}{q3}$

$$\frac{1}{18}\,\frac{12(1-x)^2 + 9y^2 + 9\left(\frac{1}{3}\sqrt{3}\,y + \frac{2}{3}(1-y)\sqrt{3}\right)^2}{\frac{2}{3}\sqrt{3} - \frac{2}{3}x\sqrt{3}} \quad \textbf{(3.3)}$$

Triangle 4:

$X_0 := \langle x, y \rangle = \begin{bmatrix} x \\ y \end{bmatrix}$ $X_1 := \langle 0, 0 \rangle = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $X_2 := \langle 1, 0 \rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Then, $A := \langle X_1 - X_0 | X_2 - X_0 \rangle = \begin{bmatrix} -x & 1-x \\ -y & -y \end{bmatrix}$.

Since $W := \begin{bmatrix} 1 & 1/2 \\ 0 & \sqrt{3}/2 \end{bmatrix}$:

$S := A.W^{-1}$

$$\begin{bmatrix} -x & \frac{1}{3} x \sqrt{3} + \frac{2}{3} (1-x) \sqrt{3} \\ -y & -\frac{1}{3} \sqrt{3}\, y \end{bmatrix} \quad \textbf{(3.4.1)}$$

$Snorm := \sqrt{LinearAlgebra[Trace](S^{\%T}.S)}$

$$\frac{1}{3} \sqrt{9x^2 + 12y^2 + 9\left(\frac{1}{3} x \sqrt{3} + \frac{2}{3} (1-x) \sqrt{3}\right)^2} \quad \textbf{(3.4.2)}$$

$\sigma := LinearAlgebra[Determinant](S)$

$$\frac{2}{3} \sqrt{3}\, y \quad \textbf{(3.4.3)}$$

Thus, the triangle quality is

```
> q4 := 2*σ/Snorm^2;
```

$$q4 := \frac{12 \sqrt{3}\, y}{9x^2 + 12y^2 + 9\left(\frac{1}{3} x \sqrt{3} + \frac{2}{3} (1-x) \sqrt{3}\right)^2} \quad \textbf{(3.4.4)}$$

```
>
```

$d4 := \frac{1}{q4}$

$$\frac{1}{36}\,\frac{\sqrt{3}\left(9x^2+12y^2+9\left(\frac{1}{3}x\sqrt{3}+\frac{2}{3}(1-x)\sqrt{3}\right)^2\right)}{y} \tag{3.4}$$

Now, the objective function *K*

$p := 1:$

$K := (x, y) \to \left(d1^p + d2^p + d3^p + d4^p\right)^{1/p}$

$$(x, y) \to \left(d1^p + d2^p + d3^p + d4^p\right)^{\frac{1}{p}} \tag{3.5}$$

$simplify(K(x, y))$

$$-\frac{1}{3}\,\frac{1}{(y-1)\,x\,(x-1)\,y}\left(\left(2x^2+2y^2-2xy-x+y^4-2x^3-2y^3+2y^2x\right.\right. \\ \left.\left.+2x^2y-2x^2y^2+x^4-y\right)\sqrt{3}\right) \tag{3.6}$$

Now, let us minimize K

```
with(VectorCalculus) :
G := simplify(Gradient(simplify(K(x, y)), [x, y])) :
```

$$G(1)$$

$$-\frac{1}{3}\,\frac{1}{(y-1)\,x^2\,(x-1)^2\,y}\left(\sqrt{3}\,\left(-x^2+2\,y^2+2\,x\,y+y^4+4\,x^3-2\,y^3-4\,y^2\,x+4\,x\,y^3-2\,x\,y^4-5\,x^4+2\,x^5-y\right)\right) \quad (3.7)$$

$$G(2)$$

$$\frac{1}{3}\,\frac{1}{(y-1)^2\,x\,(x-1)\,y^2}\left(\sqrt{3}\,\left(-2\,x^2+y^2-2\,x\,y+x+5\,y^4+2\,x^3-4\,y^3+4\,x^2\,y-4\,x^3\,y+2\,x^4\,y-x^4-2\,y^5\right)\right) \quad (3.8)$$

$$Sol := solve(\{G(1)=0,\ G(2)=0\},\ \{x, y\})$$

$$\left\{x=\frac{1}{2},\ y=\frac{1}{2}\right\},\ \left\{x=\frac{1}{2},\ y=\frac{1}{2}\ RootOf\left(\_Z^4-4\,\_Z^3+4\,\_Z^2+3\right)\right\},\ \left\{x=\frac{1}{2}\ RootOf\left(\_Z^4-4\,\_Z^3+4\,\_Z^2+3\right),\ y=\frac{1}{2}\right\} \quad (3.9)$$

$$evalf\,(Sol[1])$$

$$\{x=0.5000000000,\ y=0.5000000000\} \quad (3.10)$$

$$evalf\,(Sol[2])$$

$$\{x=0.5000000000,\ y=1.11237243569579+0.353553390593274\,I\} \quad (3.11)$$

$$evalf\,(Sol[3])$$

$$\{x=1.11237243569579+0.353553390593274\,I,\ y=0.5000000000\} \quad (3.12)$$

Note: One can also solve $G(x, y) = 0$ using the Newton method

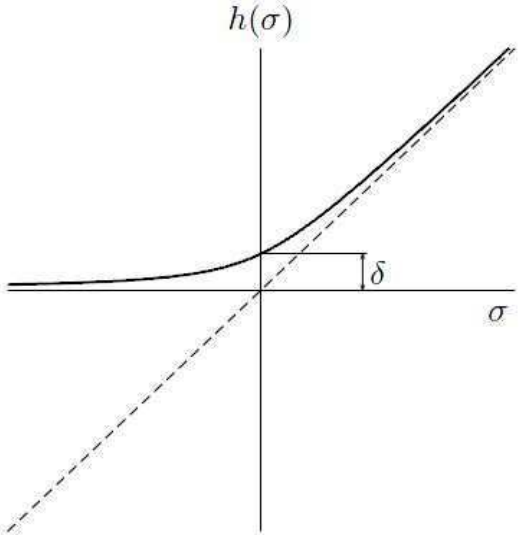
Tangled (inverted) Mesh

When $\sigma = det(S) < 0$, the triangle is said to be **tangled** (or **inverted**).
The objective function K presents a barrier in the boundary of the feasible region. This barrier avoids the optimization
method to create a tangled mesh when it starts with a valid one; however, on the other hand, it prevents the algorithm to untangle it when there are inverted elements. Therefore, the objective function is only appropriate to improve the quality of a valid mesh, not to untangle it.

To construct an objective function applicable to deal with tangled meshes, the objective function can be modified appropriately. In the literature, the modification lies in substituting σ by the positive and increasing function

$$h(\sigma) = \frac{1}{2}\sigma + \frac{1}{2}\sqrt{\sigma^2 + 4\delta^2} \qquad \textbf{(3)}$$

where δ is a parameter, $h(0) = \delta$.



Thus, the modified objective function gives

$$K_h(x, y) = \left[\sum_{m=1}^{M} \left(\frac{|S_m|^2}{2\, h(\sigma_m)} \right)^p (x, y) \right]^{1/p}$$

where p is an integer, typically $p = 1$ or $p = 2$.

Remarks:

1. Performance of K_h depends somewhat strongly on the choice of δ, in practice.
2. Modification of **a part** of distortion measure is not completely meaningful.
3. $\sigma = det(S)$ is not a quality measure. A small σ does not necessary mean a low-quality triangle; it is small if the triangle is small.
4. The transformation from the quality measure to the distortion measure is the function $y = 1/x$, which is a decreasing function for $x > 0$; however, the transformation is hardly applicable for both positive and negative sides of $[-1, 1]$.

A decreasing function

restart

$h := x \to \frac{1}{2}\left(x + \sqrt{x^2 + 4\,d^2}\right)$:

Define

$> g := x \to \frac{d}{h(x)}$

$$g := x \to \frac{d}{h(x)} \tag{4.1}$$

$$rationalize(g(x)) = \frac{1}{2}\,\frac{-x + \sqrt{x^2 + 4\,d^2}}{d}$$

$> g := x \to \frac{1}{2}\,\frac{-x + \sqrt{x^2 + 4\,d^2}}{d}$

$$g := x \to \frac{1}{2}\,\frac{-x + \sqrt{x^2 + 4\,d^2}}{d} \tag{4.2}$$

$d := 1 : g(x)$

$$-\frac{1}{2}\,x + \frac{1}{2}\sqrt{x^2 + 4} \tag{4.3}$$

$g1 := x \to -\frac{1}{2}\,x + \frac{1}{2}\sqrt{x^2 + 4}$:

$d := \frac{1}{5} : g(x)$

$$-\frac{5}{2}\,x + \frac{1}{2}\sqrt{25\,x^2 + 4} \tag{4.4}$$

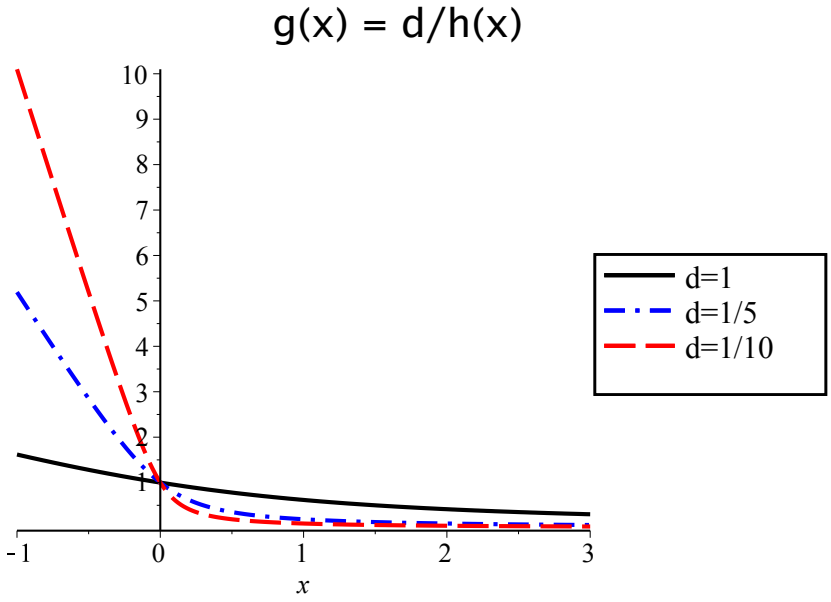
$g5 := x \to -\frac{5}{2}\,x + \frac{1}{2}\sqrt{25\,x^2 + 4}$:

$$d := \frac{1}{10} : g(x)$$

$$-5x + \sqrt{25x^2 + 1} \qquad \textbf{(4.5)}$$

$$g10 := x \to -5x + \sqrt{25x^2 + 1} :$$

```
plot([g1(x), g5(x), g10(x)], x = (-1) .. 3, thickness = [2, 2, 2], linestyle
    = [solid, dashdot, dash], color = [black, blue, red], title
    = "g(x) = d/h(x)", titlefont = ["HELVETICA", 15], legend = ["d=1",
     "d=1/5", "d=1/10"], legendstyle = [font = ["HELVETICA", 13],
     location = right]);
```



New objective function

$$K_g(x, y) = \left[\sum_{m=1}^{M} g\left(\frac{2\sigma_m}{|S_m|^2} \right)^p (x, y) \right]^{1/p}$$

where g is a positive, strictly decreasing function.

Remarks:

1. g is defined only on $[-1, 1]$
2. The graph of $y = g(x)$ must be convex up and $g'(1) \lesseqgtr 0$; in this case, tangled triangles can be more effectively untangled, and (near) ideal triangles are stabilized not to be altered.
3. For example, $g(x) = (x - 1)^2$

Your project will be related to refining the objective function K_g by (1) introducing the most effective decreasing function g and (2) applying Newton method with which the iterates would converge to the optimal point beginning from a given initial point.

Convex-Objective Mesh Optimization

```
restart
with(LinearAlgebra) : with(ArrayTools) : with(plots) : Digits := 10 :
a := 0 : b := 1 : c := 0 : d := 1 :
nx := 10 : ny := 10 :

g := x→(x − 1)^2 :   p := 1 :
perturb := 8.0 :
W := Matrix([[1, 1/2], [0, sqrt(3)/2]]) : WI := W^(-1) :
X := ⟨x, y⟩ :
itmax := 20 : tol := 0.01 :  ## For Newton's method
```

point array P := Array(0..nx,0..ny,1..2)

```
P := Array(0 ..nx, 0 ..ny, 1 ..2) :
hx := (b − a)/nx : hy := (d − c)/ny :
for j from 0 to ny do
   py := c + j·hy;
   for i from 0 to nx do
   px := a + i·hx;
   P[i, j, 1] := px;
   P[i, j, 2] := py;
  end do
end do:
```

Proc: A:=mesh2matrix(P)

```
mesh2matrix := proc(M)
   local m, n, d, i, j, mc, A;
   local LL, LR, UL, UR;
   m, n, d := Size(M);
   A := Matrix(8·(m − 1)·(n − 1), 2);
   mc := 0;
   for j to n − 1 do
      for i  to m − 1 do
         LL := M[i − 1, j − 1, 1 ..2]; LR := M[i, j − 1, 1 ..2];
         UL := M[i − 1, j, 1 ..2];      UR := M[i, j, 1 ..2];
        A[mc + 1, 1 ..2] := LL;  A[mc + 2, 1 ..2] := LR;
```

```
        A[mc + 3, 1 ..2] := UL; A[mc + 4, 1 ..2] := LL;
        A[mc + 5, 1 ..2] := LR; A[mc + 6, 1 ..2] := UR;
        A[mc + 7, 1 ..2] := UL; A[mc + 8, 1 ..2] := LR;
        mc := mc + 8;
      end do;
    end do;
    return A;
end proc:
```

Proc: meshplot(A)

```
meshplot := proc(A)
    local nT, i, T;
    nT := Size(A, 1) /4 :
    for i to nT do
        T_i := A[4 i − 3 ..4 i];
    end do;
    plot([seq(T_i, i = 1 ..nT)], color = black, thickness = 1);
end proc:
```

Proc: iT:=IsTangled(Pc,V)

```
IsTangled := proc(P0, V)
    local r, c, m, nT, iT, A;
    r, c := Dimension(V);
    nT := r − 1;
    iT := 0;
    for m from 1 to nT do
      A := ⟨Vector[column](V[m, 1 ..2]) − P0|Vector[column](V[m + 1, 1 ..2])
      − P0⟩;
      if Determinant(A) < 0 then
        iT := 1; break;
      end if;
    end do;
    return iT;
end proc:
```

Proc: nTT:=HowManyTangled(P)

```
HowManyTangled := proc(P)
    local i, nT, nTT, A, T;
    A := mesh2matrix(P);
    nT := Size(A, 1) /4 :
    nTT := 0;
    for i from 0 to nT − 1 do
       T := ⟨Vector[column](A[4 i + 2] − A[4 i + 1])|Vector[column](A[4 i + 3]
     − A[4 i + 1])⟩;
       if Determinant(T) ≤ 0 then
          nTT := nTT + 1;
       end if;
    end do;
    return nTT;
end proc:
```

Proc: V:=getVertices(P,i,j)

```
getVertices := proc(P, i, j)
    local V;
    V := Matrix(7, 2);
    V[1, 1 ..2] := P[i,      j − 1, 1 ..2];
    V[2, 1 ..2] := P[i + 1, j − 1, 1 ..2];
    V[3, 1 ..2] := P[i + 1, j,      1 ..2];
    V[4, 1 ..2] := P[i,      j + 1, 1 ..2];
    V[5, 1 ..2] := P[i − 1, j + 1, 1 ..2];
    V[6, 1 ..2] := P[i − 1, j,      1 ..2];
    V[7, 1 ..2] := V[1, 1 ..2];
    return V;
end proc:
```

Proc: K:=ObjectiveSD(V,X,WI,g,p)

```
ObjectiveSD := proc(V, X, WI, g, p)
    local U, r, c, m, nT, A, S, Snorm2, σ, q, K;
    r, c := Dimension(V);
    nT := r − 1;
    K := 0;
    for m from 1 to nT do
      A := ⟨Vector[column](V[m, 1 ..2]) -X|Vector[column](V[m + 1, 1 ..2]) -X⟩;
      S := A.WI;
```

$Snorm2 := Trace\left(S^{\%T}.S\right);$
$\sigma := Determinant(S);$
$q := simplify\left(\frac{2\,\sigma}{Snorm2}\right);$
$K := K + g(q)^{p};$
end do:
return $K^{1/p}$;
end proc:

Proc: K:=ObjectiveUT(V,X,WI,g,p)

$ObjectiveUT :=$ **proc**(V, X, WI, g, p)
local $r, c, m, nT, A, S, Snorm2, \sigma, q, K$;
local $S1, S2$;
$r, c := Dimension(V);$
$nT := r - 1;$
$K := 0;$
for m **from** 1 **to** nT **do**
$A := \langle Vector[column](V[m, 1..2]) - X | Vector[column](V[m+1, 1..2]) - X \rangle;$
$S := A.WI;$
$Snorm2 := Trace\left(S^{\%T}.S\right);$
$\sigma := Determinant(S);$
$S1 := Norm(S[1..2, 1], 2);$
$S2 := Norm(S[1..2, 2], 2);$
$\#\, q := simplify\left(\frac{2\,\sigma}{Snorm2}\right);$
$q := simplify\left(\frac{\sigma}{Snorm2} \cdot \left(S1^2 + S2^2\right)\right);$
$K := K + g(q)^{p};$
end do:
return $K^{1/p}$;
end proc:

Proc: K:=ObjectiveConvex(V,X,WI)

```
ObjectiveConvex := proc(V, X, WI, g, p)
  local r, c, m, nT, A, S, Snorm2, σ, q, K;
  r, c := Dimension(V);
  nT := r − 1;
  K := 0;
  for m from 1 to nT do
    A := ⟨Vector[column](V[m, 1 ..2]) - X|Vector[column](V[m + 1, 1 ..2]) - X⟩;
    S := A.WI;
    Snorm2 := Trace(S^%T.S);
    σ := Determinant(S);
    # q := simplify(2 σ / Snorm2);
    q := simplify(Snorm2 − 2 σ);
    K := K + q;
  end do:
  return K;
end proc:
```

Proc: Pc:=NewtonMeshOpt(Pc,K,itmax,tol)

```
NewtonMeshOpt := proc(P0, K, itmax, tol)
  local Pn, n, x0, y0, h, h0;
  local KC, KE, KW, KS, KN, KNE, KNW, KSE, KSW;
  local Kxx, Kxy, Kyy, J, G, H, Hnorm;
  h := 10^−3;
  #print(P, K, itmax, Tol);
  Pn := P0;
  for n from 1 to itmax do
    x0 := Pn[1]; y0 := Pn[2];
    KC := evalf(eval(K, [x = x0, y = y0]));
    KW := evalf(eval(K, [x = x0 − h, y = y0]));
    KE := evalf(eval(K, [x = x0 + h, y = y0]));
    KS := evalf(eval(K, [x = x0, y = y0 − h]));
    KN := evalf(eval(K, [x = x0, y = y0 + h]));
    KNE := evalf(eval(K, [x = x0 + h, y = y0 + h]));
    KNW := evalf(eval(K, [x = x0 − h, y = y0 + h]));
```

```
    KSE := evalf(eval(K, [x=x0+h, y=y0-h]));
    KSW := evalf(eval(K, [x=x0-h, y=y0-h]));
    Kxx := evalf((KE-2*KC+KW)/h^2);
    Kyy := evalf((KS-2*KC+KN)/h^2);
    Kxy := evalf((KNE+KSW-KNW-KSE)/(4*h^2));
    J := Matrix([[Kxx, Kxy], [Kxy, Kyy]]);
    G := <(KE-KW)/(2*h), (KN-KS)/(2*h)>;
    if Determinant(J)=0 then
      J := J+Matrix(2, shape=identity);
    end if;
    H := -J^(-1).G;
    Pn := Pn+H;
    Hnorm := Norm(H, 2);
    if ( Hnorm <= tol ) then
      #printf("    Newton Iteration: n=%d, |H|=%g\n", n, Hnorm);
      break;
    end if;
    h := min(h, Hnorm);
  end do;
  return Pn;
end proc:
```

Proc: P2:=MeshCopy(P1)

```
MeshCopy :=proc(P1)
  local m, n, d, i, j, P2;
  m, n, d := Size(P1);
  P2 := Array(0..(m-1), 0..(n-1), 1..d);
  for j from 0 to n-1 do
  for i from 0 to m-1 do
    P2[i, j, 1..d] := P1[i, j, 1..d];
  end do;
  end do;
  return P2;
end proc:
```

Proc: norm2:=MeshDistance(P1,P2)

```
MeshDistance :=proc(P1, P2)
```

```
    local m, n, d, i, j, d1, d2, norm2;
    m, n, d := Size(P1);
    norm2 := 0;
    for j from 0 to n - 1 do
    for i from 0 to m - 1 do
        d1 := P1[i, j, 1] - P2[i, j, 1];
        d2 := P1[i, j, 2] - P2[i, j, 2];
        norm2 := norm2 + evalf(d1·d1 + d2·d2);
    end do;
    end do;
    return sqrt(norm2/(m·n));
end proc:
```

Proc: norm8:=MeshDistance8(P1,P2)

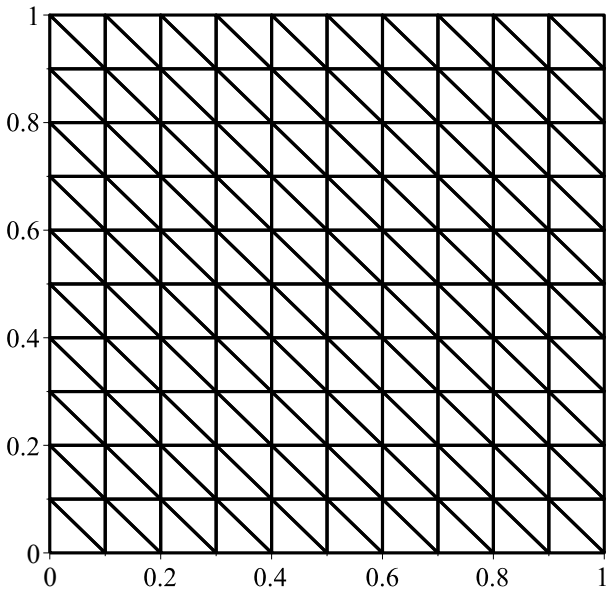
```
MeshDistance8 := proc(P1, P2)
    local m, n, d, i, j, d1, d2, norm8;
    m, n, d := Size(P1);
    norm8 := 0;
    for j from 0 to n - 1 do
    for i from 0 to m - 1 do
        d1 := P1[i, j, 1] - P2[i, j, 1];
        d2 := P1[i, j, 2] - P2[i, j, 2];
        norm8 := max(norm8, evalf(d1·d1 + d2·d2));
    end do;
    end do;
    return sqrt(norm8);
end proc:
```

Begin: Mesh Optimization and Plotting

##==========

$P0 := MeshCopy(P)$:

$meshplot(mesh2matrix(P0))$



perturb P randomly

$mrand := nx \cdot ny$:

$roll := rand(-mrand..mrand)$:

$scale := (perturb/2.) \cdot (hx/mrand)$:

for j **from** 1 **to** $ny - 1$ **do**

for i **from** 1 **to** $nx - 1$ **do**

$P[i, j, 1] := P[i, j, 1] + roll() \cdot scale;$

$P[i, j, 2] := P[i, j, 2] + roll() \cdot scale;$

end do;

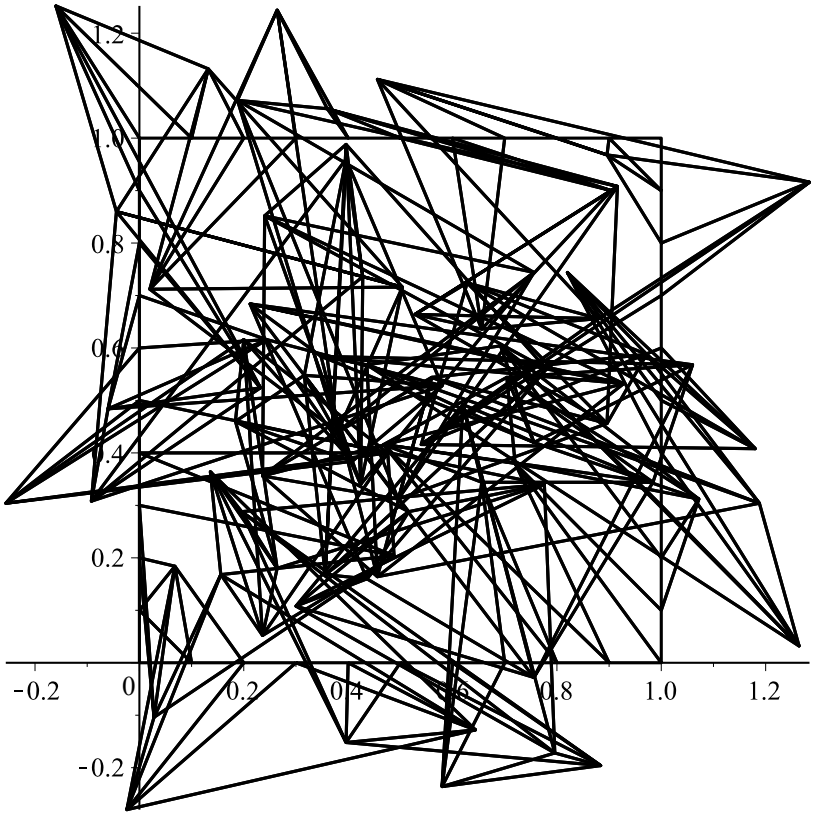
end do:

$PP := MeshCopy(P)$:

$printf("\quad \text{Mesh L2-distance from P0} = \%g\backslash n", MeshDistance(P0, PP))$;

Mesh L2-distance from P0 = 0.267679

$meshplot(mesh2matrix(PP))$



Untangling Iterations

```
nTT := HowManyTangled(P) :
printf(" The number of tangled triangles = %d\n", nTT);
```

The number of tangled triangles = 97

```
iter := 0 :
while (nTT > 0) do
  for j from 1 to ny − 1 do
  for i from 1 to nx − 1 do
    Pc := Vector[column](P[i, j, 1 ..2]);
    V := getVertices(P, i, j) :
    K := ObjectiveConvex(V, X, WI);
    P[i, j, 1 ..2] := NewtonMeshOpt(Pc, K, itmax, tol);
  end do;
```

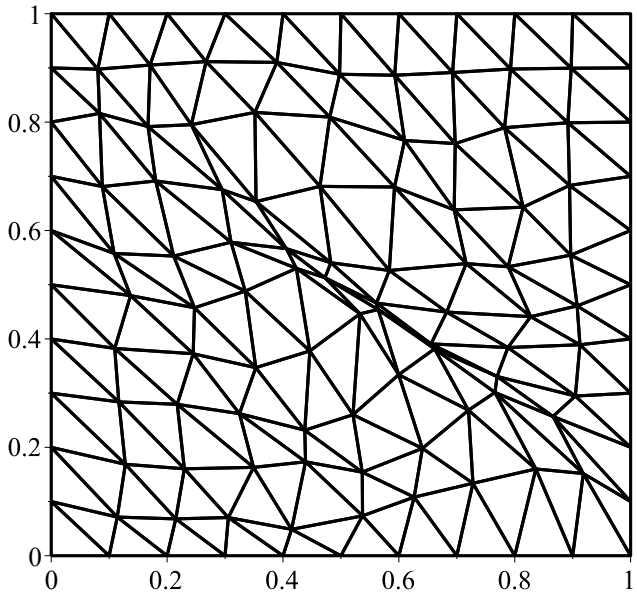
```
    end do;
    iter := iter + 1;
    nTT := HowManyTangled(P);
    printf("    iter=%d,  nTT=%d\n", iter, nTT);
end do:
```

```
    iter=1,  nTT=31
    iter=2,  nTT=0
```

```
PUT := MeshCopy(P) :
printf("    Mesh L2-distance from P0 = %g\n", MeshDistance(P0, PUT) );
```

```
    Mesh L2-distance from P0 = 0.0382116
```

```
meshplot(mesh2matrix(PUT) );
```



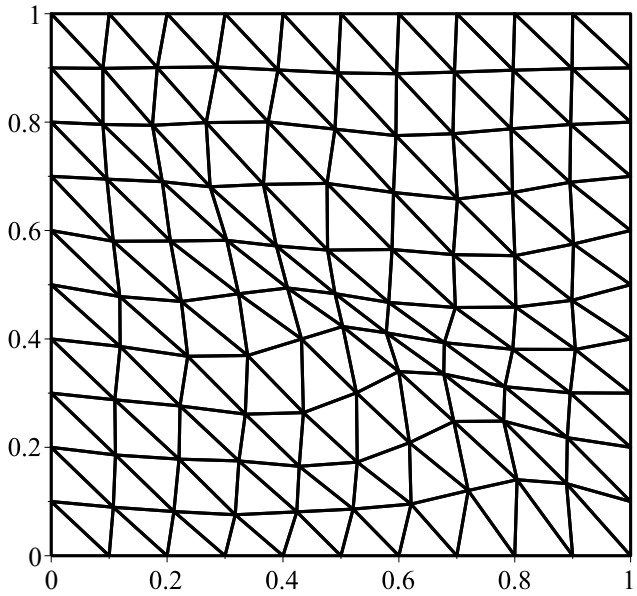
Xtra one Untangling Iteration: The algorithm may be adjusted so as to perform only if the minimum of quality measure is less than a prescribed level.

```
for j from 1 to ny − 1 do
for i from 1 to nx − 1 do
    Pc := Vector[column](P[i, j, 1 ..2]);
```

```
    V := getVertices(P, i, j) :
    K := ObjectiveConvex(V, X, WI);
    P[i, j, 1 ..2] := NewtonMeshOpt(Pc, K, itmax, tol);
  end do;
  end do:
PUT2 := MeshCopy(P) :
printf(" Mesh L2-distance from P0 = %g\n", MeshDistance(P0, PUT2));
```

Mesh L2-distance from P0 = 0.0241698

```
meshplot(mesh2matrix(PUT2));
```



Finally, minization with Quality Measure

```
tolQM := 0.003 : norm2 := 1.0 :
iter := 0 :
while (norm2 > tolQM) do
  Ps := MeshCopy(P);
  for j from 1 to ny − 1 do
  for i from 1 to nx − 1 do
    Pc := Vector[column](P[i, j, 1 ..2]);
    V := getVertices(P, i, j) :
```

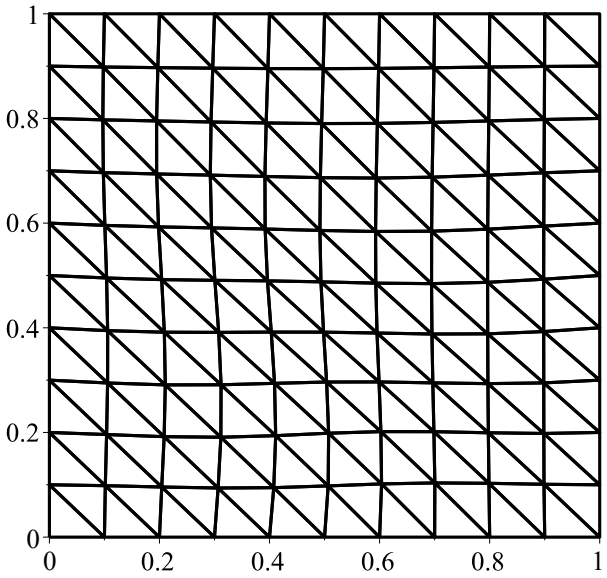
```
    K := ObjectiveConvex(V, X, WI);
    P[i, j, 1 ..2] := NewtonMeshOpt(Pc, K, itmax, tol);
  end do;
  end do;
  norm2 := MeshDistance(P, Ps);
  iter := iter + 1;
  printf("    iter=%d, norm2=%g\n", iter, norm2);
end do:
```

```
  iter=1, norm2=0.00913573
  iter=2, norm2=0.00531045
  iter=3, norm2=0.00344744
  iter=4, norm2=0.0023531
```

```
PQ := MeshCopy(P) :
printf("    Mesh L2-distance from P0 = %g\n", MeshDistance(P0, PQ) );
```

```
  Mesh L2-distance from P0 = 0.00773019
```

```
meshplot(mesh2matrix(PQ) );
```



Homework:
10. Numerical Solutions of Nonlinear Systems of Equations

#1. The nonlinear system

$$x^2 - 10x + y^2 + 8 = 0$$
$$xy^2 + x - 10y + 8 = 0$$

can be transformed into the fixed-point problem

$$x = g_1(x, y) = \frac{x^2 + y^2 + 8}{10}, \quad y = g_2(x, y) = \frac{xy^2 + x + 8}{10}$$

a. Prove that $G = \left(g_1, g_2\right)^T$ mapping from $D \subset \mathbb{R}^2$ to $\mathbb{R}^2$ has a unique fixed point in

$$D = [0, 1.5]^2$$

b. Apply Fixed-Point iteration to approximate the fixed point to within 10^{-5} in ℓ_∞ norm, beginning from $(x, y)^0 = (1, 1)$.

#2. The nonlinear system

$$5x_1^2 - x_2^2 = 0, \qquad x_2 - \frac{\sin\left(x_1\right) + \cos\left(x_2\right)}{4} = 0$$

has a solution near $(1/4, 1/4)^T$.

a. Find a function G and a set $D \subset \mathbb{R}^2$ such that $G : D \to \mathbb{R}^2$ and G has a unique fixed point in D.

b. Apply Fixed-Point iteration to approximate the fixed point to within 10^{-5} in ℓ_∞ norm.

#3. This problem deals with how to make an accurate approximation in order for the Newton's method to converge to a solution in a specific region. Consider the system of equations

$$x_1^2 - x_2^2 + 2x_2 = 0$$
$$2x_1 + x_2^2 - 6 = 0$$

a. Use a plotting software to plot the equations, e.g., *implicitplot* in Maple.
b. From the plot, find initial approximations of all (maximum four) solutions.
c. Use Newton's method to update the approximations within to 10^{-8} accuracy.

#4. Starting with $(1, 1, 1)^T$, carry out four iterations of Quasi-Newton method to approximate a root of the nonlinear system

$$\begin{cases} xy = z^2 + 1 \\ xyz + y^2 = x^2 + 2 \\ e^x + z = e^y + 3 \end{cases}$$

(For the first iteration, use the Newton's method.)

11. Boundary-Value Problems of One Variable

In This Chapter:

Topics	Applications/Properties
Shooting Method	
Existence and uniqueness of the solution of BVPs	
Linear shooting method	
Finite Difference Methods (FDM)	
Central schemes	
Algebraic systems	
Finite Element Methods (FEM)	Flexibility not only on geometry but on approximation
Variational formulation	Weak form
Rayleigh-Ritz method	Classic, minimization principle
Galerkin method	Weighted residual approaches
Petrov-Galerkin method	
Least-squares method	
Collocation method	
FDM for Non-constant Diffusion	

The two-point BVPs in this chapter involve a second-order differential equation of the form

$$y'' = f(x, y, y'), \quad a \le x \le b$$
$$y(a) = \alpha, y(b) = \beta \qquad (1)$$

11.1. Linear Shooting Method

Existence and Uniqueness Theorem

Let

$$D := \{(x, y, y') : a \le x \le b, \text{ with } -\infty < y < \infty, -\infty < y' < \infty\}.$$

For the BVP of the form in (1), If the functions $f, f_y, f_{y'}$ are continuous on D and satisfy

(i) $f_y(x, y, y') > 0$, for all $(x, y, y') \in D$

(ii) there is a constant M such that

$$|f_{y'}(x, y, y')| \le M, \text{ or all } (x, y, y') \in D$$

then the BVP has a unique solution.

Example: Show that the following BVP has a unique solution.

$$y'' + e^{-xy} + \sin(y') = 0, \quad 1 \le x \le 2, \text{ with } y(1) = y(2) = 0.$$

Solution

We have

$$f(x, y, y') = -e^{-xy} - \sin(y'),$$

which is clearly continuous. For $1 \le x \le 2$,

$$f_y = x\, e^{-xy} > 0$$

$$|f_{y'}(x, y, y')| = |-\cos(y')| \le 1$$

So the problem has a unique solution.

Linear Boundary-Value Problems

The differential equation $y'' = f(x, y, y')$ is **linear** if f can be expressed as a linear combination of y and y', that is,

$$f(x, y, y') = p(x)\, y' + q(x)\, y + r(x)$$

for some functions p, q, r.

Corollary

Suppose the linear BVP

$$y'' = p(x)\, y' + q(x)\, y + r(x), \quad a \le x \le b$$
$$y(a) = \alpha, \; y(b) = \beta \qquad (3.1)$$

satisfies

(i) $p(x)$, $q(x)$, and $r(x)$ are continuous on $[a, b]$

(ii) $q(x) > 0$ on $[a, b]$

Then the linear BVP has a unique solution.

The Linear Shooting Method:

To approximate the unique solution to the linear BVP

$$y'' = p(x)\,y' + q(x)\,y + r(x), \quad a \le x \le b$$
$$y(a) = \alpha,\ y(b) = \beta \tag{2}$$

we first solve the problem twice, with two different *initial conditions*, obtaining solutions y_1 and y_2, say,

$$\begin{cases} y_1(a) = \alpha, \quad y_1'(a) = z_1 \\ y_2(a) = \alpha, \quad y_2'(a) = z_2 \end{cases}$$

Then we form a linear combination of y_1 and y_2:

$$y(x) = \lambda y_1(x) + (1 - \lambda)\, y_2(x) \tag{3}$$

where λ is a parameter. We can easily verify that $y(a) = \alpha$.

Thus the remained requirement is to select λ for $y(x)$ to satisfy $y(b) = \beta$. That is,

$$y(b) = \lambda y_1(b) + (1 - \lambda)\, y_2(b) = \beta$$

and therefore

$$\lambda = \frac{\beta - y_2(b)}{y_1(b) - y_2(b)} \tag{4}$$

provided that $y_1(b) \neq y_2(b)$.

Note that z_1 and z_2 can be chosen arbitrarily except $z_1 \neq z_2$; however, the most common choice is $z_1 = 0$ and $z_2 = 1$.

Claim: If the linear BVP (2) has a solution y, then either $y = y_1$ or $y_1(b) \neq y_2(b)$. For the later case, the solution y is a linear combination as in (3) with (4).

Summary of the linear shooting method

1. Solve the two BVPs for $y_1(x)$ and $y_2(x)$:

$$\begin{cases} y'' = f(x, y, y') \\ y(a) = \alpha,\ y'(a) = 0 \end{cases} \qquad \begin{cases} y'' = f(x, y, y') \\ y(a) = \alpha,\ y'(a) = 1 \end{cases}$$

   where $f(x, y, y') = p(x)y' + q(x)y + r(x)$.

2. Compute

$$\lambda = \frac{\beta - y_2(b)}{y_1(b) - y_2(b)}$$

3. Get the solution of the linear BVP (2) as a linear combination of y_1 and y_2:

$$y(x) = \lambda y_1(x) + (1 - \lambda)\, y_2(x)$$

Each of the problems in the first step can be solved by applying an ODE solver for systems. For example, the **second problem** equivalently reads as a system of differential equations:

$$\begin{cases} y' = u, & y(a) = \alpha \\ u' = p\,u + q\,y + r, & u(a) = 1 \end{cases}$$

which can be solved efficiently by a fourth-order Runge-Kutta (RK4) method.

Let's see a brief review for ODE solvers.

Numerical Solutions of Ordinary Differential Equations (Review)
(See Sections 5.2 and 5.3 for details.)

Consider the initial-value problem (IVP):

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases} \qquad \text{(IVP)}$$

For the problem, approximations to y will be generated at various points, called **mesh points**, in the interval $[x_0, b]$. Let the mesh points be equally spaced:

$$x_i = x_0 + i\,h, \quad h = \frac{(b - x_0)}{N} \qquad (0 \le i \le N)$$

We denote an approximate value of $y(x_n)$ by y_n, that is,

$$y_n \approx y(x_n)$$

Then, common numerical methods for the numerical solution of (IVP) compute the approximate solution at a mesh point at a time, moving forward, estimating an accurate average of $f(x, y)$ and/or utilizing a few of solution values at previous mesh points. Thus such procedures are **step-by-step** methods.

Euler's Method

Let us try to find an approximation of $y(x_1)$, marching through the first subinterval $[x_0, x_1]$. Consider the Taylor series expansion,

$$y(x_1) = y(x_0 + h) = y(x_0) + y'(x_0)\,h + \frac{y''(x_0)}{2!}h^2 + \cdots$$

$$= y(x_0) + y'(x_0)\,h + \frac{y''(\xi)}{2!}h^2.$$

Then, utilizing $y(x_0) = y_0$ and $y'(x_0) = f(x_0, y_0)$, the value $y(x_1)$ can be approximated by

$$y_1 = y_0 + h f(x_0, y_0) \qquad (4.1)$$

Such an idea can be applied recursively for the computation of solution on later subintervals. Indeed, since

$$y(x_2) = y(x_1 + h) = y(x_1) + y'(x_1)\, h + \frac{y''(\xi)}{2!} h^2,$$

by replacing $y(x_1)$ and $y'(x_1)$ respectively with y_1 and $f(x_1, y_1)$, we obtain

$$y_2 = y_1 + h f(x_1, y_1) \tag{4.2}$$

which is an approximation of $y(x_2)$.

Summarizing the above, the **Euler's method** solving the first-order IVP is formulated as

$$y_{n+1} = y_n + h f(x_n, y_n),\ 0 \le n \tag{4.3}$$

Higher-Order Taylor Methods

If we expand the solution $y(x)$, in terms of its mth-order Taylor polynomial about x_n and evaluated at x_{n+1}, we obtain

$$\begin{aligned} y(x_{n+1}) &= y(x_n) + h\, y'(x_n) + \frac{h^2}{2!} y''(x_n) + \cdots + \frac{h^m}{m!} y^{(m)}(x_n) + \cdots \\ &= y(x_n) + h\, y'(x_n) + \frac{h^2}{2!} y''(x_n) + \cdots + \frac{h^m}{m!} y^{(m)}(x_n) \\ &\quad + \frac{h^{m+1}}{(m+1)!} y^{(m+1)}(\xi) \end{aligned}$$

Successive differentiation of the solution, $y(x)$, gives

$$y'(x) = f(x, y(x)),\ \ y''(x) = f'(x, y(x)),\ \text{ and generally,}$$
$$y^{(k)}(x) = f^{(k-1)}(x, y(x)).$$

Thus we have

$$y(x_{n+1}) = y(x_n) + h f(x_n, y(x_n)) + \frac{h^2}{2!} f'(x_n, y(x_n)) + \cdots$$

$$+\frac{h^m}{k!}f^{(m-1)}\left(x_n, y(x_n)\right)+\frac{h^{m+1}}{(m+1)!}f^{(m)}\left(\xi_n, y(\xi_n)\right)$$

The **Taylor method of order *m*** corresponding the above equation is obtained by deleting the remainder term involving ξ_n.

$$y_{n+1}=y_n+h\,T_m(x_n, y_n) \tag{5.1}$$

where

$$T_m(x_n, y_n)=f(x_n, y_n)+\frac{h}{2!}f'(x_n, y_n)+\cdots+\frac{h^{m-1}}{m!}f^{(m-1)}(x_n, y_n)$$

Runge-Kutta Methods

Runge-Kutta methods have high-order local truncation error of the Taylor methods but eliminate the need to compute and evaluate the derivatives of $f(x, y)$. That is, the **Runge-Kutta Methods** are formulated, incorporating a weighted average of slopes, as follows:

$$y_{n+1}=y_n+h\left(w_1K_1+w_2K_2+\cdots+w_mK_m\right),$$

where

$w_1+w_2+\ldots+w_m=1$

K_j are (recursive) evaluations of the slope $f(x, y)$

Nee to determine w_j and other parameters to satisfy

$$w_1K_1+w_2K_2+\ldots+w_mK_m\approx T_m(x_n, y_n)+\mathrm{O}(h^m)$$

Second-order Runge-Kutta method (RK2; Heun's method)

$$y_{n+1} = y_n + \frac{1}{2} h \left(K_1 + K_2 \right) \tag{6.1.1}$$

where

$$K_1 = f\left(x_n, y_n \right)$$

$$K_2 = f\left(x_n + h, y_n + h K_1 \right)$$

Modified Euler method:

$$y_{n+1} = y_n + h f\left(x_n + \frac{1}{2} h, y_n + \frac{1}{2} h f\left(x_n, y_n \right) \right) \tag{6.2.1}$$

Fourth-order Runge-Kutta method (RK4)

The most commonly used set of parameter values yields

$$y_{n+1} = y_n + \frac{1}{6} h \left(K_1 + 2 K_2 + 2 K_3 + K_4 \right) \tag{6.3.1}$$

where

$$K_1 = f\left(x_n, y_n \right)$$

$$K_2 = f\left(x_n + \frac{1}{2} h, y_n + \frac{1}{2} h K_1 \right)$$

$$K_3 = f\left(x_n + \frac{1}{2} h, y_n + \frac{1}{2} h K_2 \right)$$

$$K_4 = f\left(x_n + h, y_n + h K_3 \right)$$

RK4SYSTEM
##----------

Ex) $y''-2y'+2y=e^{2x}\cdot\sin(x),\quad 0\le x\le 1,\ y(0)=-0.4, y'(0)=-0.6$

$Digits := 15$:

```
> RK4SYSTEM := proc(a,b,nt,X,F,x0,xn)
      local h,hh,t,m,n,j,w,K1,K2,K3,K4;
      #### initial setting
      with(LinearAlgebra):
      m := Dimension(Vector(F));
      w  :=Vector(m);
      K1:=Vector(m);
      K2:=Vector(m);
      K3:=Vector(m);
      K4:=Vector(m);
      h:=(b-a)/nt;  hh:=h/2;
      t :=a;
      w:=x0;
      for j from 1 by 1 to m do
          xn[0,j]:=x0[j];
      end do;
      #### RK4 marching
      for n from 1 by 1 to nt do
         K1:=Vector(eval(F, [x=t,seq(X[i+1] = xn[n-1,i], i = 1 .. m)]));
         K2:=Vector(eval(F, [x=t+hh,seq(X[i+1] = xn[n-1,i]+hh*K1[i], i = 1 .. m)]));
         K3:=Vector(eval(F, [x=t+hh,seq(X[i+1] = xn[n-1,i]+hh*K2[i], i = 1 .. m)]));
         t:=t+h;
         K4:=Vector(eval(F, [x=t,seq(X[i+1] = xn[n-1,i]+h*K3[i], i = 1 .. m)]));
         w:=w+(h/6.)*(K1+2*K2+2*K3+K4);
         for j from 1 by 1 to m do
             xn[n,j]:=evalf(w[j]);
         end do
      end do
end proc:
```

$m := 2$:

$F := \left[yp, e^{2x}\cdot\sin(x) - 2y + 2yp\right]$:

$X := [x, y, yp]$:

$X0 := \langle -0.4, -0.6\rangle$:

$a := 0$: $b := 1$:

$nt := 10$:

$Xn := Array(0..nt, 1..m)$:

$RK4SYSTEM(a, b, nt, X, F, X0, Xn)$:

$DE := diff(y(x), x, x) - 2\,diff(y(x), x) + 2\,y(x) = e^{2x}\cdot\sin(x)$:
$IC := y(0) = -0.4, D(y)(0) = -0.6$:

$dsolve(\{DE, IC\}, y(x)) = y(x) = \frac{1}{5}\,(\sin(x) - 2\cos(x))\,e^{2x}$

$ey := x \to \frac{1}{5}\,e^{2x}\,(\sin(x) - 2\cos(x))$:

$diff(ey(x), x) = \frac{1}{5}\,(2\sin(x) + \cos(x))\,e^{2x} + \frac{2}{5}\,(\sin(x) - 2\cos(x))\,e^{2x}$

$eyp := x \to \frac{2}{5}\,e^{2x}\,(\sin(x) - 2\cos(x)) + \frac{1}{5}\,e^{2x}\,(2\sin(x) + \cos(x))$:

```
h := (b - a) / nt :
for n from 0 to nt do
    xp := a + n·h;
    if (n = 0) then
    printf(" n y_n y(x_n) y'_n y'(x_n) err (y) err(y')\n");
    end if;
    printf(" %d %12.8f %12.8f %12.8f %12.8f %.3g %.3g\n", n, Xn[n, 1], ey(xp), Xn[n, 2], eyp(xp), abs(Xn[n, 1]-ey(xp)), abs(Xn[n, 2]-eyp(xp)));
end do:
```

n	y_n	y(x_n)	y'_n	y'(x_n)	err(y)	err(y')
0	-0.40000000	-0.40000000	-0.60000000	-0.60000000	0	0
1	-0.46173334	-0.46173297	-0.63163124	-0.63163105	3.72e-07	1.92e-07
2	-0.52555988	-0.52555905	-0.64014895	-0.64014866	8.36e-07	2.83e-07
3	-0.58860144	-0.58860005	-0.61366381	-0.61366361	1.39e-06	1.99e-07
4	-0.64661231	-0.64661028	-0.53658203	-0.53658220	2.02e-06	1.68e-07
5	-0.69356666	-0.69356395	-0.38873810	-0.38873905	2.71e-06	9.57e-07
6	-0.72115190	-0.72114849	-0.14438087	-0.14438322	3.41e-06	2.35e-06
7	-0.71815295	-0.71814890	0.22899702	0.22899243	4.06e-06	4.59e-06
8	-0.66971133	-0.66970677	0.77199180	0.77198383	4.55e-06	7.97e-06
9	-0.55644290	-0.55643814	1.53478148	1.53476862	4.76e-06	1.29e-05
10	-0.35339886	-0.35339436	2.57876634	2.57874662	4.50e-06	1.97e-05

Example: Use the shooting method to approximate the solution of

$$y'' = -\frac{2}{x}y' + \frac{2}{x^2}y - 3x^2, \quad 1 \le x \le 2,$$
$$y(1) = 0, \; y(2) = 2.$$

Solution

$m := 2:$
$F := \left[yp, -\frac{2\,yp}{x} + \frac{2\,y}{x^2} - 3x^2\right]:$
$X := [x, y, yp]:$
$a := 1: \; b := 2:$
$al := 0: \; be := 2:$
$N := 5:$
$Y := Array(0..2\cdot N, 1..2):$

for k **to** 2 **do**
  $nt := 2^{k-1}\cdot N:$
  $Y1 := Array(0..nt, 1..m);$
  $Y2 := Array(0..nt, 1..m);$
  $X0 := \langle al, 0\rangle;$
  $RK4SYSTEM(a, b, nt, X, F, X0, Y1):$
  $X0 := \langle al, 1\rangle;$
  $RK4SYSTEM(a, b, nt, X, F, X0, Y2):$
  $lam := (be - Y2[nt, 1])/(Y1[nt, 1] - Y2[nt, 1]);$
  for i **from** 0 **to** nt **do**
    $Y[i, k] := lam\cdot Y1[i, 1] + (1 - lam)\cdot Y2[i, 1];$
  end do;
end do:

The exact solution & Error analysis

$DE := diff(y(x), x, x) = -\frac{2}{x}\,diff(y(x), x) + \frac{2}{x^2}y(x) - 3x^2:$

$BC := y(1) = 0, y(2) = 2:$

$dsolve(\{DE, BC\}, y(x)) =$

$$y(x) = -\frac{52}{21\,x^2} - \frac{1}{6}\,x^4 + \frac{37}{14}\,x$$

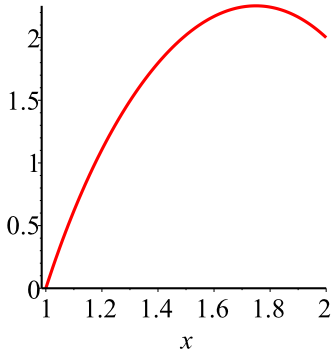
$$ey := x \to -\frac{52}{21\,x^2} - \frac{1}{6}\,x^4 + \frac{37}{14}\,x :$$

```
for i from 0 to N do
    if (i=0) then
       printf("                    h=1/5              error              h=1/10
    error\n");
    end if;
    xp := a + i·(b − a)/N;
    printf("x=%4.2f      %15.10f  %10.3g    %15.10f  %10.3g\n", xp, Y[i,
     1], abs(Y[i, 1] − ey(xp)), Y[2·i, 2], abs(Y[2·i, 2] − ey(xp)));
end do:
```

	h=1/5	error	h=1/10	error
x=1.00	0.0000000000	0	0.0000000000	0
x=1.20	1.1056335076	0.000618	1.1062189010	3.3e-005
x=1.40	1.7958326514	0.000538	1.7963419934	2.89e-005
x=1.60	2.1686944799	0.000348	2.1690241106	1.87e-005
x=1.80	2.2431240407	0.000162	2.2432777718	8.77e-006
x=2.00	2.0000000000	0	2.0000000000	0

At $x = 1.2$, the error ratio= $\frac{0.000618}{3.3\text{e-}005} = 18.72727273$

$plot(ey(x),\ x = a\ ..b)$



Empty Page

11.2. Finite Difference Methods

A linear BVP has the form

$$y'' = p(x)\,y' + q(x)\,y + r(x), \quad a \le x \le b$$
$$y(a) = \alpha, \quad y(b) = \beta$$

In order to develop finite difference methods (FDM) for the solution of the problem in more general environments, we in this section consider differential equations of the following form

$$-y'' + p(x)\,y' + q(x)\,y = f(x), \quad a \le x \le b$$
$$y(a) = \alpha, \quad \frac{d}{dx}\,y(x = b) = \beta \qquad (1)$$

Let, for some $n > 0$,

$$h = \frac{(b-a)}{n}, \quad x_i = a + i \cdot h \quad (0 \le i \le n)$$

For a simple presentation, define $g_i = g(x_i)$ for $g = y, p, q, f$; for example,

$$y_i = y(x_i)$$

The objective with finite difference methods is to approximate $\{y_i\}_{i=1}^{n}$, the solution at discrete nodal points.

We begin with the Taylor's Theorem as follows.

Taylor's Theorem:

Suppose $f \in C^n[a, b]$ and $f^{(n+1)}$ exists on (a, b). Then, for every $x,\ x + h \in [a, b]$,

$$f(x+h) = \sum_{k=0}^{n} \frac{f^{(k)}(x)}{k!} h^k + R_n(h),$$

where, for some ξ between x and $x + h$,

$$R_n(h) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \cdot h^{n+1}.$$

In detail,

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \cdots + \frac{f^{(n)}(x)}{n!}h^n + R_n(h)$$

Expanding y in a fourth Taylor polynomial about $x = x_i$ evaluated at $x_{i+1} = x_i + h$ and $x_{i-1} = x_i - h$, we have

$$(a)\ y_{i+1} = y_i + h\,y'(x_i) + \frac{h^2}{2!}y''(x_i) + \frac{h^3}{3!}y'''(x_i) + \frac{h^4}{4!}y^{(4)}(x_i) + \frac{1}{120}h^5 D^{(5)}(y)(x_i) + \frac{1}{720}h^6 D^{(6)}(y)(\xi_{i1}) \qquad (2)$$

$$(b)\ y_{i-1} = y_i - h\,y'(x_i) + \frac{h^2}{2!}y''(x_i) - \frac{h^3}{3!}y'''(x_i) + \frac{h^4}{4!}y^{(4)}(x_i) - \frac{1}{120}h^5 D^{(5)}(y)(x_i) + \frac{1}{720}h^6 D^{(6)}(y)(\xi_{i0})$$

Central FD Schemes:

(2)(a)+(2)(b):

$$y_{i+1} + y_{i-1} = 2\,y_i + 2\,\frac{h^2}{2!}y''(x_i) + 2\,\frac{h^4}{4!}y^{(4)}(x_i) + O(h^6)$$

Solving the above for $y''(x_i)$ reads

$$y''(x_i) = \frac{y_{i-1} - 2\,y_i + y_{i+1}}{h^2} - \frac{h^2}{12}y^{(4)}(x_i) + O(h^4)$$

(2)(a)−(2)(b):

$$y_{i+1} - y_{i-1} = 2\,h\,y'(x_i) + 2\,\frac{h^3}{3!}y'''(x_i) + O(h^5)$$

Thus the central FD formula for $y'(x_i)$ is obtained as

$$y'(x_i) = \frac{y_{i+1} - y_{i-1}}{2\,h} - \frac{h^2}{6}y'''(x_i) + O(h^4)$$

Use of these central FD schemes in the first equation of Equation (1) at $x = x_i$ results in the equation

$$\frac{-y_{i-1} + 2y_i - y_{i+1}}{h^2} + p_i\left(\frac{1}{2}\frac{y_{i+1} - y_{i-1}}{h}\right) + q_i y_i = f_i + K_2 h^2 + \mathrm{O}\left(h^4\right) \tag{3}$$

where

$$K_2 = \frac{1}{12}\left(y^{(4)}(x_i) - 2p_i y'''(x_i)\right)$$

Due to the second equation in Equation (1), which is a boundary condition at $x = a = x_0$, we may assign y_0 as

$$y_0 = \alpha \tag{4}$$

The last equation in Equation (1) can utilize the above central FD formula for $y'(b = x_n)$:

$$\frac{1}{2}\frac{y_{n+1} - y_{n-1}}{h} = \beta + \frac{1}{6}h^2 \mathrm{D}^{(3)}(y)(x_n) + \mathrm{O}\left(h^4\right) \tag{5}$$

Second-Order FDM:

An FDM for the model (1) with truncation error of order $O(h^2)$ follows from Equations (3), (4), and (5), ignoring the terms involving h^2 and higher-orders of h. Let u_i denote the second-order approximation of y_i. Then, the **second-order FDM** for (1) reads

$$u_0 = \alpha$$

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} + p_i\left(\frac{1}{2}\,\frac{u_{i+1} - u_{i-1}}{h}\right) + q_i u_i = f_i \quad (1 \le i \le n)$$

$$u_{n+1} = u_{n-1} + 2h\beta \qquad (6)$$

Note that when $i = n$, the second equation of (6) reads

$$\frac{-u_{n-1} + 2u_n - u_{n+1}}{h^2} + p_n\left(\frac{1}{2}\,\frac{u_{n+1} - u_{n-1}}{h}\right) + q_n u_n = f_n$$

where the quantity u_{n+1} is a *ghost value* which can be eliminated utilizing the last equation. Such a treatment of the Neumann boundary condition is called the **method of ghost grids**.

That is, Equation (6) is rewritten as

$$\left(2 + h^2 q_1\right) u_1 - \left(1 - \frac{h}{2} p_1\right) u_2 = h^2 f_1 + \left(1 + \frac{h}{2} p_1\right)\alpha$$

$$-\left(1 + \frac{h}{2} p_i\right) u_{i-1} + \left(2 + h^2 q_i\right) u_i - \left(1 - \frac{h}{2} p_i\right) u_{i+1} = h^2 f_i \quad (2 \le i \le n-1)$$

$$-2u_{n-1} + \left(2 + h^2 q_n\right) u_n = h^2 f_n + 2\left(1 - \frac{1}{2} h p_n\right) h\beta \qquad (7)$$

Then, the resulting system of equations is expressed in the following $n \times n$ tridiagonal matrix form

$$A\,u = b \tag{8}$$

where

$$A = \begin{bmatrix} 2+h^2 q_1 & -\left(1-\frac{h}{2}p_1\right) & 0 & \cdots & 0 \\ -\left(1+\frac{h}{2}p_2\right) & 2+h^2 q_2 & -\left(1-\frac{h}{2}p_2\right) & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\left(1+\frac{h}{2}p_{n-1}\right) & 2+h^2 q_{n-1} & -\left(1-\frac{h}{2}p_{n-1}\right) \\ 0 & \cdots & 0 & -2 & 2+h^2 q_n \end{bmatrix}$$

and

$$u := \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix}, \qquad b = \begin{bmatrix} h^2 f_1 \\ h^2 f_1 \\ \vdots \\ h^2 f_{n-1} \\ h^2 f_n \end{bmatrix} + \begin{bmatrix} \left(1+\frac{h}{2}p_1\right)\alpha \\ 0 \\ \vdots \\ 0 \\ 2\left(1-\frac{h}{2}p_n\right)h\,\beta \end{bmatrix}$$

Note that when **Dirichlet data** are assigned at both end points, the algebraic system can be organized in the $(n-1)$-dimensional space. However, it can also be formulated with an $(n+1)$-dimensional system.

Solvability:

Theorem: Suppose that p, q, and f are continuous on $[a, b]$. If

$$0 \le q(x) \tag{3.1}$$

on $[a, b]$, then the linear system (8) has unique solution provided that

$$\frac{1}{2} h L < 1 \tag{3.2}$$

where $L = \max_{a \le x \le b} |p(x)|$.

Proof:

When (3.2) is satisfied, the quantities in the parentheses in the matrix A is all positive, which implies that A is diagonally dominant, i.e.,

$$a_{ii} \ge \sum_{j=i, j \ne i}^{n} |a_{ij}| =: \Lambda_i \quad \text{for all } 1 \le i \le 1.$$

Indeed,

$$0 < \Lambda_1 = 1 - \frac{h}{2} p_1 < 2 \le 2 + h^2 q_1 = a_{11}$$

$$\Lambda_i = 2 \le 2 + h^2 q_i = a_{ii} \quad (2 \le i \le n)$$

Since A is clearly irreducible, it is *irreducibly diagonally dominant*. Thus, A is nonsingular and therefore the linear system has a unique solution. Because all the diagonal entries of A are in addition positive real, real parts of eigenvalues of A are all positive.

Example: Use the second-order FDM to approximate the solution of

$$y'' = \frac{1}{x}\,y' + \frac{3}{x^2}\,y - 4x^2, \quad 1 \le x \le 2,$$
$$y(1) = 0, \; y'(2) = -2$$

with $h = 1/5$ and $h = 1/10$. (Note that $q(x) = 3/x^2 > 0$ and $L = \max |p(x)| = 1$ and therefore $h\,L/2 \ll 1$.)

Solution:

The differential equation can be written as

$$-y'' + \frac{1}{x}\,y' + \frac{3}{x^2}\,y = 4x^2$$

```
a := 1 : b := 2 :
al := 0 : be := -2 :
p := x -> 1/x : p(x) :
q := x -> 3/x^2 : q(x) :
f := x -> 4*x^2 : f(x) :
u := Array(1..2) :

for it from 1 to 2 do
    n := 2^(it-1)*5;
    h := (b-a)/n :
    A := Matrix(n, n);
    bb := Vector(n) :

    k := 1 :
        xk := a + k*h;
        A[k, k] := 2 + h^2*q(xk);
        A[k, k+1] := -(1 - h*p(xk)/2);
        bb[k] := h^2*f(xk) + (1 + h*p(xk)/2)*al;
    for k from 2 to n-1 do
        xk := a + k*h;
        A[k, k] := 2 + h^2*q(xk);
        A[k, k-1] := -(1 + h*p(xk)/2);
```

```
        A[k, k+1] := -(1 - h·p(xk)/2);
        bb[k] := h^2·f(xk);
    end do;
    k := n :
        xk := a + k·h;
        A[k, k] := 2 + h^2·q(xk);
        A[k, k-1] := -2;
        bb[k] := h^2·f(xk) + (1 - h·p(xk)/2)·2 h be;

    u[it] := A^(-1).bb;
end do:
evalf(u[1]^(%T))
```

$$\left[\begin{array}{ccccc} 0.7568758562 & 1.468826946 & 2.049431628 & 2.373015183 & 2.278832693 \end{array}\right] \quad (5.1)$$

```
evalf(u[2]^(%T))
```

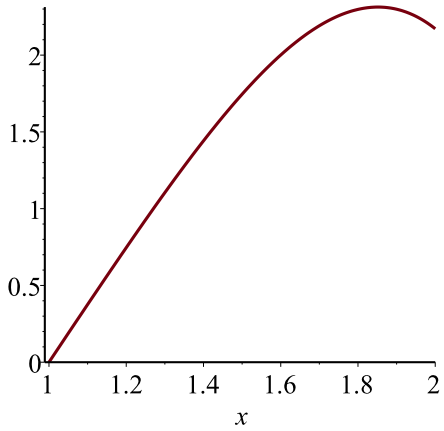
$$[0.3768648057, 0.7487054320, 1.109051883, 1.448396854, 1.754565212, 2.012946081, 2.206643611, 2.316577822, 2.321553777, 2.198310114] \quad (5.2)$$

The exact solution & Error analysis:

```
DE := diff(y(x), x, x) = 1/x diff(y(x), x) + 3/x^2 y(x) - 4 x^2 :
BC := y(1) = 0, D(y)(2) = -2 :
dsolve({DE, BC}, y(x))
```

$$y(x) = -\frac{8}{7x} + \frac{68}{35}x^3 - \frac{4}{5}x^4 \quad (6.1)$$

```
ey := x -> -8/(7 x) + 68/35 x^3 - 4/5 x^4 :
ey(x) :
plot(ey(x), x = 1 ..2)
```




```
n := 5 :
h := (b − a) / n :
for k to 5 do
    xk := a + k·h;
    yk := ey(xk);
    err1 := abs(yk-u[1][k]);
    err2 := abs(yk − u[2][2 k]);
    richardson := (4·u[2][2 k] − u[1][k]) /3;
    err3 := abs(yk − richardson);
    if (k = 1) then
        printf(" exact_y(x)  u(h=1/5)  error  u(h=1/10)  error
    Richardson  error\n");
    end if;
    printf(" x=%4.2f  %8.5f | %8.5f  %8.5f | %8.5f  %8.5f | %8.5f
    %9.4g\n", xk, yk, u[1][k], err1, u[2][2 k], err2, richardson, err3);
end do:
```

	exact_y(x)	u(h=1/5)	error	u(h=1/10)	error	Richardson	error
x=1.20	0.74600	0.75688	0.01088	0.74871	0.00271	0.74598	1.423e-05
x=1.40	1.44159	1.46883	0.02723	1.44840	0.00680	1.44159	6.646e-06
x=1.60	2.00078	2.04943	0.04865	2.01295	0.01217	2.00078	7.089e-06
x=1.80	2.29774	2.37302	0.07527	2.31658	0.01884	2.29777	2.315e-05
x=2.00	2.17143	2.27883	0.10740	2.19831	0.02688	2.17147	4.068e-05

Richardson's Extrapolation

Let u_h be a second-order approximation of y. Then, it follows from Equation (3) that

$$y = u_h + K_2 h^2 + K_4 h^4 + \mathrm{O}(h^6) \tag{9}$$

for some K_2 and K_4. Similarly, when u_{2h} is the numerical solution of y with the mesh size $2h$, we have

$$y = u_{2h} + K_2 (2h)^2 + K_4 (2h)^4 + \mathrm{O}((2h)^6)$$

and therefore

$$y = u_{2h} + 4 K_2 h^2 + 16 K_4 h^4 + \mathrm{O}(h^6) \tag{10}$$

Subtracting Equation (10) from four times Equation (9) reads

$$3y = 4u_h - u_{2h} - 12 K_4 h^4 + \mathrm{O}(h^6)$$

which implies that

$$y = \frac{4u_h - u_{2h}}{3} - 4 K_4 h^4 + \mathrm{O}(h^6)$$

Implication:

A simple combination of two solutions of different mesh size results in an approximation of y that shows a fourth-order truncation error. Such an idea, called the **Richardson's extrapolation**, can be applied for a sixth-order approximate solution when the second-order numerical solutions are computed with three different mesh sizes, h, $2h$, $4h$.

Empty Page

11.3. Finite Element Methods

To describe finite element methods (FEM), we consider approximating the solution to a linear two-point boundary-value problem of the form

$$\text{(D)} \qquad \begin{cases} -\dfrac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\dfrac{\mathrm{d}u}{\mathrm{d}x}\right) = f(x), \quad 0 \le x \le 1, \\ u(0) = 0, \quad u(1) = 0 \end{cases}$$

The differential equation describes the displacement $u(x)$ of a uniaxial rod of length 1, due to distributed force/length $f(x)$.

A little bit of physics:

Distributed force/length	$f(x)$
Axial (normal) stress	σ_{xx}
Internal axial force	$P = \sigma_{xx} A$, A = cross-sectional area
Uniaxial Hooke's law	$E\varepsilon_{xx} = \sigma_{xx} - \nu(\sigma_{yy} + \sigma_{yy}) = \sigma_{xx}$
Force equilibrium	$\dfrac{\mathrm{d}P}{\mathrm{d}x} + f = 0$

$$P = \sigma_{xx} A = E\varepsilon_{xx} \cdot A = EA\,\varepsilon_{xx} = EA\,\frac{\mathrm{d}u}{\mathrm{d}x}$$

Thus, in the above model, $p(x) = E \cdot A(x)$.

We assume that $p \in C^1[0, 1]$ and there exists a constant $\delta > 0$ such that

$$\delta \le p(x), x \in [0, 1] \qquad \textbf{(1)}$$

Variational Formulation:

Define a linear space

$$V = \{ v : v \in C^0[0,\ 1],\ v' \text{ is piecewise continuous on } [0,\ 1], \text{ and } v(0) = v(1) = 0 \}$$

Since, by the integration by parts,

$$\int_0^1 -(p\,u')'\,v\,dx = [\,-p\,u'\,v]_{x=0}^{x=1} + \int_0^1 p\,u'\,v'\,dx = \int_0^1 p\,u'\,v'\,dx, \quad v \in V$$

the differential problem (D) can be written as the **weak form**

$$\int_0^1 p \left(\frac{d}{dx} u(x) \right) \left(\frac{d}{dx} v(x) \right) dx = \int_0^1 f(x)\,v(x)\,dx \tag{2}$$

Define a bilinear functional and a linear product as

$$B(u, v) = \int_0^1 p \left(\frac{d}{dx} u(x) \right) \left(\frac{d}{dx} v(x) \right) dx \tag{3}$$

$$(f, v) = \int_0^1 f(x)\,v(x)\,dx \tag{4}$$

Then, Equation **(2)** reads, for $v \in V$,

$$B(u, v) = (f, v) \tag{5}$$

Now, we define the variational problem corresponding to the differential problem (D):

(V) Find $u \in V$ such that $B(u, v) = (f, v),\ \forall\, v \in V$

Theorem: The differential problem (D) is equivalent to the variational problem (V), when the solutions are sufficiently smooth.

Minimization Problem:

Let $F : V \to \mathbb{R}$ be a functional defined as

$$F(v) = \frac{1}{2} B(v, v) - (f, v)$$

Consider the following minimization problem

(M)	Find $u \in V$ such that $F(u) \leq F(v), \ \forall v \in V$

Theorem: The minimization problem (M) is equivalent to the variational problem (V).

Theorem: Problem (V) admits a unique solution, provided the condition **(1)** is satisfied.

In summary:

- (D) $\Leftrightarrow$ (V) $\Leftrightarrow$ (M) [(D) $\Leftarrow$ (V), when u" exists]
- They admit a unique solution.

Formulation of FEMs

- **Partitioning**: The domain is partitioned into a collection of elements of the maximum mesh size h.

- **Subspace $V_h \subset V$ and basis functions $\{\phi_j\}$**: A finite-dimensional subspace is set to represent the numerical solution as a linear combination of basis functions

$$u_h(x) = \sum_{j=1}^{n} c_j \phi_j(x) \tag{6}$$

  Examples are piecewise polynomials (splines)
  - linear, quadratic, cubic, etc.

- **Application of variational principles**: Different FEMs are formulated with various variational principles.

  Examples are
  - the minimization principle (Rayleigh-Ritz),
  - weighted residual approaches with the weights being either the basis functions (Galerkin) or different functions (Petrov-Galerkin),
  - least-square approaches, and
  - collocation evaluation.

- **Assembly for a linear system**: The linear system can be assembled for the unknown $(c_1, c_2, \cdots, c_n)^T$ with the integrals approximated by numerical quadratures.

1. Partitioning

Let

$$0 = x_0 < x_1 < x_2 < \cdots < x_m = 1$$

be a partition of the unit interval $[0, 1]$.

Define

$$I_j = \left[x_{j-1}, x_j\right], \quad h_j = x_j - x_{j-1} \quad (1 \leq j \leq m)$$

and

$$h = \max_{1 \leq j \leq m} h_j$$

2. Subspace & Basis functions

Define a finite-dimensional subspace of V as

$$V_h = \{ v \in V : \ v \text{ is a polynomial of degree } \leq k \text{ on each } I_j \}$$

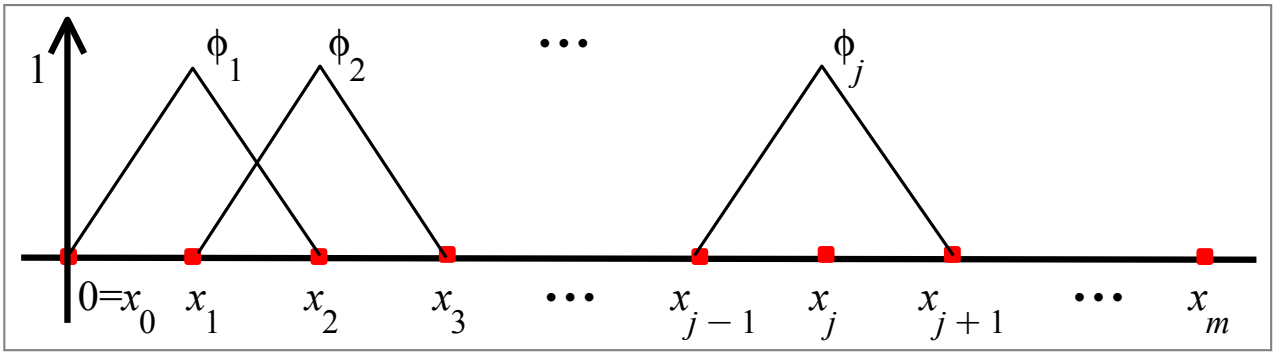
Corresponding basis functions are determined depending on the choice of polynomial degree $k \geq 1$ and therefore on the nodal points.

Each of basis functions is related to a nodal point.

Linear FEM ($k = 1$):

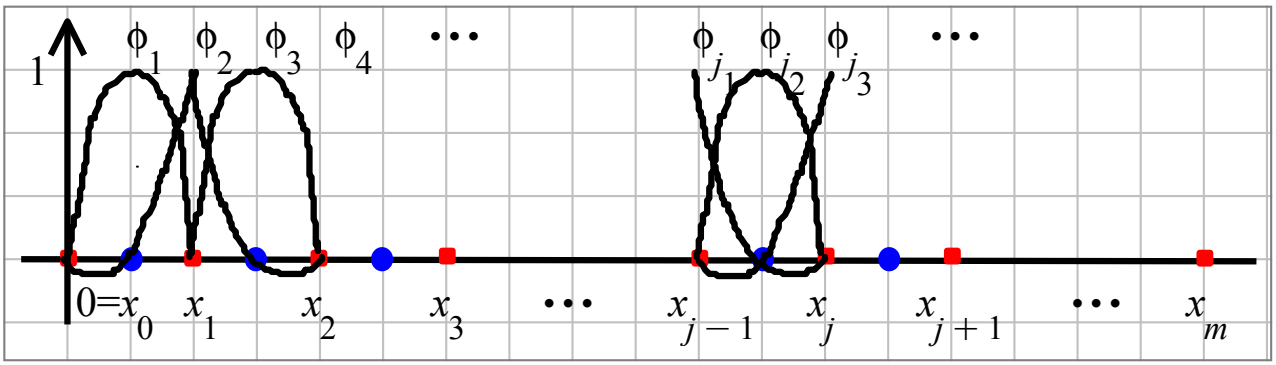
For the linear FEM, the nodal points coincide with the grid points $\{x_j\}$. The basis function associated with x_j is defined as

$$\phi_j(x) = \begin{cases} 0 & x < x_{j-1} \\ \dfrac{1}{h_j}\left(x - x_{j-1}\right) & x \in \left[x_{j-1}, x_j\right] \\ \dfrac{1}{h_{j+1}}\left(x_{j+1} - x\right) & x \in \left[x_j, x_{j+1}\right] \\ 0 & x_{j+1} < x \end{cases}$$



Quadratic FEM ($k = 2$):

On each subinterval $I_j = [x_{j-1}, x_j]$, the basis function is a quadratic polynomial, which requires to determine three coefficients; three nodal points must be set on each subinterval. The two endpoints can naturally become nodal points. We may **select the center of the subinterval for the extra nodal point.**



$$\phi_{2j-1}(x) = \begin{cases} 0 & x < x_{j-1} \\ -\frac{4}{h_j^2}\left(x - x_{j-1}\right)\left(x - x_j\right) & x \in \left[x_{j-1}, x_j\right] \\ 0 & x_j < x \end{cases}$$

$$\phi_{2j}(x) = \begin{cases} 0 & x < x_{j-1} \\ \frac{2}{h_j^2}\left(x - x_{j-1}\right)\left(x - x_{j-1/2}\right) & x \in \left[x_{j-1}, x_j\right] \\ \frac{2}{h_{j+1}^2}\left(x - x_{j+1/2}\right)\left(x - x_{j+1}\right) & x \in \left[x_j, x_{j+1}\right] \\ 0 & x_{j+1} < x \end{cases}$$

Higher-order FEMs ($k \geq 3$):

For each interval $I_j = [x_{j-1}, x_j]$, the **degree of freedom** of k-th order polynomials is $k+1$ and therefore it requires to choose $k+1$ nodal points. As for the quadratic FEM, the two endpoints can naturally become nodal points. We should select $k-1$ extra nodal points inside the j-th interval I_j.

In the literature, a common practice is to select those nodal points in such a way that the numerical quadrature of the integrals is as accurate as possible when the nodal points are used as quadrature points. Such selection is related to the family of orthogonal polynomials such as Legendre polynomials and Chebyshev polynomials.

Theorem (Gauss-Lobatto Integration):

Let $x_0 = -1$ and $x_k = 1$ and $\{x_1, x_2, \cdots, x_{k-1}\}$ are the roots of the first-derivative of the k-th Legendre polynomial, $P_k'(x)$. Let $\{w_0, w_2, \cdots, w_k\}$ be obtained by

$$w_i = \int_{-1}^{1} \prod_{j=0, j\neq i}^{k} \frac{x - x_j}{x_i - x_j} \, dx \qquad (0 \leq i \leq k)$$

Then,

$$\int_{-1}^{1} f(x) \, dx = \sum_{i=0}^{k} w_i \, f(x_i) \quad \forall f \in \mathbb{P}_{2k-1}$$

Legendre Polynomials:

```
kmax := 5 :
for k from 0 to kmax do
    Legendre[k] := sort(orthopoly[P](k, t));
end do
```

$$1$$

$$t$$

$$\frac{3}{2}t^2-\frac{1}{2}$$

$$\frac{5}{2}t^3-\frac{3}{2}t$$

$$\frac{35}{8}t^4-\frac{15}{4}t^2+\frac{3}{8}$$

$$\frac{63}{8}t^5-\frac{35}{4}t^3+\frac{15}{8}t \qquad \textbf{(6.1)}$$

Gauss-Lobatto nodal points:

```
for k from 2 to kmax do
    Lpzeros_k := solve(diff(Legendre[k], t) = 0, t);
end do;
```

$$0$$

$$\frac{1}{5}\sqrt{5},\ -\frac{1}{5}\sqrt{5}$$

$$0,\ \frac{1}{7}\sqrt{21},\ -\frac{1}{7}\sqrt{21}$$

$$\frac{1}{21}\sqrt{147-42\sqrt{7}},\ -\frac{1}{21}\sqrt{147-42\sqrt{7}},\ \frac{1}{21}\sqrt{147+42\sqrt{7}},\ -\frac{1}{21}\sqrt{147+42\sqrt{7}} \qquad \textbf{(6.1.1)}$$

```
with(Statistics) :
for k from 2 to kmax do
    T[k] := Sort([-1, 1, Lpzeros_k]);
    printf("k=%d:  %2d", k, T[k][1]);
    for i from 2 to k do printf("  %12.10f", T[k][i]); end do;
    printf("  %2d\n", T[k][k+1]);
```

```
end do:
```

```
k=2:  -1  0.0000000000   1
k=3:  -1  -0.4472135954  0.4472135954   1
k=4:  -1  -0.6546536709  0.0000000000  0.6546536709   1
k=5:  -1  -0.7650553238  -0.2852315163  0.2852315163  0.7650553238   1
```

Gauss-Lobatto weights:

```
stringout := 1 :
for k from 2 to kmax do
    for i from 0 to k do
        w[k][i + 1] := simplify(int(mul((x - T[k][j + 1]) / (T[k][i
    + 1] - T[k][j + 1]), j = 0 ..i - 1)·mul((x - T[k][j + 1]) / (T[k][i
    + 1] - T[k][j + 1]), j = i + 1 ..k), x = -1 ..1));
    end do;
    if (stringout = 1) then
        printf("k=%d:  %s", k, convert(w[k][1], string));
        for i from 2 to k do printf("  %s", convert(w[k][i], string)); end
    do;
        printf("  %s\n", convert(w[k][k + 1], string));
    else
        printf("k=%d:  %12.10f", k, w[k][1]);
        for i from 2 to k do printf("  %12.10f", w[k][i]); end do;
        printf("  %12.10f\n", w[k][k + 1]);
    end if;
end do:
```

```
k=2:  1/3  4/3  1/3
k=3:  1/6  5/6  5/6  1/6
k=4:  1/10  49/90  32/45  49/90  1/10
k=5:  1/15  -1/30*7^(1/2)+7/15  1/30*7^(1/2)+7/15  1/30*7^(1/2)+7/15
-1/30*7^(1/2)+7/15  1/15
```

The maps P and S defined by

$$P(t) = \frac{x_j - x_{j-1}}{2} t + \frac{x_{j-1} + x_j}{2} : [-1, 1] \to I_j = [x_{j-1}, x_j]$$

$$S(w) = \frac{x_j - x_{j-1}}{2} w$$

transform the nodal points $\{t_i\}_{i=0}^{k} \subset [-1, 1]$ and weights $\{w_i\}_{i=0}^{k}$ respectively to I_j and corresponding weights. Note that

$$\sum_{j=0}^{k} S(w_j) = x_j - x_{j-1} \tag{6.2}$$

which is the length of the j-th subinterval.

3. Application of variational principles

Once basis functions are determined, the approximate solution u_h can be formulated as in **(6)**, that is,

$$u_h(x) = \sum_{j=1}^{n} c_j \phi_j(x) \tag{7}$$

Rayleigh-Ritz Method:

The method seeks an approximate solution of the form **(7)** which satisfies

$$u_h = \arg \min_{v \in V_h} F(v)$$

where

$$F(v) = \frac{1}{2} B(v, v) - (f, v) =$$

$$\frac{1}{2} \int_0^1 p(x) \left(\frac{\mathrm{d}}{\mathrm{d}x} v(x) \right) \left(\frac{\mathrm{d}}{\mathrm{d}x} v(x) \right) \mathrm{d}x - \left(\int_0^1 f(x)\, v(x)\, \mathrm{d}x \right) \tag{8}$$

It follows from **(7)** and **(8)** that

$$F(u_h) = \frac{1}{2} \int_0^1 p(x) \left(\sum_{j=1}^{n} c_j \left(\frac{\mathrm{d}}{\mathrm{d}x} \phi_j(x) \right) \right)^2 \mathrm{d}x - \left(\int_0^1 f(x) \left(\sum_{j=1}^{n} c_j \phi_j(x) \right) \mathrm{d}x \right) \tag{9}$$

In order for a minimum to occur, it is necessary, when considering $F(u_h)$ as a function of $(c_1, c_2, \cdots, c_n)$, to have the normal equations

$$\frac{\partial}{\partial c_i} F(u_h) = 0, \quad 1 \le i \le n$$

Rayleigh-Ritz Method (2)

Differentiating **(9)** gives

$$\frac{\partial}{\partial c_i} F(u_h) = \int_0^1 p(x) \left(\sum_{j=1}^{n} c_j \left(\frac{\mathrm{d}}{\mathrm{d}x} \phi_j(x) \right) \left(\frac{\mathrm{d}}{\mathrm{d}x} \phi_i(x) \right) \right) \mathrm{d}x - \left(\int_0^1 f(x)\, \phi_i(x)\, \mathrm{d}x \right)$$

which equivalently reads, for $i = 1, 2, \cdots, n$,

$$B(u_h, \phi_i) = (f, \phi_i) \tag{10}$$

and yields

$$\sum_{j=1}^{n} \left(\int_0^1 p(x) \left(\frac{\mathrm{d}}{\mathrm{d}x} \phi_j(x) \right) \left(\frac{\mathrm{d}}{\mathrm{d}x} \phi_i(x) \right) \mathrm{d}x \right) c_j - \left(\int_0^1 f(x)\, \phi_i(x)\, \mathrm{d}x \right) = 0 \tag{11}$$

Note that $(M) \Leftrightarrow (V)$.

Letting

$$a_{ij} = \int_0^1 p(x) \left(\frac{\mathrm{d}}{\mathrm{d}x} \phi_j(x) \right) \left(\frac{\mathrm{d}}{\mathrm{d}x} \phi_i(x) \right) \mathrm{d}x,\ b_i = \int_0^1 f(x)\, \phi_i(x)\, \mathrm{d}x \tag{12}$$

the normal equations in **(11)** produce an $n \times n$ linear system

$$A\, c = b \tag{13}$$

4. Assembly for a linear system

As shown in the previous page, the algebraic system can be obtained by integrating the quantities in Equation **(12)**. Such a process of constructing a matrix system is called *assembly*.
Here we consider the simplest case, the linear FEM.

Recall the basis functions for the linear FEM are defined as

$$\phi_j(x) = \begin{cases} \dfrac{1}{h_j}\left(x - x_{j-1}\right) & x \in \left[x_{j-1}, x_j\right] \\ \dfrac{1}{h_{j+1}}\left(x_{j+1} - x\right) & x \in \left[x_j, x_{j+1}\right] \\ 0 & \text{elsewhere} \end{cases}$$

Thus its derivative reads

$$\phi_j'(x) = \begin{cases} \dfrac{1}{h_j} & x \in \left[x_{j-1}, x_j\right] \\ -\dfrac{1}{h_{j+1}} & x \in \left[x_j, x_{j+1}\right] \\ 0 & \text{elsewhere} \end{cases}$$

Since ϕ_j and ϕ_j' are nonzero only on $I_j \cup I_{j+1} = \left[x_{j-1}, x_{j+1}\right]$,

$$\left(\frac{\mathrm{d}}{\mathrm{d}x}\phi_j(x)\right)\left(\frac{\mathrm{d}}{\mathrm{d}x}\phi_i(x)\right) = 0 \qquad \textbf{(14)}$$

except for $j = i - 1$, i, or $i + 1$.

As a consequence, the linear system given by Equation **(13)** reduces to an $n \times n$ tridiagonal linear system.

Assembly for a linear system (2)

The nonzero entries in the i-th row of A are $a_{i,\,i-1}$, a_{ii}, and $a_{i,\,i+1}$.

Computation of a_{ii}:

$$a_{ii} = \int_0^1 p(x)\ \phi_i'(x)\ \phi_i'(x)\ dx = \left(\frac{1}{h_i}\right)^2 \int_{x_i-1}^{x_i} p(x)\ dx + \left(-\frac{1}{h_{i+1}}\right)^2 \int_{x_i}^{x_i+1} p(x)\ dx$$

$$\approx \left(\frac{1}{h_i}\right)^2 \frac{h_i}{2}\left(p_{i-1}+p_i\right) + \left(\frac{1}{h_{i+1}}\right)^2 \frac{h_{i+1}}{2}\left(p_i+p_{i+1}\right)$$

Thus we have

$$\left(a_{ii}\right) \approx \left(\frac{1}{2}\,\frac{p_{i-1}+p_i}{h_i} + \frac{1}{2}\,\frac{p_i+p_{i+1}}{h_{i+1}}\right) \tag{7.1}$$

where the trapezoidal quadrature rule has been applied.

Computation of $a_{i,\,i-1}$:

$$a_{i,\,i-1} = \int_0^1 p(x)\ \phi_{i-1}'(x)\ \phi_i'(x)\ dx = -\left(\frac{1}{h_i}\right)^2 \int_{x_i-1}^{x_i} p(x)\ dx$$

$$\approx -\left(\frac{1}{h_i}\right)^2 \frac{h_i}{2}\left(p_{i-1}+p_i\right)$$

and therefore

$$\left(a_{i,i-1}\right) \approx \left(-\frac{1}{2}\,\frac{p_{i-1}+p_i}{h_i}\right) \tag{7.2}$$

Computation of $a_{i,i+1}$: Similarly,

$$a_{i,i+1} = \int_0^1 p(x)\ \phi_{i+1}'(x)\ \phi_i'(x)\ \mathrm{d}x = -\left(\frac{1}{h_{i+1}}\right)^2 \int_{x_i}^{x_{i+1}} p(x)\ \mathrm{d}x$$

$$\approx -\left(\frac{1}{h_{i+1}}\right)^2 \frac{h_{i+1}}{2}\left(p_i + p_{i+1}\right)$$

Thus,

$$\left(a_{i,i+1}\right) \approx \left(-\frac{1}{2}\,\frac{p_i+p_{i+1}}{h_{i+1}}\right) \tag{7.3}$$

Computation of b_i:

$$b_i = \int_0^1 f(x)\ \phi_i(x)\ \mathrm{d}x = \int_{x_{i-1}}^{x_i} f(x)\,\frac{1}{h_i}\left(x - x_{j-1}\right)\ \mathrm{d}x$$

$$+ \int_{x_i}^{x_{i+1}} f(x)\,\frac{1}{h_{i+1}}\left(x_{j+1} - x\right)\ \mathrm{d}x$$

$$\approx \frac{h_i}{2}\left(f_{i-1}\cdot 0 + f_i\cdot 1\right) + \frac{h_{i+1}}{2}\left(f_i\cdot 1 + f_{i+1}\cdot 0\right)$$

Hence , we have

$$\left(b_i\right) \approx \left(\frac{1}{2}\,h_i f_i + \frac{1}{2}\,h_{i+1} f_i\right) \tag{7.4}$$

Summary for the Linear FEM:

The i-th row of the algebraic system

$$A\,c = b \tag{15}$$

can be written as

$$a_{i,i-1}\,c_{i-1} + a_{i,i}\,c_i + a_{i,i+1}\,c_{i+1} = b_i \tag{16}$$

where

$$a_{i,i-1} = -\frac{p_{i-1}+p_i}{2\,h_i}$$

$$a_{i,i} = \frac{p_{i-1}+p_i}{2\,h_i} + \frac{p_i+p_{i+1}}{2\,h_{i+1}}$$

$$a_{i,i+1} = -\frac{p_i+p_{i+1}}{2\,h_{i+1}}$$

$$b_i = f_i\,\frac{h_i+h_{i+1}}{2}$$

Note

When $h_i = h$ for all $i = 1 \ldots n$ and $p(x) \equiv 1$, equation **(16)** becomes

$$-\frac{c_{i-1}}{h} + \frac{2\,c_i}{h} - \frac{c_{i+1}}{h} = f_i\,h \tag{8.1}$$

Dividing both sides by h, we have

$$\frac{-c_{i-1} + 2\,c_i - c_{i+1}}{h^2} = f_i \tag{8.2}$$

which is the same equation obtained from the second-order **finite difference method**.

Example: Use the **linear Rayleigh-Ritz FEM** to approximate the solution to the boundary-value problem

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left((x+1)\,\frac{\mathrm{d}}{\mathrm{d}x}u\right)+6u=-12x^4+44x^3-2x+1,\quad 0\le x\le 1,$$

$$u(0)=u(1)=0$$

using $x_0=0$, $x_1=0.3$, $x_2=0.7$, and $x_3=1$. Compare your result with the actual solution $u(x)=-2x^4+2x^3-x^2+x$. (Note that this problem involves a reaction term.)

Solution:

$p := x \to x+1:$
$q := x \to 6:$
$f := x \to -12x^4+44x^3-2x+1:$
$u := x \to -2x^4+2x^3-x^2+x:$

 # Partitioning:

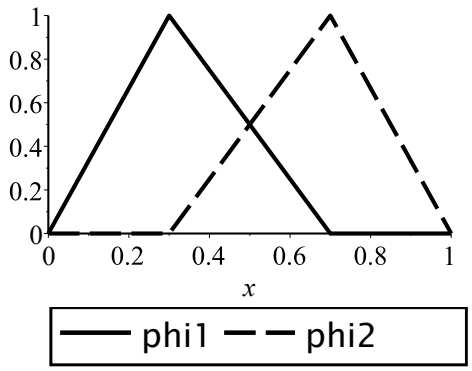
$m := 3:$
$x_0 := 0:\ x_1 := 0.3:\ x_2 := 0.7:\ x_3 := 1:$

 # Basis functions and u_h:

Since $u(x_0)=u(x_3)=0$, we only need to define

$$u_h = c_1\phi_1 + c_2\phi_2 \qquad \textbf{(10.1)}$$

where



 # Variational principle and Assembly:

$A := Matrix(m+1, m+1):$
$b := Vector(m+1):$

Element-wise Assembly with Lagrange form of linear basis functions: with exact evaluation of integrals.

for k **from** 1 **to** m **do**
$g1 := k-1:\ \ g2 := k:$
$x1 := x_{k-1}:\ \ x2 := x_k:$
$ph[1] := x \to \frac{(x-x2)}{x1-x2}:\quad ph[2] := x \to \frac{(x-x1)}{x2-x1}:$
for i **to** 2 **do**
for j **to** 2 **do**
$Ae[k][i,j] := int(p(x)\cdot diff(ph[j](x), x)\cdot diff(ph[i](x), x), x = x1..x2)$
$+ int(q(x)\cdot ph[j](x)\cdot ph[i](x), x = x1..x2);$
end do;
$be[k][i] := int(f(x)\cdot ph[i](x), x = x1..x2);$
end do;
You may do the below, along with the above
for i **to** 2 **do**
for j **to** 2 **do**

```
            A[g1 + i, g1 + j] := A[g1 + i, g1 + j] + Ae[k][i, j];
        end do;
        b[g1 + i] := b[g1 + i] + be[k][i];
    end do;
end do:
```

 # Print A and b:

$$A = \begin{bmatrix} 4.433333333 & -3.533333333 & 0 & 0 \\ -3.533333333 & 8.983333333 & -3.350000000 & 0 \\ 0 & -3.350000000 & 11.31666667 & -5.866666666 \\ 0 & 0 & -5.866666666 & 6.766666666 \end{bmatrix}$$

$$b = \begin{bmatrix} 0.1368480000 \\ 0.8939186667 \\ 4.013585333 \\ 3.555648000 \end{bmatrix}$$

 # Incorporation of the Dirichlet Boundary Condition:

$A[1, 1] := 1:$ **for** j **from** 2 **to** $m + 1$ **do** $A[1, j] := 0$: **end do**:
$A[m + 1, m + 1] := 1:$ **for** j **to** m **do** $A[m + 1, j] := 0$: **end do**:
$b[1] := 0:\ \ b[m + 1] := 0:$

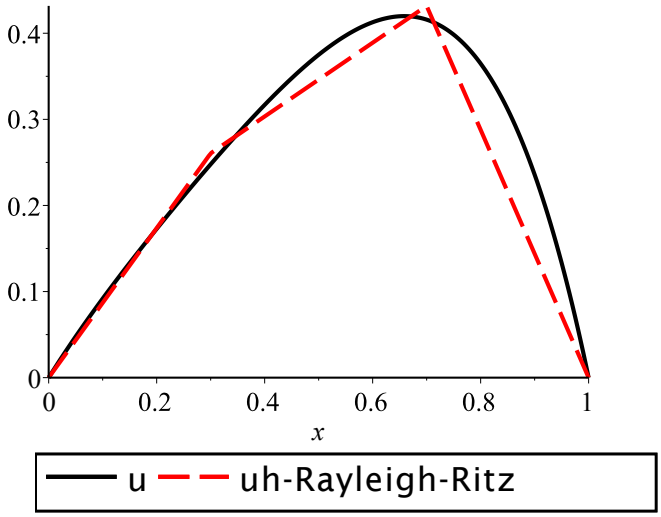
$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3.533333333 & 8.983333333 & -3.350000000 & 0 \\ 0 & -3.350000000 & 11.31666667 & -5.866666666 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 \\ 0.8939186667 \\ 4.013585333 \\ 0 \end{bmatrix}$$

$c := A^{-1}.b$

$$\begin{bmatrix} 0. \\ 0.260526093689779 \\ 0.431783305928261 \\ 0. \end{bmatrix} \quad \textbf{(10.2)}$$

plotting



(3)'. Other Variational Principles: Weighted Residual Approaches

We will go back to the 3rd step of FEM formulation for other variational principles; so far, we have considered Rayleigh-Ritz Method which employs the minimization principle.

A WRA begins with defining the residual. Let

$$P(u) = -\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right)$$

The residual R is defined as

$$R(v) = P(v) - f \tag{17}$$

Then, $R(u) = P(u) - f = 0$; however, for $u_h(x) = \sum_{j=1}^{n} c_j \phi_j(x)$,

$$R(u_h) = P(u_h) - f \neq 0, \text{ in general.}$$

Weighed residual approaches are seeking an approximate solution

$$u_h(x) = \sum_{j=1}^{n} c_j \phi_j(x) \tag{18}$$

which satisfies

$$\int_0^1 R(u_h)\, w(x)\, \mathrm{d}x = 0 \tag{19}$$

for a sequence of weight functions $w(x)$, which is also called a trial function. When the integration by parts is utilized, Equation **(19)** can be written as

$$B(u_h, w) = (f, w) \tag{20}$$

Choices of Weight (Trial/Test) Functions

Methods	Weight functions
Galerkin method	$w(x) = \phi_i(x)$, the basis functions
Petrov-Galerkin method	$w(x) \neq \phi_i(x)$
Least-Squares method	$w(x) = \frac{\partial}{\partial c_i} R(u_h) = P(\phi_i(x))$
Collocation method	$w(x) = \delta(x - x_i)$

Galerkin method

$$R\left(u_h(x)\right) = -\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}}{\mathrm{d}x}u_h(x)\right) - f(x)$$
$$= -\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}}{\mathrm{d}x}\sum_{j=1}^{n} c_j \phi_j(x)\right) - f(x)$$

$$\int_0^1 R\left(u_h\right) w(x)\,\mathrm{d}x = 0$$

The weight function are the basis functions: for $= 1, 2, \bullet\bullet\bullet, n$,

$$w(x) = \phi_i(x) \qquad \textbf{(21)}$$

Utilizing the integration by parts, it is easy to derive

$$0 = \int_0^1 R\left(u_h\right)\,\phi_i(x)\,\mathrm{d}x = \int_0^1 p(x)\left(\sum_{j=1}^{n} c_j \left(\frac{\mathrm{d}}{\mathrm{d}x}\,\phi_j(x)\right)\left(\frac{\mathrm{d}}{\mathrm{d}x}\,\phi_i(x)\right)\right)\mathrm{d}x - \int_0^1 f(x)\,\phi_i(x)\,\mathrm{d}x$$

which in turn implies, for $i = 1, 2, \bullet\bullet\bullet, n$,

$$B\left(u_h, \phi_i\right) = \left(f, \phi_i\right) \qquad \textbf{(22)}$$

Equivalence between Rayleigh-Ritz and Galerkin methods:

Equation **(22)** for Galerkin FEM is the same as Equation **(10)** for Rayleigh-Ritz FEM. In general, the two methods are equivalent for linear partial differential equations.

Petrov-Galerkin method

$$\int_0^1 R(u_h)\, \psi_i(x)\, dx = 0$$

The weight function are the basis functions: for $= 1, 2, \bullet\bullet\bullet, n$,

$$\psi_i(x) \neq \phi_i(x) \qquad \textbf{(23)}$$

Various Petrov-Galerkin methods have been developed with various choices of weight functions, particularly for convection-dominated fluid problems. We will not deal with those problems here. If you are interested in those, try "Numerical Solutions of PDEs II".

Least-Squares method

$$u_h = \arg \min_{v \in V_h} \int_0^1 R(v)^2 \, \mathrm{d}x$$

which is equivalent to

$$\frac{\partial}{\partial c_i} \int_0^1 R\left(u_h\right)^2 \mathrm{d}x = 0, \quad i = 1, 2, \bullet\bullet\bullet, n$$

Differentiating the integral reads

$$\int_0^1 R\left(u_h\right) w(x) \, \mathrm{d}x = 0$$

where

$$w(x) = \frac{\partial}{\partial c_i} R\left(u_h(x)\right) = -\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}}{\mathrm{d}x}\,\phi_i(x)\right) = P\left(\phi_i(x)\right)$$

Thus the Least-Squares FEM can be formulated as

$$\sum_{j=1}^{n}\left(\int_0^1 P\left(\phi_j(x)\right) P\left(\phi_i(x)\right) \mathrm{d}x\right) c_j = \int_0^1 f(x)\, P\left(\phi_i(x)\right) \mathrm{d}x \qquad \textbf{(24)}$$

Although rarely happening, utilizing the integration by parts gives

$$B\left(u_h, P\left(\phi_i\right)\right) = \left(f, P\left(\phi_i\right)\right) \qquad \textbf{(25)}$$

Note:
Due to the derivatives applied on the basis functions, the basis functions for least-squares methods must be high-order splines.

Collocation method

$$\int_0^1 R\left(u_h(x)\right)\, w(x)\, \mathrm{d}x = 0$$

The weight function are, for $= 1, 2, \cdots, n$,

$$w(x) = \delta\left(x - x_i\right) \tag{26}$$

The algebraic system:

Since

$$\begin{aligned} R\left(u_h(x)\right) &= -\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}}{\mathrm{d}x}u_h(x)\right) - f(x) \\ &= -\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}}{\mathrm{d}x}\sum_{j=1}^{n} c_j\,\phi_j(x)\right) - f(x) \\ &= \sum_{j=1}^{n} -\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}}{\mathrm{d}x}\,\phi_j(x)\right) c_j - f(x) \end{aligned}$$

and

$$\int_0^1 R\left(u_h(x)\right)\, \delta\left(x - x_i\right)\, \mathrm{d}x = R\left(u_h\left(x_i\right)\right)$$

when n nodal points are selected, the collocation method produces the $n \times n$ algebraic system:

$$\sum_{j=1}^{n} -\frac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\frac{\mathrm{d}}{\mathrm{d}x}\,\phi_j(x)\right)\Bigg|_{x = x_i} c_j = f\left(x_i\right) \qquad (1 \le i \le n)$$

Much attention in the literature has been given to the choice of the basis functions $\left\{\phi_j(x)\right\}$ and the collocation points $\left\{x_i\right\}$. One popular choice is to let $\left\{\phi_j(x)\right\}$ be cubic B-Spline functions and $\left\{x_i\right\}$ be Gaussian points.

Example: Use collocation method to approximate the solution to the boundary-value problem considered in the preceding example:

$$-\frac{d}{dx}\left((x+1)\frac{d}{dx}u\right)+6u=-12x^4+44x^3-2x+1,\quad 0\le x\le 1,$$
$$u(0)=u(1)=0$$

using three nodal points $x_1=1/4, x_2=1/2, x_3=3/4$ and basis functions

$$\phi_1(x)=x(x-1),\ \phi_2(x)=x^2(x-1),\ \phi_3(x)=x^2(x-1)^2$$

Compare your result with the actual solution $u(x)=-2x^4+2x^3-x^2+x$.

Solution:

```
restart
phi1 := x→x (x − 1) :  phi2 := x→x^2 (x − 1) :  phi3 := x→x^2 (x
    − 1)^2 :

x1 := 1/4 :  x2 := 1/2 : x3 := 3/4 :
uh := x→c1·phi1(x) + c2·phi2(x) + c3·phi3(x) :

-diff((x + 1)· diff(uh(x), x), x) + 6·uh(x)
```

$$-c1\,(x-1)-c1\,x-2\,c2\,x\,(x-1)-c2\,x^2-2\,c3\,x\,(x-1)^2-2\,c3\,x^2\,(x-1)\quad\textbf{(14.1)}$$
$$-(x+1)\left(2\,c1+2\,c2\,(x-1)+4\,c2\,x+2\,c3\,(x-1)^2+8\,c3\,x\,(x-1)\right.$$
$$\left.+2\,c3\,x^2\right)+6\,c1\,x\,(x-1)+6\,c2\,x^2\,(x-1)+6\,c3\,x^2\,(x-1)^2$$

```
Puh := x→ -c1 (x − 1) − c1 x − 2 c2 x (x − 1) − c2 x^2 − 2 c3 x (x
    − 1)^2 − 2 c3 x^2 (x − 1) − (x + 1) (2 c1 + 2 c2 (x − 1) + 4 c2 x
    + 2 c3 (x − 1)^2 + 8 c3 x (x − 1) + 2 c3 x^2) + 6 c1 x (x − 1)
    + 6 c2 x^2 (x − 1) + 6 c3 x^2 (x − 1)^2 :
f := x→ -12 x^4 + 44 x^3 − 2 x + 1 :
```

$$eq1 := Puh(x1) = f(x1) = -\frac{25}{8}\,c1 + \frac{21}{32}\,c2 + \frac{43}{128}\,c3 = \frac{73}{64}$$

$$eq2 := Puh(x2) = f(x2) = -\frac{9}{2}\,c1 - 2\,c2 + \frac{15}{8}\,c3 = \frac{19}{4}$$

$$eq3 := Puh(x3) = f(x3) = -\frac{41}{8}\,c1 - \frac{173}{32}\,c2 + \frac{107}{128}\,c3 = \frac{913}{64}$$

$$C := solve(\{eq1, eq2, eq3\}, \{c1, c2, c3\})$$

$$\{c1 = -1, c2 = -2, c3 = -2\} \quad \textbf{(14.2)}$$

$$c1 := rhs(C[1]) : \; c2 := rhs(C[2]) : \; c3 := rhs(C[3]) :$$

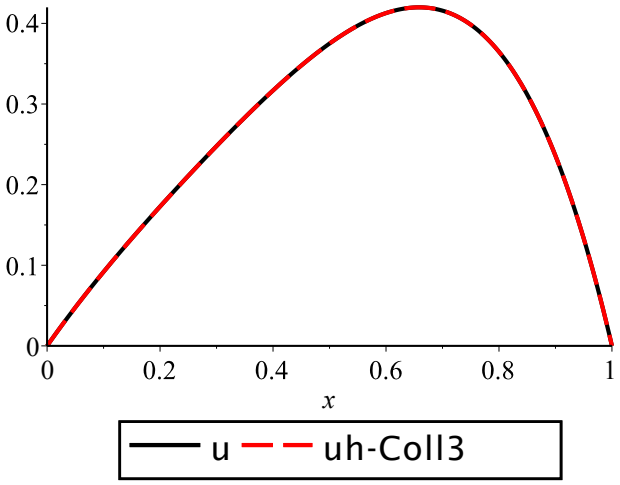
$$uh(x);$$

$$-x\,(x-1) - 2\,x^2\,(x-1) - 2\,x^2\,(x-1)^2 \quad \textbf{(14.3)}$$

$$u := -2\,x^4 + 2\,x^3 - x^2 + x$$

$$-2\,x^4 + 2\,x^3 - x^2 + x \quad \textbf{(14.4)}$$

▶ plotting



$$sort(expand(uh(x)))$$

$$-2\,x^4 + 2\,x^3 - x^2 + x \quad \textbf{(14.5)}$$

The Same Example: with two basis functions

$$\phi_1(x) = x(x-1),\ \phi_2(x) = x^2(x-1);\quad x_1 = 1/4,\ x_2 = 3/4$$

Solution:

```
restart
phi1 := x→x (x − 1) :  phi2 := x→x^2 (x − 1) :
x1 := 1/4 :  x2 := 3/4 :
uh := x→c1·phi1(x) + c2·phi2(x) :
```

computation

```
−diff((x + 1)· diff(uh(x), x), x) + 6·uh(x)
```

$$-c1\,(x-1) - c1\,x - 2\,c2\,x\,(x-1) - c2\,x^2 - (x+1)\,(2\,c1 + 2\,c2\,(x-1) + 4\,c2\,x) + 6\,c1\,x\,(x-1) + 6\,c2\,x^2\,(x-1) \quad \textbf{(15.1.1)}$$

```
Puh := x→ -c1 (x − 1) − c1 x − 2 c2 x (x − 1) − c2 x^2 − (x
    + 1) (2 c1 + 2 c2 (x − 1) + 4 c2 x) + 6 c1 x (x − 1) + 6 c2 x^2 (x
    − 1) :
f := x→ -12 x^4 + 44 x^3 − 2 x + 1 :
```

$$eq1 := Puh(x1) = f(x1) = -\frac{25}{8}\,c1 + \frac{21}{32}\,c2 = \frac{73}{64}$$

$$eq2 := Puh(x2) = f(x2) = -\frac{41}{8}\,c1 - \frac{173}{32}\,c2 = \frac{913}{64}$$

```
C := solve({eq1, eq2}, {c1, c2})
```

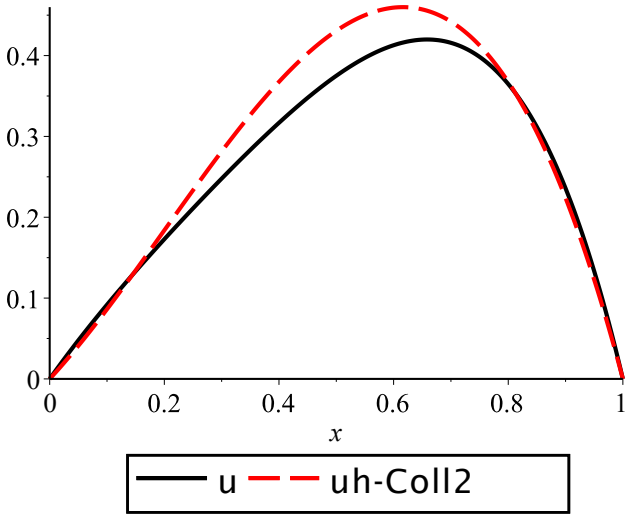
$$\left\{c1 = -\frac{15901}{20744},\ c2 = -\frac{4958}{2593}\right\} \quad \textbf{(15.1.2)}$$

```
c1 := rhs(C[1]) :  c2 := rhs(C[2]) :

uh(x);
```

$$-\frac{15901}{20744}\,x\,(x-1) - \frac{4958}{2593}\,x^2\,(x-1) \quad \textbf{(15.1)}$$

plotting



Example: Use the **least-squares FEM** to approximate the solution to the boundary-value problem considered in the preceding example:

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left((x+1)\,\frac{\mathrm{d}}{\mathrm{d}x}u\right)+6u=-12x^4+44x^3-2x+1,\quad 0\le x\le 1,$$

$$u(0)=u(1)=0$$

using a **single element** and **two basis functions**

$$\phi_1(x)=x(x-1),\ \phi_2(x)=x^2(x-1);$$

Compare your result with the actual solution $u(x)=-2x^4+2x^3-x^2+x$.

Solution:

restart

$p := x \to x+1:$

$q := x \to 6:$

$f := x \to -12x^4+44x^3-2x+1:$

$u := x \to -2x^4+2x^3-x^2+x:$

\# **Partitioning**

$x_0 := 0:\ x_1 := 1:$

$m := 1:$

\# Variational principle and Assembly: Equation (24)

$A := Matrix(m+1, m+1):$

$b := Vector(m+1):$

$P := u \to -diff(p(x)\cdot diff(u,x),x)+q(x)\cdot u$

$$u \to -\left(\frac{\partial}{\partial x}\left(p(x)\left(\frac{\partial}{\partial x}u\right)\right)\right)+q(x)\,u \qquad \textbf{(16.1)}$$

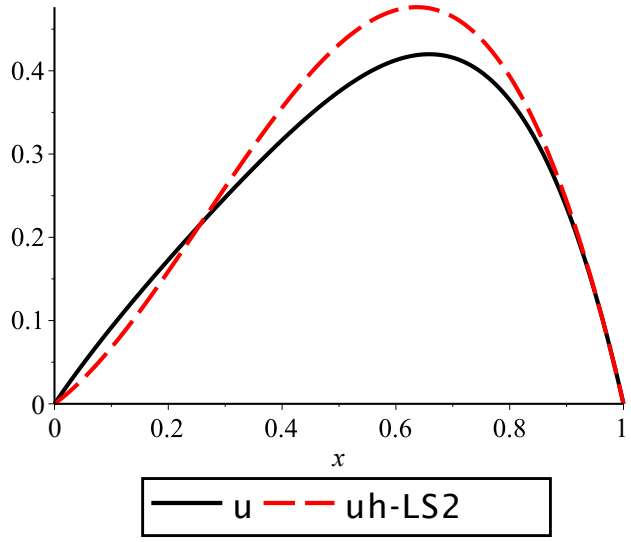
```
for k from 1 to m do
    g1 := k - 1 :  g2 := k :
    x1 := x[k-1] :  x2 := x[k] :
    ph[1] := x -> x (x - 1) :   ph[2] := x -> x^2 (x - 1) :
    for i to 2 do
        for j to 2 do
            Ae[k][i, j] := int(P(ph[j](x), x) . P(ph[i](x), x), x = x1 ..x2);
        end do;
        be[k][i] := int(f(x) . P(ph[i](x), x), x = x1 ..x2);
    end do;
    ## You may do the below, along with the above
    for i to 2 do
        for j to 2 do
            A[g1 + i, g1 + j] := A[g1 + i, g1 + j] + Ae[k][i, j];
        end do;
        b[g1 + i] := b[g1 + i] + be[k][i];
    end do;
end do:
```

$$A = \begin{bmatrix} \frac{263}{15} & \frac{41}{3} \\ \frac{41}{3} & \frac{1856}{105} \end{bmatrix} \qquad b = \begin{bmatrix} -\frac{4433}{105} \\ -\frac{5254}{105} \end{bmatrix}$$

```
c := A^(-1) . b :
uh := x -> c[1] . ph[1](x) + c[2] . ph[2](x) :
uh(x)
```

$$-\frac{229386}{452557}\, x\,(x-1) - \frac{157679}{64651}\, x^2\,(x-1) \qquad \textbf{(16.2)}$$

plotting



Empty Page

11.4. Finite Element Methods: More General Boundary-Value Problems

For a deeper understanding of FEMs, we consider approximating the solution to a more general boundary-value problem of the form

$$\text{(D)} \quad \begin{cases} -\dfrac{\mathrm{d}}{\mathrm{d}x}\left(p(x)\dfrac{\mathrm{d}u}{\mathrm{d}x}\right) + q(x)\dfrac{\mathrm{d}u}{\mathrm{d}x} + r(x)\,u = f(x), \quad 0 \le x \le 1, \\ u(0) = \alpha, \quad u'(1) = \beta \end{cases}$$

The Dirichlet BC is called an *essential* BC, while the Neumann BC is called a *natural* BC.

Variational Formulation:

Define a linear space

$$V = \{v : v \in C^0[0, 1],\ v' \text{ is piecewise continuous on } [0, 1], \text{ and } v(0) = 0\}$$

Since, by the integration by parts,

$$\int_0^1 -(p\,u')'\,v\,\mathrm{d}x = [-p\,u'\,v]_{x=0}^{x=1} + \int_0^1 p\,u'\,v'\,\mathrm{d}x$$

$$= -\beta\,p(1)v(1) + \int_0^1 p\,u'\,v'\,\mathrm{d}x, \quad v \in V$$

the differential problem (D) can be written as the **weak form**

$$\int_0^1 p\left(\frac{\mathrm{d}}{\mathrm{d}x}u(x)\right)\left(\frac{\mathrm{d}}{\mathrm{d}x}v(x)\right)\mathrm{d}x + \int_0^1 q\left(\frac{\mathrm{d}}{\mathrm{d}x}u(x)\right)v(x)\,\mathrm{d}x + \int_0^1 r\,u(x)\,v(x)\,\mathrm{d}x = \tag{1}$$

$$\int_0^1 f(x)\,v(x)\,\mathrm{d}x + \beta\,p(1)\,v(1)$$

Define a bilinear functional and a linear term as

$$B(u, v) = \int_0^1 p \left(\frac{\mathrm{d}}{\mathrm{d}x} u(x) \right) \left(\frac{\mathrm{d}}{\mathrm{d}x} v(x) \right) \mathrm{d}x + \int_0^1 q \left(\frac{\mathrm{d}}{\mathrm{d}x} u(x) \right) v(x)\, \mathrm{d}x + \tag{2}$$

$$\int_0^1 r\, u(x)\, v(x)\, \mathrm{d}x$$

$$\ell(v) = \int_0^1 f(x)\, v(x)\, \mathrm{d}x + \beta p(1)\, v(1) \tag{3}$$

Then, Equation (1) reads, for $v \in V$,

$$B(u, v) = \ell(v) \tag{4}$$

Now, we define the variational problem corresponding to the differential problem (D):

(V)	Find $u \in V$ such that $B(u, v) = \ell(v), \ \forall\, v \in V$

Theorem: The differential problem (D) is equivalent to the variational problem (V), when u'' exists.

The Galerkin FEM

• **Partitioning**:

$$0 = x_0 < x_1 < x_2 < \cdots < x_m = 1$$

with

$$I_j = [x_{j-1}, x_j]; \quad h_j = x_j - x_{j-1} \ (1 \le j \le m); \quad h = \max_{1 \le j \le m} h_j$$

• **Subspace**:

$$V_h = \{ v \in V : \ v \text{ is linear on each } I_j \}$$

When the basis functions are chosen as

$$\{ \phi_j(x) \}, \quad j = 1, 2, \cdots, m,$$

the approximate solution is

$$u_h(x) = \sum_{j=1}^{m} c_j \phi_j(x) \tag{1.1}$$

• **Application of variational principles**: The linear Galerkin FEM is formulated as

$$B(u_h, \phi_i) = \ell(\phi_i), \ i \in [1, m] \tag{1.2}$$

which can be written as

$$\sum_{j=1}^{m} B(\phi_j, \phi_i) \, c_j = \ell(\phi_i), \ i \in [1, m] \tag{1.3}$$

where

$$B(\phi_j, \phi_i) = \int_0^1 p(x) \left(\frac{\mathrm{d}}{\mathrm{d}x} \phi_j(x) \right) \left(\frac{\mathrm{d}}{\mathrm{d}x} \phi_i(x) \right) \mathrm{d}x + \int_0^1 q(x) \left(\frac{\mathrm{d}}{\mathrm{d}x} \phi_j(x) \right) \phi_i(x) \tag{1.4}$$

$$\mathrm{d}x + \int_0^1 r(x) \, \phi_j(x) \, \phi_i(x) \, \mathrm{d}x$$

$$\ell(\phi_i) = \int_0^1 f(x) \, \phi_i(x) \, \mathrm{d}x + \beta p(1) \, \phi_i(1) \tag{1.5}$$

• **Assembly: Algebraic system for (1.3) using (1.4) and (1.5).**

Example: (Revisit to the example in Section 11.2) Use the Linear Galerkin FEM to approximate the solution of

$$-y'' + \frac{1}{x}y' + \frac{3}{x^2}y = 4x^2, \quad 1 \le x \le 2,$$
$$y(1) = 0, \; y'(2) = -2$$

with $h = 1/5$ and $h = 1/10$.

Solution:

```
a := 1 : b := 2 :
p := x -> 1 : p(x) :
q := x -> 1/x : q(x) :
r := x -> 3/x^2 : r(x) :
f := x -> 4*x^2 : f(x) :
α := 0 : β := -2 :

# The Linear Galerkin FEM
# with Lagrange form of linear basis functions,
# and exact evaluation of integrals.

u := Array(1..2) :

for it from 1 to 2 do
    # Partitioning
    m := 2^(it-1)·5;
    h := (b - a)/m :
    for i from 0 to m do xx_i := a + i·h; end do;
    A  := Matrix(m + 1, m + 1);
    bb := Vector(m + 1) :

    # Element-wise Assembly, with exact evaluation of integrals
    for k from 1 to m do
        g1 := k - 1 :   g2 := k :
```

```
    x1 := xx[k-1] : x2 := xx[k] :
    ph[1] := (x - x2)/(x1 - x2) : ph[2] := (x - x1)/(x2 - x1) :
    for i to 2 do
        for j to 2 do
            A[g1 + i, g1 + j] := A[g1 + i, g1 + j]
                + int(p(x)·diff(ph[j], x)·diff(ph[i], x), x = x1 ..x2)
                + int(q(x)·diff(ph[j], x)·ph[i], x = x1 ..x2)
                + int(r(x)·ph[j]·ph[i], x = x1 ..x2);
        end do;
        bb[g1 + i] := bb[g1 + i]
            + int(f(x)·ph[i], x = x1 ..x2);
    end do;
end do;

# Incorporation of the BCs:
A[1, 1] := 1 : for j from 2 to m + 1 do A[1, j] := 0 : end do;
bb[1] := α;
bb[m + 1] := bb[m + 1] + β·p(1);

# Solve by matrix inversion
u[it] := evalf[15](A)^(-1).bb;
end do:
```

Error Analysis: the linear Galerkin FEM

The exact solution:

$$ey := x \to -\frac{8}{7x} + \frac{68}{35}x^3 - \frac{4}{5}x^4 :$$

```
n := 5 :
h := (b - a) / n :
for k to 5 do
    xk := a + k·h;
    yk := ey(xk);
    err1 := abs(yk-u[1][k + 1]);
    err2 := abs(yk - u[2][2 k + 1]);
    richardson := (4·u[2][2 k + 1] - u[1][k + 1]) /3;
    err3 := abs(yk - richardson);
    if (k = 1) then
        printf(" exact_y(x) u(h=1/5) error u(h=1/10) error
    Richardson error\n");
    end if;
    printf(" x=%4.2f %8.5f | %8.5f %8.5f | %8.5f %8.5f | %8.5f
    %9.4g\n", xk, yk, u[1][k + 1], err1, u[2][2 k + 1], err2, richardson, err3);
end do:
```

	exact_y(x)	u(h=1/5)	error	u(h=1/10)	error	Richardson	error
x=1.20	0.74600	0.74927	0.00328	0.74681	0.00081	0.74598	1.155e-005
x=1.40	1.44159	1.44769	0.00610	1.44311	0.00151	1.44158	1.498e-005
x=1.60	2.00078	2.00842	0.00764	2.00268	0.00190	2.00076	1.256e-005
x=1.80	2.29774	2.30483	0.00709	2.29951	0.00177	2.29774	4.459e-006
x=2.00	2.17143	2.17505	0.00362	2.17234	0.00091	2.17144	9.758e-006

Second-order FDM Results (Section 11.2)

	exact_y(x)	u(h=1/5)	error	u(h=1/10)	error	Richardson	error
x=1.20	0.74600	0.75688	0.01088	0.74871	0.00271	0.74598	1.423e-05
x=1.40	1.44159	1.46883	0.02723	1.44840	0.00680	1.44159	6.646e-06
x=1.60	2.00078	2.04943	0.04865	2.01295	0.01217	2.00078	7.089e-06
x=1.80	2.29774	2.37302	0.07527	2.31658	0.01884	2.29777	2.315e-05
x=2.00	2.17143	2.27883	0.10740	2.19831	0.02688	2.17147	4.068e-05

FEM with Numerical Quadrature

Solution: The Preceding BVP

```
a := 1 : b := 2 :
p := x -> 1 : p(x) :
q := x -> 1/x : q(x) :
r := x -> 3/x^2 : r(x) :
f := x -> 4 x^2 : f(x) :
α := 0 : β := -2 :

# The Linear Galerkin FEM

u := Array(1 ..2) :

for it from 1 to 2 do

    # Partitioning
    m := 2^(it - 1)·5;
    h := (b - a)/m :
    for i from 0 to m do xx_i := a + i·h; end do;
    A  := Matrix(m + 1, m + 1);
    bb := Vector(m + 1) :

    # Element-wise Assembly, with Trapezoid Rule

    for k from 1 to m do
        g1 := k - 1 :   g2 := k :
        x1 := xx_(k - 1) :  x2 := xx_k :   hk := x2 - x1;
        ph[1] := (x - x2)/(x1 - x2) :   ph[2] := (x - x1)/(x2 - x1) :
        for i to 2 do
            for j to 2 do
                A[g1 + i, g1 + j] := A[g1 + i, g1 + j]
```

```
                + hk/2 · (eval(p(x)·diff(ph[j], x)·diff(ph[i], x), x = x1)
                        + eval(p(x)·diff(ph[j], x)·diff(ph[i], x), x = x2)
                        + eval(q(x)·diff(ph[j], x)·ph[i], x = x1)
                        + eval(q(x)·diff(ph[j], x)·ph[i], x = x2)
                        + eval(r(x)·ph[j]·ph[i], x = x1)
                        + eval(r(x)·ph[j]·ph[i], x = x2)   );
              #+int(p(x)·diff(ph[j], x)·diff(ph[i], x), x = x1 ..x2)
              #+int(q(x)·diff(ph[j], x)·ph[i], x = x1 ..x2)
              #+int(r(x)·ph[j]·ph[i], x = x1 ..x2);
        end do;
        bb[g1 + i] := bb[g1 + i]
            + hk/2 · (eval(f(x)·ph[i], x = x1)
                    + eval(f(x)·ph[i], x = x2) );
          #+int(f(x)·ph[i], x = x1 ..x2);
    end do;
end do;

# Incorporation of the BCs:
A[1, 1] := 1 :  for j from 2 to m + 1 do A[1, j] := 0 : end do;
bb[1] := α;
bb[m + 1] := bb[m + 1] + β·p(1);

# Solve by matrix inversion
u[it] := evalf15(A)^-1.bb;
end do:
```

Error Analysis: the linear Galerkin FEM,with Trapezoid Rule

```
n := 5 :
h := (b − a) / n :
for k to 5 do
    xk := a + k·h;
    yk := ey(xk);
    err1 := abs(yk-u[1][k + 1]);
    err2 := abs(yk − u[2][2 k + 1]);
    richardson := (4·u[2][2 k + 1] − u[1][k + 1]) /3;
    err3 := abs(yk − richardson);
    if (k = 1) then
        printf(" exact_y(x)  u(h=1/5)  error  u(h=1/10)  error
    Richardson  error\n");
    end if;
    printf(" x=%4.2f  %8.5f | %8.5f  %8.5f | %8.5f  %8.5f | %8.5f
    %9.4g\n", xk, yk, u[1][k + 1], err1, u[2][2 k + 1], err2, richardson, err3);
end do:
```

	exact_y(x)	u(h=1/5)	error	u(h=1/10)	error	Richardson	error
x=1.20	0.74600	0.75132	0.00532	0.74731	0.00131	0.74597	2.78e-005
x=1.40	1.44159	1.45619	0.01460	1.44522	0.00363	1.44156	3.185e-005
x=1.60	2.00078	2.02780	0.02703	2.00751	0.00673	2.00075	2.977e-005
x=1.80	2.29774	2.34011	0.04237	2.30831	0.01057	2.29772	2.632e-005
x=2.00	2.17143	2.23203	0.06060	2.18656	0.01513	2.17141	2.291e-005

FEM Results with Exact Integrals

	exact_y(x)	u(h=1/5)	error	u(h=1/10)	error	Richardson	error
x=1.20	0.74600	0.74927	0.00328	0.74681	0.00081	0.74598	1.155e-005
x=1.40	1.44159	1.44769	0.00610	1.44311	0.00151	1.44158	1.498e-005
x=1.60	2.00078	2.00842	0.00764	2.00268	0.00190	2.00076	1.256e-005
x=1.80	2.29774	2.30483	0.00709	2.29951	0.00177	2.29774	4.459e-006
x=2.00	2.17143	2.17505	0.00362	2.17234	0.00091	2.17144	9.758e-006

Second-order FDM Results (Section 11.2)

	exact_y(x)	u(h=1/5)	error	u(h=1/10)	error	Richardson	error
x=1.20	0.74600	0.75688	0.01088	0.74871	0.00271	0.74598	1.423e-05
x=1.40	1.44159	1.46883	0.02723	1.44840	0.00680	1.44159	6.646e-06
x=1.60	2.00078	2.04943	0.04865	2.01295	0.01217	2.00078	7.089e-06
x=1.80	2.29774	2.37302	0.07527	2.31658	0.01884	2.29777	2.315e-05
x=2.00	2.17143	2.27883	0.10740	2.19831	0.02688	2.17147	4.068e-05

Empty Page

11.5. Finite Difference Methods for Non-constant Diffusion

We in this section consider differential equations of the following form

$$-(p(x)\,y')' + q(x)\,y = f(x), \quad a \le x \le b,$$
$$y(a) = \alpha, \ \frac{\mathrm{d}}{\mathrm{d}x}\,y(x=b) = \beta \qquad \textbf{(1)}$$

where $q(x) \ge 0$.

As in Section 11.2, let, for some $n > 0$,

$$h = \frac{(b-a)}{n}, \quad x_i = a + i \cdot h \quad (0 \le i \le n)$$

and define $g_i = g(x_i)$ for $g = y, p, q, f$.

Review of FDMs:

$$y'(x_i) = \frac{y_{i+1} - y_{i-1}}{2h} - \frac{h^2}{3!}y'''(x_i) + \mathrm{O}(h^4)$$
$$y''(x_i) = \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} - \frac{h^2}{12}y^{(4)}(x_i) + \mathrm{O}(h^4)$$

FDM for $(p(x)\,y')'$

Claim:

$$(p\,y_x)_x(x_i) = \frac{p_{i-\frac{1}{2}}\,y_{i-1} - \left(p_{i-\frac{1}{2}} + p_{i+\frac{1}{2}}\right)y_i + p_{i+\frac{1}{2}}\,y_{i+1}}{h^2} + O(h^2) \qquad (2)$$

where $p_{i\pm 1/2} = p\left(\frac{x_i + x_{i\pm 1}}{2}\right)$

Proof:

Let $g = p\,y_x$. Then

$$(p\,y_x)_x(x_i) = g_x(x_i) = \frac{g(x_{i+1/2}) - g(x_{i-1/2})}{h} + O(h^2)$$

Now, we should approximate $g(x_{i+1/2})$ and $g(x_{i-1/2})$. For example,

$$\begin{aligned} g(x_{i+1/2}) &= p_{i+1/2}\,y_x(x_{i+1/2}) \\ &= p(x_{i+1/2})\cdot\left(\frac{y_{i+1}-y_i}{h} - \frac{y_{xxx}(x_{i+1/2})}{3!\cdot 2^2}h^2 - \frac{y_{xxxxx}(x_{i+1/2})}{5!\cdot 2^4}h^4 - \ldots\right) \\ &= p_{i+1/2}\,\frac{y_{i+1}-y_i}{h} + p_{i+1/2}\left(-\frac{y_{xxx}(x_{i+1/2})}{3!\cdot 2^2}h^2 - \frac{y_{xxxxx}(x_{i+1/2})}{5!\cdot 2^4}h^4 - \ldots\right) \end{aligned}$$

Let

$$K(x) = p(x)\cdot\left(-\frac{y_{xxx}(x)}{3!\cdot 2^2}h^2 - \frac{y_{xxxxx}(x)}{5!\cdot 2^4}h^4 - \ldots\right).$$

Then, we have

$$g(x_{i+1/2}) = p_{i+1/2}\cdot\frac{y_{i+1}-y_i}{h} + K(x_{i+1/2})$$

Similarly,

$$g(x_{i-1/2}) = p(x_{i-1/2})\cdot\frac{y_i - y_{i-1}}{h} + K(x_{i-1/2})$$

Therefore, combining above equations reads

$$\left(p\,y_x\right)_x\left(x_i\right) = \frac{p_{i-1/2}\,y_{i-1} - \left(p_{i-1/2} + p_{i+1/2}\right)y_i + p_{i+1/2}\,y_{i+1}}{h^2} + \frac{K\left(x_{i+1/2}\right) - K\left(x_{i-1/2}\right)}{h} + \mathrm{O}\left(h^2\right)$$

Since

$$\frac{K\left(x_{i+1/2}\right) - K\left(x_{i-1/2}\right)}{h} = \frac{K'(\xi)h}{h} = K'(\xi) = \mathrm{O}\left(h^2\right)$$

we conclude that the given difference formula is in the second-order accuracy.

Assembly

Use of these central FD schemes in the first equation of Equation **(1)** at $x = x_i$ results in the equation

$$\frac{-p_{i-\frac{1}{2}}\,y_{i-1} + \left(p_{i-\frac{1}{2}} + p_{i+\frac{1}{2}}\right) y_i - p_{i+\frac{1}{2}}\,y_{i+1}}{h^2} + q_i y_i = f_i + \mathrm{O}\left(h^2\right) \qquad \textbf{(3)}$$

From the boundary conditions of **(1)**, we have

$$y_0 = \alpha \qquad \textbf{(4)}$$

and

$$\frac{1}{2}\,\frac{y_{n+1} - y_{n-1}}{h} = \beta + \frac{1}{6}\,h^2\,\mathrm{D}^{(3)}(y)\left(x_n\right) + \mathrm{O}\left(h^4\right) \qquad \textbf{(5)}$$

Equation **(5)** can be written as

$$\left(y_{n+1}\right) \approx \left(y_{n-1} + 2\,h\beta\right) \qquad \textbf{(6)}$$

Combining Equations **(3)**, **(4)**, and **(6)**, the differential equation **(1)** is discretized as

$$\left(p_{1-1/2} + p_{1+1/2} + h^2 q_1\right) y_1 - p_{1+1/2}\,y_2 = h^2 f_1 + p_{1-1/2}\,\alpha$$

$$-p_{i-1/2}\,y_{i-1} + \left(p_{i-1/2} + p_{i+1/2} + h^2 q_i\right) y_i - p_{i+1/2}\,y_{i+1} = h^2 f_i \qquad (2 \le i \le n-1)$$

$$-\left(p_{n-1/2} + p_{n+1/2}\right) y_{n-1} + \left(p_{n-1/2} + p_{n+1/2} + h^2\,q_n\right) y_n = h^2 f_n + 2\,h\,\beta p_{n+1/2}$$

Then, the resulting system of equations is expressed in the following $n \times n$ tridiagonal matrix form

$$A\,u = b \tag{7}$$

where, defining $pp_i := p_{i-1/2} + p_{i+1/2}$,

$$A = \begin{bmatrix} pp_1 + h^2 q_1 & -p_{1+1/2} & 0 & \cdots & 0 \\ -p_{2-1/2} & pp_2 + h^2 q_2 & -p_{2+1/2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -p_{n-3/2} & pp_{n-1} + h^2 q_{n-1} & -p_{n-1/2} \\ 0 & \cdots & 0 & -pp_n & pp_n + h^2 q_n \end{bmatrix}$$

and

$$u := \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}, \qquad b = \begin{bmatrix} h^2 f_1 \\ h^2 f_1 \\ \vdots \\ h^2 f_{n-1} \\ h^2 f_n \end{bmatrix} + \begin{bmatrix} p_{1-1/2}\alpha \\ 0 \\ \vdots \\ 0 \\ 2\,h\,\beta\,p_{n+1/2} \end{bmatrix}$$

Example: (Revisit) Use the second-order FDM to approximate the solution of

$$-y''+\frac{1}{x}y'+\frac{3}{x^2}y=4x^2,\quad 1\le x\le 2,$$
$$y(1)=0,\ y_x(2)=-2 \qquad \textbf{(8)}$$

Solution:

By dividing both sides of the differential equation by x, we have

$$-\frac{1}{x}y''+\frac{1}{x^2}y'+\frac{3}{x^3}y=4x$$

Thus, DE **(8)** equivalently reads

$$-\left(\frac{1}{x}y'\right)'+\frac{3}{x^3}y=4x$$
$$y(1)=0,\ y_x(2)=-2 \qquad \textbf{(3.1)}$$

Thus,

```
a := 1 : b := 2 :
al := 0 : be := -2 :
p := x -> 1/x : p(x) :
q := x -> 3/x^3 : q(x) :
f := x -> 4*x : f(x) :
ey := x -> -8/(7*x) + 68/35*x^3 - 4/5*x^4 :
u := Array(1..2) :

for it from 1 to 2 do
    n := 2^(it-1)*5;
    h := (b-a)/n :
    A := Matrix(n, n);
    bb := Vector(n) :

    k := 1 :
        xk := a + k*h;
        pl := p(xk - h/2); pr := p(xk + h/2);
```

```
    A[k, k] := pl + pr + h^2·q(xk);
    A[k, k + 1] := -pr;
    bb[k] := h^2·f(xk) + pl·al;
  for k from 2 to n - 1 do
    xk := a + k·h;
    pl := p(xk - h/2); pr := p(xk + h/2);
    A[k, k] := pl + pr + h^2·q(xk);
    A[k, k - 1] := -pl;
    A[k, k + 1] := -pr;
    bb[k] := h^2·f(xk);
  end do;
   k := n :
    xk := a + k·h;
    pl := p(xk - h/2); pr := p(xk + h/2);
    A[k, k] := pl + pr + h^2·q(xk);
    A[k, k-1] := -(pl + pr);
    bb[k] := h^2·f(xk) + pr·2 h be;

  u[it] := A^-1.bb;
end do:
evalf(u[1]^%T)
```

$$\left[\begin{array}{ccccc} 0.7538642304 & 1.463251872 & 2.041761984 & 2.363896095 & 2.269144027 \end{array}\right] \quad \textbf{(3.2)}$$

```
evalf(u[2]^%T)
```

$$[0.3764668394, 0.7479458197, 1.107958783, 1.446997149, 1.752888380, 2.011026465, 2.204521797, 2.314301495, 2.319178279, 2.195898805] \quad \textbf{(3.3)}$$

Error analysis:

```
n := 5 :
h := (b - a) / n :
for k to 5 do
  xk := a + k·h;
  yk := ey(xk);
```

```
    err1 := abs(yk-u[1][k]);
    err2 := abs(yk-u[2][2k]);
    richardson := (4·u[2][2k]-u[1][k])/3;
    err3 := abs(yk-richardson);
    if (k=1) then
      printf("          exact_y(x)    u(h=1/5)  error       u(h=1/10)  error
    Richardson  error\n");
    end if;
    printf(" x=%4.2f  %8.5f  |%8.5f  %8.5f  |%8.5f  %8.5f  |%8.5f
    %9.4g\n", xk, yk, u[1][k], err1, u[2][2k], err2, richardson, err3);
end do:
```

	exact_y(x)	u(h=1/5)	error	u(h=1/10)	error	Richardson	error
x=1.20	0.74600	0.75386	0.00787	0.74795	0.00195	0.74597	2.317e-05
x=1.40	1.44159	1.46325	0.02166	1.44700	0.00540	1.44158	1.456e-05
x=1.60	2.00078	2.04176	0.04098	2.01103	0.01025	2.00078	4.150e-06
x=1.80	2.29774	2.36390	0.06615	2.31430	0.01656	2.29777	2.774e-05
x=2.00	2.17143	2.26914	0.09772	2.19590	0.02447	2.17148	5.516e-05

FDM Solutions for (8), computed in Section 11.2

	exact_y(x)	u(h=1/5)	error	u(h=1/10)	error	Richardson	error
x=1.20	0.74600	0.75688	0.01088	0.74871	0.00271	0.74598	1.423e-05
x=1.40	1.44159	1.46883	0.02723	1.44840	0.00680	1.44159	6.646e-06
x=1.60	2.00078	2.04943	0.04865	2.01295	0.01217	2.00078	7.089e-06
x=1.80	2.29774	2.37302	0.07527	2.31658	0.01884	2.29777	2.315e-05
x=2.00	2.17143	2.27883	0.10740	2.19831	0.02688	2.17147	4.068e-05

Note: The differential equation

$$-\left(p\,y_x\right)_x + qy = f \tag{9}$$

reads equivalently

$$-p\,y_{xx} - p_x y_x + qy = f \tag{10}$$

However, it is often the case that the numerical solution of (9) is much more accurate than that of (10), particularly when the diffusion function $p(x)$ is oscillatory or non-smooth. When an FEM is applied, you should not use DE of the form (10) when (9) is available.

FDM vs. FEM, for Non-constant Diffusion

Recall:

The **FDM** is expressed as

$$-\left(p\,y_x\right)_x\left(x_i\right) = \frac{-p_{i-1/2}\,y_{i-1} + \left(p_{i-1/2}+p_{i+1/2}\right)y_i - p_{i+1/2}\,y_{i+1}}{h^2} + \mathrm{O}\left(h^2\right)$$

with $p_{i\pm 1/2} = p\left(\frac{x_i + x_{i\pm 1}}{2}\right)$

From the **linear Galerkin FEM,** the coefficients and the right-hand side are

$$a_{i,i-1} = -\frac{p_{i-1}+p_i}{2\,h_i},\quad a_{i,i} = \frac{p_{i-1}+p_i}{2\,h_i} + \frac{p_i+p_{i+1}}{2\,h_{i+1}},\quad a_{i,i+1} = -\frac{p_i+p_{i+1}}{2\,h_{i+1}}$$

$$b_i = f_i\,\frac{h_i + h_{i+1}}{2}$$

Thus, when $h := h_i$ for all i (uniform grid), the algebraic equation becomes

$$\frac{-\frac{p_{i-1}+p_i}{2}\,y_{i-1} + \left(\frac{p_{i-1}+p_i}{2} + \frac{p_i+p_{i+1}}{2}\right)y_i - \frac{p_i+p_{i+1}}{2}\,y_{i+1}}{h^2} = f_i$$

Conclusions:

The FDM is equivalent to the linear Galerkin FEM when

$$p_{i\pm 1/2} \approx \frac{p\left(x_i\right) + p\left(x_{i\pm 1}\right)}{2}$$

With the approximation (as in realistic simulations), the FDM solution is still in a second-order accuracy.

Empty Page

Homework:
11. Boundary-Value Problems of One Variable

#1. Given a BVP:

$$y'' = y' + 2y + \cos(x),\ 0 \le x \le \pi/2,$$
$$y(0) = -0.3,\ y(\pi/2) = -0.1$$

a. Show that the BVP has a unique solution.
b. Use the **shooting method (and RK4SYSTEM)** to approximate the solution,

with $h = \pi/10$ and $h = \pi/20$.

c. The exact solution $y(x) = -\frac{1}{10}(\sin(x) + 3\cos(x))$. Measure the errors for the approximate solutions computed in (b) and compare them.

#2. The second-order differential equation

$$y'' = -\frac{3}{x}y' + \frac{3}{x^2}y - \frac{\ln(x)}{x^2},\quad 1 \le x \le 3,$$
$$y(1) = 0,\ y'(3) = 0$$

has the exact solution $y(x) = -\frac{3}{28x^3} - \frac{29}{252}x + \frac{1}{3}\ln(x) + \frac{2}{9}$.

a. Use the **second-order FDM** to approximate the solution of y with $n = 10, 20, 40$, that is, with

$$h = \frac{2}{10}, \frac{2}{20}, \frac{2}{40}$$

b. Apply the Richardson's extrapolation method to get two sets of fourth-order approximations of the solution, one from $u_{n=10}$ and $u_{n=20}$ (say, R_{20}) and the other from $u_{n=20}$ and $u_{n=40}$ (say, R_{40}).
c. Verify if the Richardson's extrapolation really produces the desired theoretical accuracy. This requires you to measure the errors for R_{20} and R_{40} and check if the error for R_{40} is sixteen times smaller than the error for R_{20}.

#3. Use the **linear Rayleigh-Ritz-Galerkin FEM** to approximate the solution to the boundary-value problem

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(\mathrm{e}^{x}\frac{\mathrm{d}}{\mathrm{d}x}u\right)+\mathrm{e}^{x}u=(2-x)\,\mathrm{e}^{x}+x,\quad 0\le x\le 1,$$

$$u(0)=0,\ u(1)=0$$

using equally distributed nodal points with $h = 1/10$.

a. Carry out the integrations exactly and using Trapezoidal rule.

b. Compare both of your results with the actual solution

$u(x)=(x-1)\left(\mathrm{e}^{-x}-1\right)$.

#4.(Optional (and extra credit) for undergraduates). For the boundary-value problem considered in the preceding problem, use the **method of collocation** to approximate the solution.

a. Choose a set of three basis functions in your own. (The basis functions should be linearly independent each other and satisfy the boundary condition.)

b. Select the nodal points inside $[0,\ 1]$ conveniently, again in your own.

c. Compare your result with the actual solution.

12. Numerical Solutions to Partial Differential Equations

In This Chapter:

Topics	Concerns
FDM: Parabolic PDEs (1D)	Heat equation
Forward/Backward Euler method	
Crank-Nicolson method	Error=O$\left(\Delta x^2 + \Delta t^2\right)$
θ-method	Stability analysis
FDM: Parabolic PDEs (2D/3D)	Rectangular domain
Grid selection	
Point ordering	
Alternating direction implicit (ADI) method	
FEM: Elliptic PDEs (2D)	
Variational formulation	Weak form
Rayleigh-Ritz-Galerkin method	
Galerkin method	
Algebraic solvers	
Hyperbolic PDEs (1D)	Wave equation

12.1. Partial Differential Equations (PDEs)

General form of PDEs: For $\nabla = \left(\partial_x, \partial_y\right)^T$,

$$\rho u_t + s^2 u_{tt} - \nabla \cdot (K \nabla u) + v \cdot \nabla u + r u = f, \quad (x, t) \in \Omega \times [0, T]$$
$$\text{BC} + \text{IC}$$

Elliptic PDEs: Equilibrium (time-independent)

$$-\nabla \cdot (K \nabla u) + v \cdot \nabla u + r u = f$$

Examples:

Poisson equation: $-u_{xx} - u_{yy} = f$

Laplace equation: $-u_{xx} - u_{yy} = 0$

Parabolic PDEs: Diffusion-reaction

$$\rho u_t - \nabla \cdot (K \nabla u) + r u = f$$

Example:

Heat equation: $c\rho u_t - \nabla \cdot (K \nabla u) = f$

Hyperbolic PDEs: Convection/vibration

$$u_t + v u_x = 0 \quad \textbf{or} \quad s^2 u_{tt} - \nabla \cdot (K \nabla u) = f$$

Example:

Wave equation: $\frac{1}{v^2} u_{tt} + a u_t - \nabla \cdot (K \nabla u) = f$

Parabolic Partial Differential Equations (1D)

The parabolic PDE in one spatial variable we will consider is the heat equation of the form

$$\frac{\partial}{\partial t} u(x, t) - \alpha^2 \frac{\partial^2}{\partial x^2} u(x, t) = f(x, t), \quad (x, t) \in [a, b] \times [0, T]$$
$$u(a, t) = 0, u(b, t) = 0, u(x, 0) = u_0(x) \tag{1}$$

Partitioning:

Let $h = (b - a)/m$, for some $m > 0$, and

$$x_j = a + j h, j \in [0, m] \tag{2}$$

Similarly, $k = T/n_t$, for some $n_t > 0$, and

$$t^n = n k, n \in [0, n_t] \tag{3}$$

Spatial Discretization:

Define $u_j^n = u\left(x_j, t^n\right)$.

Recall the 2nd-order central FD Schemes:

$$u_x(x_j) = \frac{1}{2}\frac{u_{j+1} - u_{j-1}}{h} - \frac{1}{6}h^2 u_{xxx}(x_j) + \mathrm{O}(h^4) \tag{4}$$

$$u_{xx}(x_j) = \frac{u_{j-1} - 2u_j + u_{j+1}}{h^2} - \frac{1}{12}h^2 u_{xxxx}(x_j) + \mathrm{O}(h^4) \tag{5}$$

Then, the differential equation in **(1)** is approximated as

$$u_t(x_j, t) + \frac{\alpha^2\left(-u_{j-1}(t) + 2u_j(t) - u_{j+1}(t)\right)}{h^2} = f(x_j, t) - \frac{1}{12}h^2 u_{xxxx}(x_j, t) + \mathrm{O}(h^4), j \in [0, m] \tag{6}$$

Incorporated with the boundary conditions in **(1)**, Equation **(6)** is written in a vector form

$$u_t(t) + Au(t) = f(t) - \frac{1}{12}h^2 u_{xxxx}(t) + \mathrm{O}(h^4), t \in [0, T] \tag{7}$$

Notes:

- Equation **(7)** is called a **semi-discrete problem.**
- It is a system of ordinary differential equations. Depending on how the time-derivative is approximated, various methods can be formulated.

Forward Euler Method (Explicit)

Expanding u in a Taylor polynomial about $t = t^n$ evaluated at $t^{n+1} = t^n + k$, we have

$$u^{n+1} = u^n + k\,u_t(t^n) + \frac{1}{2}k^2\,u_{tt}(t^n) + \frac{1}{6}k^3\,u_{ttt}(t^n) + \frac{1}{24}k^4\,u_{tttt}(\xi_n) \qquad \textbf{(8)}$$

which in turn implies that

$$u_t(t^n) = \frac{u^{n+1} - u^n}{k} - \frac{1}{2}k\,u_{tt}(t^n) - \frac{1}{6}k^2\,u_{ttt}(t^n) - \frac{1}{24}k^3\,u_{tttt}(\xi_n) \qquad \textbf{(9)}$$

Then, the **Forward Euler method** is formulated from **(7)** and **(9)** as

$$\frac{u^{n+1} - u^n}{k} + A\,u^n = f^n - \frac{1}{12}h^2\,u_{xxxx}(t^n) + \frac{1}{2}k\,u_{tt}(t^n) + \mathrm{O}(h^4 + k^2) \qquad \textbf{(10)}$$

When the error terms are ignored, then the above equation reads

$$\frac{w^{n+1} - w^n}{k} + A\,w^n = f^n \qquad \textbf{(11)}$$

where $w^n = \left(w_0^n,\, w_1^n, \bullet\bullet\bullet,\, w_m^n\right)^T$ is an approximation of $u^n = \left(u_0^n,\, u_1^n, \bullet\bullet\bullet,\, u_m^n\right)^T$.

Forward Euler method: It follows from **(11)** that

$$w^{n+1} = (1 - kA)\,w^n + kf^n \qquad \textbf{(12)}$$

for which

$$TruncationError = \mathrm{O}(h^2 + k) \qquad \textbf{(13)}$$

Stability Analysis

Let $f \equiv 0$. Then each component of the Forward Euler method **(12)** reads

$$w_j^{n+1} = \left(1 - 2 \cdot \frac{\alpha^2 k}{h^2}\right) w_j^n + \frac{\alpha^2 k}{h^2} w_{j-1}^n + \frac{\alpha^2 k}{h^2} w_{j+1}^n$$

Letting $\mu = \frac{\alpha^2 k}{h^2}$, we have

$$w_j^{(n+1)} = (1 - 2\mu) w_j^n + \mu w_{j-1}^n + \mu w_{j+1}^n \qquad \textbf{(14)}$$

Theorem: Suppose that h and k be chosen such that

$$\mu \le \frac{1}{2} \qquad \textbf{(15)}$$

Then

$$\| w^{n+1} \|_\infty \le \| w^n \|_\infty .$$

Proof:

Let the condition **(15)** be satisfied. Then **each of the coefficients in the right side of (14) is nonnegative and their sum become 1**. Thus,

$$\begin{aligned} |w_j^{n+1}| &\le (1 - 2\mu) |w_j^n| + \mu |w_{j-1}^n| + \mu |w_{j+1}^n| \\ &\le (1 - 2\mu) \| w^n \|_\infty + \mu \| w^n \|_\infty + \mu \| w^n \|_\infty \\ &= \| w^n \|_\infty \qquad (0 \le j \le m) \end{aligned}$$

from which the conclusion follows.

Stability condition:

- The condition in **(15)** is called a stability condition, which equivalently reads

$$k \leq \frac{1}{2}\frac{h^2}{\alpha^2} \tag{6.1}$$

- Thus, the forward Euler method guarantees a bounded (stable) solution when the temporal step-size is selected sufficiently small to become

$$k = \mathrm{O}\left(h^2\right) \tag{6.2}$$

- When the stability condition **(6.1)** is violated, the numerical solution can be easily oscillatory and is not convergent.

- Due to the requirement of choosing a huge number of time steps, the computation may be expensive.

Example: Use the **forward Euler method** with (a) $h = 1/10,\ k = 1/200$ and (b) $h = 1/10,\ k = 1/170$ to approximate the solution to

$$\frac{\partial}{\partial t}u(x, t) - \frac{\partial^2}{\partial x^2}u(x, t) = 0,\ (x, t) \in [0, 1] \times [0, 1]$$

$$u(0, t) = 0,\ u(1, t) = 0,\ u(x, 0) = \sin(\pi x) \qquad \textbf{(16)}$$

for which the actual solution is $u(x, t) = e^{-\pi^2 t}\sin(\pi x)$.

Solution:

```
maxerror1D := proc(a, b, nx, t0, u, w)
    local j, h, uxj, maxerr;
    h := (b - a)/nx;
    maxerr := 0;
    for j from 0 to nx do
        uxj := eval(u, [x = a + j·h, t = t0]);
        maxerr := max(maxerr, abs(uxj - w[j + 1]));
    end do;
    return maxerr;
end proc:

ForwardEuler := proc(a, b, T, nx, nt, α, f, u0)
    local j, n, h, k, xj, tn, id1, id2, mu, w, wT;
    with(LinearAlgebra) :
    h := (b - a)/nx;
    k := T/nt;
    mu := α^2·k/h^2;
    print(`mu=`, mu);

    w := Matrix(nx + 1, 2);
    wT := Vector(nx + 1);
    for j to nx + 1 do
        w[j, 1] := eval(u0, x = a + (j - 1)·h);
    end do;
```

```
    for n from 1 to nt do
        tn := (n - 1)·k;
        id1 := modp(n + 1, 2) + 1;
        id2 := modp(n, 2) + 1;
        for j from 2 to nx do
            xj := a + (j - 1)·h;
            w[j, id2] := (1 - 2·mu)·w[j, id1]
                         + mu·(w[j - 1, id1] + w[j + 1, id1])
                         + k·eval(f, [x = xj, t = tn]);
        end do;
    end do;
    for j to nx + 1 do wT[j] := w[j, id2]; end do;
    return wT;
end proc:

a := 0 : b := 1 : T := 1 : α := 1 : f := 0 :
u := exp(-π^2 t)·sin(π x) :
u0 := eval(u, t = 0) :
u0 = sin(π x)
nx := 10 :
### (a):
nt := 200 :
w := ForwardEuler(a, b, T, nx, nt, α, f, u0) :
```

$$mu = \frac{1}{2} \tag{7.1}$$

```
evalf(maxerror1D(a, b, nx, T, u, w));
```

$$0.00000794425945 \tag{7.2}$$

(b) **When the stability condition is slightly violated:**

$nt := 170$:

$w := ForwardEuler(a, b, T, nx, nt, \alpha, f, u0)$:

$$mu=, \frac{10}{17} \tag{7.3}$$

$print(seq(evalf(w[j]), j = 1 ..nx + 1))$;

$$0., -1.1\,10^{9}, 1.\,10^{9}, -1.\,10^{9}, 1.3\,10^{9}, -9.\,10^{8}, 1.3\,10^{9}, -1.\,10^{9}, 1.\,10^{9}, -1.1\,10^{9}, 0. \tag{7.4}$$

Backward Euler Method (Implicit)

Expanding u in a Taylor polynomial about $t = t^n$ evaluated at $t^{n-1} = t^n - k$, we have

$$u^{n-1} = u^n - k\,u_t(t^n) + \frac{1}{2}k^2 u_{tt}(t^n) - \frac{1}{6}k^3 u_{ttt}(t^n) + \frac{1}{24}k^4 u_{tttt}(\xi) \qquad \textbf{(17)}$$

which in turn implies that

$$u_t(t^n) = \frac{u^n - u^{n-1}}{k} + \frac{1}{2}k\,u_{tt}(t^n) - \frac{1}{6}k^2 u_{ttt}(t^n) + \frac{1}{24}k^3 u_{tttt}(\xi) \qquad \textbf{(18)}$$

Then, the **Backward Euler method** is formulated from **(7)** and **(18)** as

$$\frac{u^n - u^{n-1}}{k} + A\,u^n = f^n - \frac{1}{12}h^2 u_{xxxx}(t^n) - \frac{1}{2}k\,u_{tt}(t^n) + \mathrm{O}(h^4 + k^2) \qquad \textbf{(19)}$$

When the error terms are ignored, then the above equation reads

$$\frac{w^n - w^{n-1}}{k} + A\,w^n = f^n \qquad \textbf{(20)}$$

where $w^n = \left(w_0^n,\, w_1^n, \cdots,\, w_m^n\right)^T$ is an approximation of $u^n = \left(u_0^n,\, u_1^n, \cdots,\, u_m^n\right)^T$.

Backward Euler method: It follows from **(20)** that

$$(1 + kA)\,w^n = w^{n-1} + kf^n \qquad \textbf{(21)}$$

for which

$$TruncationError = \mathrm{O}(h^2 + k) \qquad \textbf{(22)}$$

Stability Analysis

Let $f \equiv 0$. Then each component of the backward Euler method **(21)** reads

$$\left(1 + 2 \cdot \frac{\alpha^2 k}{h^2}\right) w_j^n - \frac{\alpha^2 k}{h^2} w_{j-1}^n - \frac{\alpha^2 k}{h^2} w_{j+1}^n = w_j^{n-1}$$

Letting $\mu = \dfrac{\alpha^2 k}{h^2}$, we have

$$(1 + 2\mu) w_j^n - \mu w_{j-1}^n - \mu w_{j+1}^n = w_j^{(n-1)} \qquad \textbf{(23)}$$

Theorem: For any choice of h and k, the approximate solutions produced by the backward Euler method satisfy

$$\left\| w^n \right\|_\infty \le \left\| w^{n-1} \right\|_\infty .$$

Hence, the backward Euler method is unconditionally stable.

Proof:

Equation **(23)** gives

$$\begin{aligned} (1 + 2\mu)\left|w_j^n\right| &= \left|\mu w_{j-1}^n + \mu w_{j+1}^n + w_j^{n-1}\right| \\ &\le \mu \left|w_{j-1}^n\right| + \mu \left|w_{j+1}^n\right| + \left|w_j^{n-1}\right| \\ &\le \mu \left\| w^n \right\|_\infty + \mu \left\| w^n \right\|_\infty + \left\| w^{n-1} \right\|_\infty \end{aligned}$$

which implies that

$$(1 + 2\mu)\left\| w^n \right\|_\infty \le \mu \left\| w^n \right\|_\infty + \mu \left\| w^n \right\|_\infty + \left\| w^{n-1} \right\|_\infty$$

from which the conclusion follows.

Example: Use the **backward Euler method** with $h = 1/10$ and with (a) $k = 1/200$, (b) $k = 1/100$, and (c) $k = 1/50$ to approximate the solution to

$$\frac{\partial}{\partial t}u(x, t) - \frac{\partial^2}{\partial x^2}u(x, t) = 0, \quad (x, t) \in [0, 1] \times [0, 1]$$

$$u(0, t) = 0, \; u(1, t) = 0, \; u(x, 0) = \sin(\pi x) \tag{24}$$

for which the actual solution is $u(x, t) = e^{-\pi^2 t}\sin(\pi x)$.

Solution:

```
TridiagLU := proc(A)
    local i, j, k, n, r, B;
    n := LinearAlgebra[RowDimension](A);
    #B:=A;  ## Do not use this line for Matrix copy
    B := Matrix(n, 3);
    for j to 3 do; for i to n do
        B[i, j] := A[i, j];
    end do; end do;
    for k from 1 to n - 1 do
        r := B[k + 1, 1]/B[k, 2];
        B[k + 1, 1] := r;
        B[k + 1, 2] := B[k + 1, 2] - r·B[k, 3];
    end do;
    return B;
end proc:

TridiagSOLVE := proc(A, b)
    local k, n, x, y;
    n := LinearAlgebra[RowDimension](A);
    x := Vector(n);
    y := Vector(n);
    ## Forward Elimination
    y[1] := b[1];
    for k from 2 to n do
        y[k] := b[k] - A[k, 1]·y[k - 1];
```

```
    end do;
    ## Backward Substitution
    x[n] := y[n]/A[n, 2];
    for k from n − 1 by −1 to 1 do
        x[k] := (y[k] − A[k, 3]·x[k + 1])/A[k, 2];
    end do;
    return x;
end proc:

BackwardEuler := proc(a, b, T, nx, nt, α, f, u0)
    local j, n, h, k, xj, tn, mu, w, r, A, B;
    with(LinearAlgebra) :
    h := (b − a)/nx;
    k := T/nt;
    mu := α^2·k/h^2;
    print(`mu=`, mu);

    w := Vector(nx + 1);
    r := Vector(nx + 1);
    for j to nx + 1 do
        w[j] := eval(u0, x = a + (j − 1)·h);
    end do;

    A := Matrix(nx + 1, 3);
    A[1, 2] := 1 :
    A[nx + 1, 2] := 1 :
    for j from 2 to nx do
        A[j, 1] := −mu; A[j, 2] := 1 + 2 mu; A[j, 3] := −mu;
    end do;
    B := TridiagLU(A);

    for n from 1 to nt do
        tn := n·k;
        for j from 2 to nx do
            xj := a + (j − 1)·h;
```

```
            r[j] := w[j] + k·eval(f, [x = xj, t = tn]);
        end do;
        w := TridiagSOLVE(B, r);
    end do;
    return w;
end proc:
```

$a := 0 : \ b := 1 : \ T := 1 : \ \alpha := 1 : \ f := 0 :$

$u := \exp\left(-\pi^2 \, t\right) \cdot \sin(\pi x) :$

$u0 := eval(u, t = 0) :$

$u0 = \sin(\pi x)$

$nx := 10 :$

(a)

$nt := 200 :$

$w := BackwardEuler(a, b, T, nx, nt, \alpha, f, u0) :$

$$mu=, \frac{1}{2} \tag{9.1}$$

$evalf(maxerror1D(a, b, nx, T, u, w));$

$$0.00001900330762 \tag{9.2}$$

(b)

$nt := 100$:

$w := BackwardEuler(a, b, T, nx, nt, \alpha, f, u0)$:

$$mu=, 1 \tag{9.3}$$

$evalf(maxerror1D(a, b, nx, T, u, w))$;

$$0.00003622705276 \tag{9.4}$$

(c)

$nt := 50$:

$w := BackwardEuler(a, b, T, nx, nt, \alpha, f, u0)$:

$$mu=, 2 \tag{9.5}$$

$evalf(maxerror1D(a, b, nx, T, u, w))$;

$$0.00007935966322 \tag{9.6}$$

Note: The error is mainly from temporal truncation error $O(k)$.

System of Tridiagonal Matrices

$$A := \begin{bmatrix} d_1 & e_1 & 0 & 0 & 0 & 0 \\ c_2 & d_2 & e_2 & 0 & 0 & 0 \\ 0 & c_3 & d_3 & e_3 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & c_{n-1} & d_{n-1} & e_{n-1} \\ 0 & 0 & 0 & 0 & c_n & d_n \end{bmatrix}$$

Such a matrix can be saved in an $n \times 3$ array:

$$\begin{bmatrix} 0 & d_1 & e_1 \\ c_2 & d_2 & e_2 \\ c_3 & d_3 & e_3 \\ & \bullet\bullet\bullet & \\ c_{n-1} & d_{n-1} & e_{n-1} \\ c_n & d_n & 0 \end{bmatrix}$$

and row operations can be applied correspondingly.

Example: Use Gauss Elimination to solve the tridiagonal system of 5 equations, $A\mathbf{u} = b$, of which the coefficient matrix is given in a 5×3 array for nonzero entries.

$$A := \begin{bmatrix} 0 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \\ -2 & 2 & 0 \end{bmatrix} : \quad b := \begin{bmatrix} -2 \\ -2 \\ -2 \\ -2 \\ 18 \end{bmatrix} :$$

Solution: (The underlined numbers are **pivots**.)

$$Aug := \langle A|b \rangle = \begin{bmatrix} 0 & \underline{\mathbf{2}} & -1 & -2 \\ -1 & 2 & -1 & -2 \\ -1 & 2 & -1 & -2 \\ -1 & 2 & -1 & -2 \\ -2 & 2 & 0 & 18 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 0 & 2 & -1 & -2 \\ 0 & \underline{\mathbf{3/2}} & -1 & -3 \\ -1 & 2 & -1 & -2 \\ -1 & 2 & -1 & -2 \\ -2 & 2 & 0 & 18 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 2 & -1 & -2 \\ 0 & 3/2 & -1 & -3 \\ 0 & \underline{\mathbf{4/3}} & -1 & -4 \\ -1 & 2 & -1 & -2 \\ -2 & 2 & 0 & 18 \end{bmatrix} \rightarrow \cdots$$

$$\rightarrow \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 4 \\ 0 & 1 & 0 & 9 \\ 0 & 1 & 0 & 16 \\ 0 & 1 & 0 & 25 \end{bmatrix}$$

The last column reads the unknown **u**.

Crank-Nicolson Method (Semi-Implicit)

It follows from Equation (7) that

$$u_t\left(t^n + \frac{1}{2}\,k\right) + A\left(\frac{1}{2}\,u^{n+1} + \frac{1}{2}\,u^n\right) = f\left(t^n + \frac{1}{2}\,k\right) + \mathrm{O}\left(h^2 + k^2\right) \qquad \textbf{(25)}$$

The Crank-Nicolson method utilizes a second-order central difference scheme for the temporal derivative in the above equation.

Expanding u in a Taylor polynomial about $t = t^{n+1/2} = t^n + k/2$ evaluated at $t^{n+1} = t^n + k$, we have

$$u^{n+1} = u^{n+1/2} + \left(\frac{k}{2}\right) u_t\left(t^{n+1/2}\right) + \frac{1}{2!}\left(\frac{k}{2}\right)^2 u_{tt}\left(t^{n+1/2}\right) + \frac{1}{3!}\left(\frac{k}{2}\right)^3 u_{ttt}\left(\xi_1\right)$$

Similarly, expanding u in a Taylor polynomial about $t = t^n$ evaluated at $t^{n+1/2}$, we have

$$u^n = u^{n+1/2} - \left(\frac{k}{2}\right) u_t\left(t^{n+1/2}\right) + \frac{1}{2!}\left(\frac{k}{2}\right)^2 u_{tt}\left(t^{n+1/2}\right) - \frac{1}{3!}\left(\frac{k}{2}\right)^3 u_{ttt}\left(\xi_2\right)$$

Subtracting the above two equations gives

$$u^{n+1} - u^n = k\,u_t\left(t^{n+1/2}\right) + \frac{1}{12}k^3 u_{ttt}\left(\xi_3\right)$$

which introduces

$$u_t\left(t^n + \frac{1}{2}\,k\right) = \frac{u^{n+1} - u^n}{k} - \frac{1}{12}\,k^2 u_{ttt}\left(\xi_3\right) \qquad \textbf{(26)}$$

Utilizing (25) and (26), we have

$$\frac{u^{n+1} - u^n}{k} + A\left(\frac{1}{2}\,u^{n+1} + \frac{1}{2}\,u^n\right) = f^{\,n+\frac{1}{2}} + \mathrm{O}\left(h^2 + k^2\right) \qquad \textbf{(27)}$$

When the error terms are ignored, then the above equation reads

$$\frac{w^{n+1}-w^{n}}{k}+A\left(\frac{1}{2}\,w^{n+1}+\frac{1}{2}\,w^{n}\right)=f^{\,n+\frac{1}{2}} \tag{28}$$

where $w^{n}=\left(w_0^n,\, w_1^n,\bullet\bullet\bullet,\, w_m^n\right)^T$ is an approximation of $u^{n}=\left(u_0^n,\, u_1^n,\bullet\bullet\bullet,\, u_m^n\right)^T$.

Crank-Nicolson method: It follows from **(28)** that

$$\left(1+\frac{1}{2}\,kA\right)w^{n+1}=\left(1-\frac{1}{2}\,kA\right)w^{n}+kf^{\,n+\frac{1}{2}} \tag{29}$$

for which

$$TruncationError=\mathrm{O}\left(h^2+k^2\right) \tag{30}$$

The Crank-Nicolson method is called a *semi-implicit* method.

Theorem: For any choice of h and k, the approximate solutions produced by the Crank-Nicolson method satisfy

$$\left\|w^{n}\right\|_2\le\left\|w^{n-1}\right\|_2.$$

Hence, the Crank-Nicolson method is unconditionally stable.

What/Why Stability?

Definition: A numerical algorithm is said to be **consistent** if the truncation error approaches to 0 as $h, k \to 0$, that is,

$$P(u) - P_{h,k}(u) \to 0 \text{ as } h, k \to 0.$$

Definition: A numerical method is **stable** if there is a constant $K > 0$, independent of h and k, such that

$$\left\| w^n \right\|_2 \le K \left\| u_0 \right\|_2, \quad \left(1 \le n \le n_t\right)$$

(Here K may depend on the terminal time T.)

Definition: A numerical algorithm is **convergent** if

$$\left| w_j^n - u\left(x_j, t^n\right) \right| \to 0 \text{ as } h, k \to 0.$$

Rax-Richtmyer Equivalence Theorem: For consistent numerical algorithms, stability is a necessary and sufficient condition for convergence.

Claim: If there is a constant $C \geq 0$ such that

$$\left\| w^n \right\|_2 \leq (1 + C\,k)\, \left\| w^{n-1} \right\|_2$$

Then the method is stable.

Proof. Once the condition is satisfied, we have

$$\left\| w^n \right\|_2 \leq (1 + C\,k)\, \left\| w^{n-1} \right\|_2 \leq (1 + C\,k)^2\, \left\| w^{n-2} \right\|_2$$
$$\leq \cdots \leq (1 + C\,k)^n\, \left\| w^0 \right\|_2$$

Since $k = T/n_t$,

$$(1 + C\,k)^n \leq (1 + C\,k)^{n_t} = \left(1 + \frac{CT}{n_t} \right)^{n_t}$$
$$= \left[\left(1 + \frac{CT}{n_t} \right)^{n_t/CT} \right]^{CT} \nearrow \mathrm{e}^{CT}$$

and therefore

$$\left\| w^n \right\|_2 \leq \mathrm{e}^{CT} \left\| w^0 \right\|_2 = \mathrm{e}^{CT} \left\| u_0 \right\|_2$$

which completes the proof.

Example: Use the **Crank-Nicolson method** with $h = 1/5, 1/10, 1/20, 1/40$ and $k = h/2$ to approximate the solution to

$$\frac{\partial}{\partial t} u(x, t) - \frac{\partial^2}{\partial x^2} u(x, t) = 0, \quad (x, t) \in [0, 1] \times [0, 1]$$

$$u(0, t) = 0, \; u(1, t) = 0, \; u(x, 0) = \sin(\pi x) \tag{31}$$

for which the actual solution is $u(x, t) = e^{-\pi^2 t} \sin(\pi x)$.

Solution

▶ *TridiagLU, TridiagSOLVE, and maxerror1D*

```
CrankNicolson := proc(a, b, T, nx, nt, α, f, u0)
    local j, n, h, k, xj, tn, mu, w, r, A, ALU, B;
    with(LinearAlgebra) :
    h := (b − a)/nx;
    k := T/nt;
    mu := α^2·k/h^2;
    print(`mu=`, mu);

    w := Vector(nx + 1);
    r := Vector(nx + 1);
    for j to nx + 1 do
        w[j] := eval(u0, x = a + (j − 1)·h);
    end do;

    A := Matrix(nx + 1, 3);
    B := Matrix(nx + 1, 3);
    A[1, 2] := 1 : A[nx + 1, 2] := 1 :
    B[1, 2] := 1 : B[nx + 1, 2] := 1 :
    for j from 2 to nx do
        A[j, 1] := −mu/2; A[j, 2] := 1 + mu; A[j, 3] := −mu/2;
        B[j, 1] :=  mu/2; B[j, 2] := 1 − mu;  B[j, 3] := mu/2;
    end do;
```

```
    ALU := TridiagLU(A);

    for n from 1 to nt do
        tn := (n - 1/2)·k;
        for j from 2 to nx do
            xj := a + (j - 1)·h;
            r[j] := B[j, 1]·w[j - 1] + B[j, 2]·w[j]
                    + B[j, 3]·w[j + 1] + k·eval(f, [x = xj, t = tn]);
        end do;
        w := TridiagSOLVE(ALU, r);
    end do;
    return w;
end proc:
```

$$a := 0 : \ b := 1 : \ T := 1 : \ \alpha := 1 : \ f := 0 :$$

$$u := \exp\left(-\pi^2\, t\right)\cdot\sin(\pi x) :$$

$$u0 := eval(u, t = 0) :$$

$$u0$$

$$\sin(\pi x) \tag{11.1}$$

(a)

$$nx := 5 : \ nt := 2\cdot nx :$$

$$w := CrankNicolson(a, b, T, nx, nt, \alpha, f, u0) :$$

$$mu=, \frac{5}{2} \tag{11.2}$$

$$evalf\,(maxerror1D(a, b, nx, T, u, w))$$

$$0.0000200662405 \tag{11.3}$$

(b)

$$nx := 10 : \ nt := 2\cdot nx :$$

$$w := CrankNicolson(a, b, T, nx, nt, \alpha, f, u0) :$$

$$mu=, 5 \tag{11.4}$$

$$evalf\,(maxerror1D(a, b, nx, T, u, w));$$

$$0.000005932829 \tag{11.5}$$

(c)

$nx := 20 : \ nt := 2 \cdot nx :$

$w := CrankNicolson(a, b, T, nx, nt, \alpha, f, u0) :$

$mu=, 10$ **(11.6)**

$evalf(maxerror1D(a, b, nx, T, u, w));$

0.000001525890 **(11.7)**

(d)

$nx := 40 : \ nt := 2 \cdot nx :$

$w := CrankNicolson(a, b, T, nx, nt, \alpha, f, u0) :$

$mu=, 20$

$evalf(maxerror1D(a, b, nx, T, u, w));$

$3.84162833\ 10^{-7}$ **(11.9)**

The θ-Method

The semi-discrete problem for **(1)** is presented in **(7)** and rewritten here:

$$u_t(t) + A\,u(t) = f(t) + \mathrm{O}(h^2),\ t \in [0,\ T] \qquad \textbf{(32)}$$

Consider the following fully discretized problem:

$$\frac{w^{n+1} - w^n}{k} + A\left(\theta\, w^{n+1} + (1-\theta)\, w^n\right) = f^{n+\theta},\ \theta \in [0,\ 1] \qquad \textbf{(33)}$$

Similarly, multiplying the both sides of Equation **(33)** by k and rearranging terms, we obtain the **θ-method**

$$(1 + \theta\, k\, A)\, w^{n+1} = (1 - (1-\theta)\, k\, A)\, w^n + k f^{n+\theta},\ \theta \in [0,\ 1] \qquad \textbf{(34)}$$

which sometimes called the weighted average method.

Three common choices:

$\theta = 0$	$w^{n+1} = (1 - k\,A)\, w^n + k f^n$
$\theta = 1$	$(1 + k\,A)\, w^{n+1} = w^n + k f^{n+1}$
$\theta = \frac{1}{2}$	$\left(1 + \frac{k}{2} A\right) w^{n+1} = \left(1 - \frac{k}{2} A\right) w^n + k f^{n+1/2}$

Notes:

θ	The θ-method becomes
0	Forward Euler method
1	Backward Euler method
$1/2$	Crank-Nicolson method

Stability and Accuracy:

θ	Stability condition	Accuracy
$0 \le \theta < \frac{1}{2}$	$\mu \le \frac{1}{2(1-2\theta)}$	$O(h^2 + k)$
$\frac{1}{2} \le \theta \le 1$	unconditionally stable	$O(h^2 + k^2)$, when $\theta = \frac{1}{2}$

Empty Page

12.2. Parabolic PDEs in Two or Three Variables

A natural generalization of the one-dimensional model problem in two dimensions is the problem

$$u_t - \alpha^2 \nabla^2 u = f, \quad ((x, y), t) \in R \times [0, T]$$

$$u(x, y, 0) = u_0(x, y) \qquad \textbf{(1)}$$

B.C.

where α is a positive constant.

Let the region $R = [a, b] \times [c, d]$ be covered with a uniform rectangular grid of points, with a spacing h_x in the x-direction and h_y in the y-direction, where

$$h_x = \frac{(b-a)}{n_x}, \quad h_y = \frac{(d-c)}{n_y}$$

When the temporal grid size is set to be

$$k = T/n_t,$$

the approximate solution is then denoted by

$$\left(w_{p,q}^n\right) \approx \left(u\left(x_p, y_q, t^n\right)\right) \qquad \textbf{(2)}$$

where

$$x_p = a + p \cdot h_x, \quad p = 0 \,..\, n_x$$

$$y_q = c + q\, h_y, \quad q = 0 \,..\, n_y \qquad \textbf{(3)}$$

$$t^n = n \cdot k, \quad k = 0 \,..\, n_t$$

Semi-discrete Problem

As for the model in one variable, we first discretize the problem for spatial derivatives. For a fixed $t \in [0, T]$,

$$-\alpha^2 u_{xx}(x_p, y_q, t) = \frac{\alpha^2 \left(-u_{p-1,q}(t) + 2\,u_{p,q}(t) - u_{p+1,q}(t) \right)}{h_x^2} + O\left(h_x^2\right) \quad \textbf{(4)}$$

$$-\alpha^2 u_{yy}(x_p, y_q, t) = \frac{\alpha^2 \left(-u_{p,q-1}(t) + 2\,u_{p,q}(t) - u_{p,q+1}(t) \right)}{h_y^2} + O\left(h_y^2\right) \quad \textbf{(5)}$$

Then Equation **(1)** can be written in a vector form

$$u_t(t) + \left(A_1 + A_2\right) u(t) = f(t) + O\left(h_x^2 + h_y^2\right) \quad \textbf{(6)}$$

where

$$\left(A_1\, u(t)\right)_{p,q} = \frac{\alpha^2 \left(-u_{p-1,q}(t) + 2\,u_{p,q}(t) - u_{p+1,q}(t) \right)}{h_x^2} \quad \textbf{(7)}$$

$$\left(A_2\, u(t)\right)_{p,q} = \frac{\alpha^2 \left(-u_{p,q-1}(t) + 2\,u_{p,q}(t) - u_{p,q+1}(t) \right)}{h_y^2} \quad \textbf{(8)}$$

Forward Euler Method (Explicit Scheme)

The simplest explicit difference scheme is the natural extension of the forward Euler method in one variable, and is given by

$$\frac{w^{n+1} - w^n}{k} + A\, w^n = f^n \tag{9}$$

where $A = A_1 + A_2$ and

$$TruncationError = \mathrm{O}\left(h_x^2 + h_y^2 + k\right) \tag{10}$$

Equation **(9)** can be rearranged as

$$w^{n+1} = (1 - kA)\, w^n + k f^n \tag{11}$$

On the other hand, Equation **(9)** reads in its grid-point form

$$\frac{w_{p,q}^{(n+1)} - w_{p,q}^n}{k} + \frac{\alpha^2 \left(-w_{p-1,q}^n + 2\, w_{p,q}^n - w_{p+1,q}^n\right)}{h_x^2} + \frac{\alpha^2 \left(-w_{p,q-1}^n + 2\, w_{p,q}^n - w_{p,q+1}^n\right)}{h_y^2} = f_{p,q}^n \tag{12}$$

Letting

$$\mu_x = \frac{\alpha^2 k}{h_x^2}, \quad \mu_y = \frac{\alpha^2 k}{h_y}$$

we have

$$w_{p,q}^{(n+1)} = \left(1 - 2\,\mu_x - 2\,\mu_y\right) w_{p,q}^n + \mu_x\, w_{p-1,q}^n + \mu_x\, w_{p+1,q}^n + \mu_y\, w_{p,q-1}^n + \mu_y\, w_{p,q+1}^n + k f_{p,q}^n \tag{13}$$

Forward Euler Method: Stability

One can show the following theorem (stability analysis) using the same arguments introduced for the problem in one variable.

Theorem: Let $f \equiv 0$. Suppose that

$$\mu_x + \mu_y \leq \frac{1}{2} \tag{14}$$

Then the forward Euler method is stable and it produces a sequence of solutions satisfying

$$\left\| w^{n+1} \right\|_\infty \leq \left\| w^n \right\|_\infty \text{ and } \left| u^n - w^n \right| = \mathrm{O}\left(h_x^2 + h_y^2 + k \right).$$

Notes

- The forward Euler method requires to choose a sufficiently small k to become stable, which may make the method expensive after all.
- The θ-method can be formulated similarly as for problems in one variable. However, the most interesting case is the Crank-Nicolson method for which $\theta = \frac{1}{2}$, due to its unconditional stability and second-order accuracy in both time and space.

The Crank-Nicolson (CN) Procedure

Ignoring the error term from the semi-discrete problem in **(6)**, the CN time-stepping procedure can be formulated as

$$\frac{w^{n+1}-w^{n}}{k}+A\left(\frac{1}{2}w^{n+1}+\frac{1}{2}w^{n}\right)=f^{n+\frac{1}{2}} \qquad \textbf{(15)}$$

where $A=A_1+A_2$ and $w^{n}=\left(w_0^{n},\, w_1^{n},\bullet\bullet\bullet,\, w_m^{n}\right)^{T}$ is an approximation of $u^{n}=\left(u_0^{n},\, u_1^{n},\bullet\bullet\bullet,\, u_m^{n}\right)^{T}$.

Crank-Nicolson Procedure: It follows from **(15)** that

$$\left(1+\frac{1}{2}kA\right)w^{n+1}=\left(1-\frac{1}{2}kA\right)w^{n}+kf^{n+\frac{1}{2}} \qquad \textbf{(16)}$$

where $A=A_1+A_2$ and

$$TruncationError=\mathrm{O}\left(h_x^2+h_y^2+k^2\right) \qquad \textbf{(17)}$$

Notes:

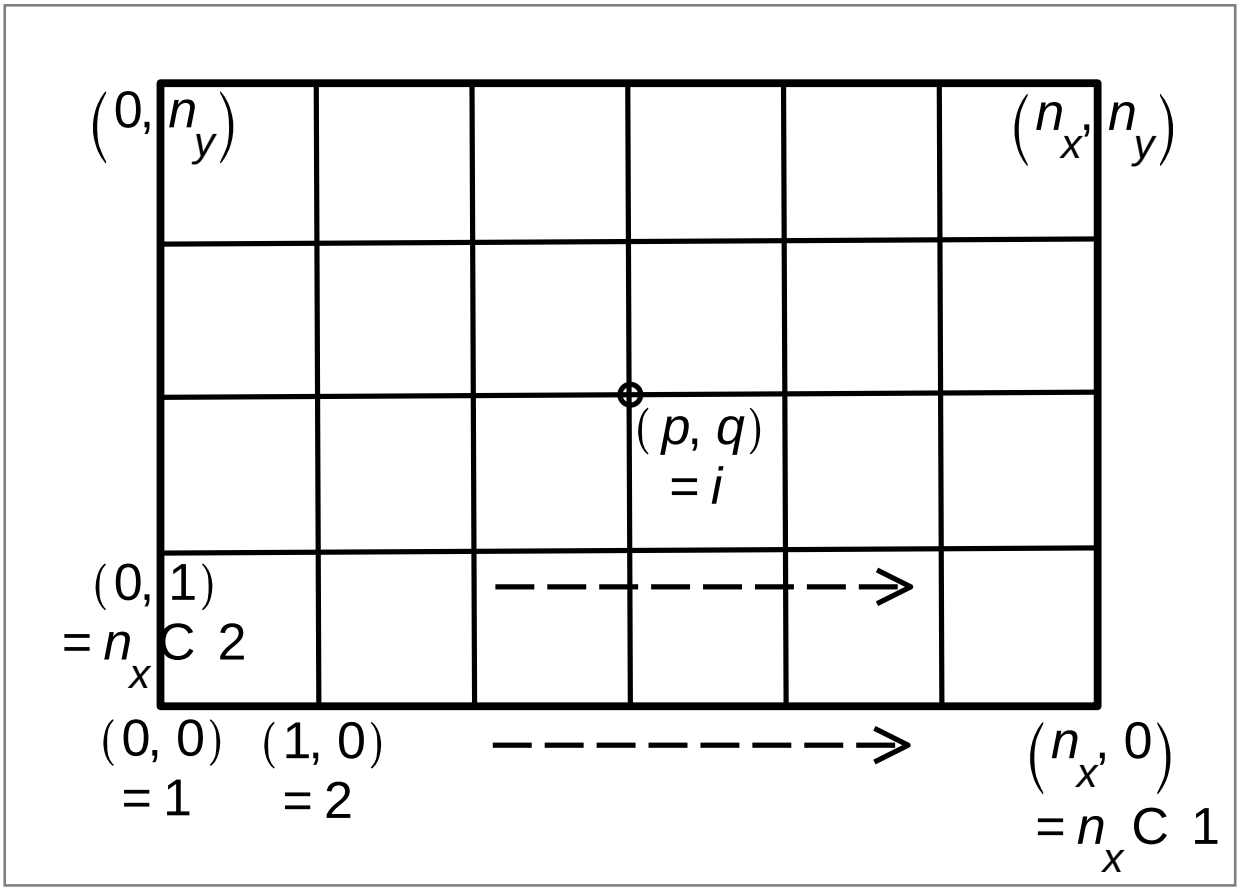
- The Crank-Nicolson method is called a *semi-implicit* method.
- The matrix in the left-hand side of Equation **(16)** is strictly diagonally dominant; the linear system can be solved **iteratively with a great efficiency**, particularly when the number of grid points is relatively small and the initial value is set as, e.g., for $n\geq 1$,

$$w_0^{(n+1)}=2w^{n}-w^{n-1} \qquad \textbf{(2.1)}$$

- In general, a fast solver must be developed.

Point-Ordering for the Algebraic System

Row-wise order:



$$i = \left(n_x + 1\right) q + p + 1,\ (p,\ q) = i \qquad \textbf{(18)}$$

Operations at the **node** (p, q) occupy **the i-th row** of A.

Examples:

$$(0,\ 1) = \left(n_x + 1\right) \cdot 1 + 0 + 1 = n_x + 2$$
$$\left(0,\ n_y\right) = \left(n_x + 1\right) \cdot n_y + 0 + 1 = \left(n_x + 1\right) \cdot n_y + 1$$
$$\left(n_x,\ n_y\right) = \left(n_x + 1\right) \cdot n_y + n_x + 1 = \left(n_x + 1\right) \cdot \left(n_y + 1\right)$$

Column-wise order can be considered similarly.

Alternating Direction Implicit (ADI) Method

The ADI method was developed in 1955 by Douglas, Peaceman, and Rachford, as a perturbation of the Crank-Nicolson time-stepping procedure. It introduces an extra error called the **splitting error** in $O(k^2)$ which is the same order as the truncation error. However, it does not require to solve a huge linear system, but it marches a time step by solving a series of tridiagonal matrix systems, once in the x-direction followed by one y-directional sweep.

We first rewrite the Crank-Nicolson procedure in Equation **(15)** as

$$\frac{w^{n+1}-w^n}{k}+\frac{1}{2}A_1\left(w^{n+1}+w^n\right)+\frac{1}{2}A_2\left(w^{n+1}+w^n\right)=f^{n+\frac{1}{2}} \qquad \textbf{(19)}$$

Then, the ADI method is formulated as a method of two steps

$$\frac{\overline{v}-v^n}{k}+\frac{1}{2}A_1\left(\overline{v}+v^n\right)+A_2 v^n=f^{n+\frac{1}{2}} \qquad \textbf{(20)}$$

$$\frac{v^{n+1}-v^n}{k}+\frac{1}{2}A_1\left(\overline{v}+v^n\right)+\frac{1}{2}A_2\left(v^{n+1}+v^n\right)=f^{n+\frac{1}{2}} \qquad \textbf{(21)}$$

where $\overline{w}$ denotes an intermediate solution.

Notes

- Equation **(20)** is to solve for the x-direction, with $A_2 w^{n+1}$ replaced by the last available quantity $A_2 w^n$.
- Equation **(21)** is to solve for the y-direction, with $A_1 w^{n+1}$ replaced by the last available quantity $A_1 \overline{w}$.
- **(21) - (20)**: $\frac{v^{n+1}-\overline{v}}{k}+\frac{1}{2}A_2 v^{n+1}-\frac{1}{2}A_2 v^n=0$

Implementation of ADI method

Equations **(20)** and **(21)** read equivalently

$$\left(1+\frac{1}{2}kA_1\right)\overline{v}=\left(1-\frac{1}{2}kA_1-kA_2\right)v^n+kf^{n+\frac{1}{2}} \qquad \textbf{(22)}$$

$$\left(1+\frac{1}{2}kA_2\right)v^{n+1}=\overline{v}+\frac{1}{2}kA_2v^n \qquad \textbf{(23)}$$

Notes

- Equation **(22)** is a simple rearrangement of Equation **(20)**.
- Equation **(23)** is obtained from Equation **(21)** subtracted by Equation **(20)**.
- Both **(22)** and **(23)** require only to solve a series of tridiagonal algebraic systems.
- The matrix A_1 is an x-directional operator, while A_2 becomes an y-directional operator. Thus the other directional copy of the solution vector v is needed to carry out the computation conveniently.

Splitting Error of ADI

It follows from **(23)** that

$$\overline{v} = \left(1 + \frac{1}{2}\, k A_2\right) v^{n+1} - \frac{1}{2}\, k A_2\, v^n$$

Plugging it into **(22)**

$$\left(1 + \frac{1}{2}\, k A_1\right)\left(1 + \frac{1}{2}\, k A_2\right) v^{n+1}$$

$$= \left(1 + \frac{1}{2}\, k A_1\right) \frac{1}{2}\, k A_2\, v^n + \left(1 - \frac{1}{2}\, k A_1 - k A_2\right) v^n + k f^{n+\frac{1}{2}}$$

$$= v^n - \frac{1}{2}\, k A\, v^n + \frac{k^2}{4} A_1 A_2\, v^n + k f^{n+\frac{1}{2}}$$

Since the left-hand side of the above equation gives

$$v^{n+1} + \frac{1}{2}\, k A\, v^{n+1} + \frac{k^2}{4} A_1 A_2 v^{n+1},$$

by rearranging the terms, we obtain

$$\frac{v^{n+1} - v^n}{k} + A\left(\frac{1}{2}\, v^{n+1} + \frac{1}{2}\, v^n\right) + \mathbf{\frac{1}{4}\, k A_1 A_2 \left(v^{n+1} - v^n\right)} = f^{n+\frac{1}{2}} \qquad \textbf{(24)}$$

Notes

- The term in boldface is the *splitting error*, which is in $\mathrm{O}\left(k^2\right)$, more precisely,

$$\mathrm{O}\left(\partial_{xx}\partial_{yy}\partial_t u \cdot k^2\right)$$

- The splitting error can be much larger than the truncation error

$$\mathrm{O}\left(u_{xxxx} h_x^2 + u_{yyyy} h_y^2 + u_{ttt} k^2\right)$$

- A remedy has been proposed by (Douglas-Kim, 2001): adding to the right side of **(24)**

$$\frac{1}{4}\, k A_1 A_2 \left(v^n - v^{n-1}\right), \quad n \geq 1$$

which makes the splitting error in $\mathrm{O}\left(k^3\right)$

- **Applications of ADI**

  Various operator splitting methods

  - **Examples**

    System of nonlinear equations

    Projection methods for Navier-Stokes's Equations

Example: Let

$$((x, y), t) \in R \times [0, T] = [0, 1]^2 \times [0, 1]$$

Use the ADI method to approximate the solution to

$$u_t - u_{xx} - u_{yy} = \pi \sin(2\pi x)\ \sin(2\pi y)\left(\cos(\pi t) + 8\pi \sin(\pi t)\right)$$

$$u_0(x, y) = 0,\ \ (x, y) \in R$$

$$u(x, y, t) = 0,\ \ ((x, y), t) \in \partial R \times [0, T]$$

with $n_x = n_y = n_t = 10, 20,$ and 40. Then analyze the error; the actual solution

$$u(x, y, t) = \sin(2\pi x) \cdot \sin(2\pi y) \cdot \sin(\pi t)$$

Solution:

Recall that the ADI can be implemented with Equations **(22)** and **(23)**:

$$\left(1 + \frac{1}{2} k A_1\right) \overline{v} = \left(1 - \frac{1}{2} k A_1 - k A_2\right) v^n + k f^{n + \frac{1}{2}}$$

$$\left(1 + \frac{1}{2} k A_2\right) v^{n+1} = \overline{v} + \frac{1}{2} k A_2 v^n$$

where each of A_ℓ, $\ell = 1, 2$, is a collection of tridiagonal matrices. For example, A_1 contains $n_y \pm 1$ tridiagonal matrices, each of which is representing the FD approximation of $-\partial_{xx}$ on a horizontal grid line and has dimension of $n_x \pm 1$. The differential operator $-\partial_{xx}$ is identical for all horizontal grid lines, so is the tridiagonal matrices. Thus you may save only one tridiagonal matrix for A_1. The same is applied for A_2.

Empty Page

CN-ADI for Parabolic PDE in 2D ($u = 0$ on ∂R, only)

Let

$((x, y), t) \in R \times [0, T] = [0, 1]^2 \times [0, 1]$

Use the ADI method to approximate the solution to

$u_t - u_{xx} - u_{yy} = \sin(2\pi x)\ \sin(2\pi y)\ \pi\ (\cos(\pi t) + 8\pi \sin(\pi t))$

$u(x, y, t = 0) = 0,\ (x, y) \in R$

$u(x, y, t) = 0,\ ((x, y), t) \in \partial R \times [0, T]$

with $n_x = n_y = n_t = 10$, 20, and 40.

Then analyze the error. The actual solution

$u(x, y, t) = \sin(2\pi x) \cdot \sin(2\pi y) \cdot \sin(\pi t)$

```
restart
Digits := 15 :

Matrix3Vector := proc(A, b)
    local i, n, x;
    n := LinearAlgebra[RowDimension](A);
    x := Vector(n);
    ## Forward Elimination
    i := 1;
       x[i] := A[i, 2]·b[i] + A[i, 3]·b[i + 1];
    for i from 2 to n - 1 do
       x[i] := A[i, 1]·b[i - 1] + A[i, 2]·b[i] + A[i, 3]·b[i + 1];
    end do;
    i := n;
       x[i] := A[i, 1]·b[i - 1] + A[i, 2]·b[i];
    return x;
end proc:
```

```
TridiagLU := proc(A)
    local i, j, k, n, r, B;
    n := LinearAlgebra[RowDimension](A);
    #B:=A;  ## Do not use this line for Matrix copy
    B := Matrix(n, 3);
    for j to 3 do; for i to n do
        B[i, j] := A[i, j];
    end do; end do;
    for k from 1 to n - 1 do
        r := B[k + 1, 1]/B[k, 2];
        B[k + 1, 1] := r;
        B[k + 1, 2] := B[k + 1, 2] - r·B[k, 3];
    end do;
    return B;
end proc:

TridiagSOLVE := proc(A, b)
    local k, n, x, y;
    n := LinearAlgebra[RowDimension](A);
    x := Vector(n);
    y := Vector(n);
    ## Forward Elimination
    y[1] := b[1];
    for k from 2 to n do
        y[k] := b[k] - A[k, 1]·y[k - 1];
    end do;
    ## Backward Substitution
    x[n] := y[n]/A[n, 2];
    for k from n - 1 by -1 to 1 do
        x[k] := (y[k] - A[k, 3]·x[k + 1])/A[k, 2];
    end do;
    return x;
end proc:
```

```
##=========================================
## The Main Code
##=========================================
ADI2D := proc(a, b, c, d, T, nx, ny, nt, f, u0)
    local p, q, n, hx, hy, k, halfk;
    local xp, yq, tn;
    local v1, v2, vb, kf, A1, A2, B1, B2, B1LU, B2LU, A1v, A2v;
    local vec1, vec2, ws1, ws2;

    print(`a,b,c,d,T,nx,ny,nt,f,u0=`, a, b, c, d, T, nx, ny, nt, f, u0);
    with(LinearAlgebra) :
    hx := (b - a) / nx;
    hy := (d - c) / ny;
    k  := T / nt;
    halfk := k / 2;

    A1 := Matrix(nx + 1, 3);  B1 := Matrix(nx + 1, 3);
    A2 := Matrix(ny + 1, 3);  B2 := Matrix(ny + 1, 3);
    B1LU := Matrix(nx + 1, 3);
    B2LU := Matrix(ny + 1, 3);
    A1v := Matrix(nx + 1, ny + 1);
    A2v := Matrix(ny + 1, nx + 1);
    v1 := Matrix(nx + 1, ny + 1);
    v2 := Matrix(ny + 1, nx + 1);
    vb := Matrix(nx + 1, ny + 1);
    kf := Matrix(nx + 1, ny + 1);

    ws1 := Matrix(nx + 1, ny + 1);
    ws2 := Matrix(ny + 1, nx + 1);
    vec1 := Vector(nx + 1);
    vec2 := Vector(ny + 1);

   # Set the initial solution vector v^0
    for q from 1 to ny + 1 do
       yq := c + (q - 1) · hy;
       for p from 1 to nx + 1 do
          xp := a + (p - 1) · hx;
          v1[p, q] := evalf(eval(u0, [x = xp, y = yq]));
       end do;
    end do;
```

```
v2 := v1^%T;

# Obtain the Directional matrices
 for p from 2 to nx do
    A1[p, 1] := -1/hx^2; A1[p, 2] :=  2/hx^2; A1[p, 3] := -1/hx^2;
 end do;
 for q from 2 to ny do
    A2[q, 1] := -1/hy^2; A2[q, 2] :=  2/hy^2; A2[q, 3] := -1/hy^2;
 end do;

 B1 := halfk·A1; B2 := halfk·A2;
 for p from 1 to nx + 1 do B1[p, 2] := 1 + B1[p, 2]; end do;
 for q from 1 to ny + 1 do B2[q, 2] := 1 + B2[q, 2]; end do;
 B1LU := TridiagLU(B1); B2LU := TridiagLU(B2);

# ========================================
# Now, Marching with the ADI
# ========================================
 for n from 1 to nt do
    tn := (n - 1/2)·k;

   # Set kf = k·f(tn)
    for q from 2 to ny do
       yq := c + (q - 1)·hy;
       for p from 2 to nx do
          xp := a + (p - 1)·hx;
          kf[p, q] := evalf( k·eval(f, [x=xp, y=yq, t=tn]) );
       end do;
    end do;

   # Compute "A1v"
    for q from 2 to ny do
       vec1 := Matrix3Vector(A1, Vector[column](v1[1 ..nx + 1, q]) );
       for p from 1 to nx + 1 do A1v[p, q] := vec1[p]; end do;
    end do;

   # Compute "A2v"
    for p from 2 to nx do
       vec2 := Matrix3Vector(A2, Vector[column](v2[1 ..ny + 1, p]) );
```

```
        for q from 1 to ny + 1 do A2v[q, p] := vec2[q]; end do;
      end do;

      # X-sweep
      #====================
      for q from 1 to ny + 1 do; for p from 1 to nx + 1 do
        ws1[p, q] := v1[p, q] − halfk·A1v[p, q] − k·A2v[q, p] + kf[p, q];
      end do; end do;

      for q from 2 to ny do
        vec1 := TridiagSOLVE(B1LU, Vector[column](ws1[1 ..nx + 1, q]));
        for p from 1 to nx + 1 do
          vb[p, q] := vec1[p];
        end do;
      end do;

      # Y-sweep
      #====================
      for p from 1 to nx + 1 do; for q from 1 to ny + 1 do
        ws2[q, p] := vb[p, q] + halfk·A2v[q, p];
      end do; end do;

      for p from 2 to nx do
        vec2 := TridiagSOLVE(B2LU, Vector[column](ws2[1 ..ny + 1, p]));
        for q from 1 to ny + 1 do
          v2[q, p] := vec2[q];
        end do;
      end do;

      # Another copy of the solution
      #====================
      v1 := v2^%T;
    end do;
    return v1;
end proc:
```

```
##=========================================
## Problem Setting & Solve
##=========================================
a := 0 : b := 1 : c := 0 : d := 1 : T := 1 :
u := sin(2 π x)·sin(2 π y)·sin(π t) :
diff(u, t) − diff(u, x, x) − diff(u, y, y)
```

$$\sin(2\pi x)\sin(2\pi y)\cos(\pi t)\pi + 8\sin(2\pi x)\pi^2\sin(2\pi y)\sin(\pi t) \quad (1)$$

```
simplify(%)
```

$$\sin(2\pi x)\sin(2\pi y)\pi\left(\cos(\pi t) + 8\pi\sin(\pi t)\right) \quad (2)$$

```
> u0 := eval(u, t=0);
```

$$u0 := 0 \quad (3)$$

```
> f := sin(2 π x) sin(2 π y) π (cos(π t) + 8 π sin(π t))
```

$$f := \sin(2\pi x)\sin(2\pi y)\pi\left(\cos(\pi t) + 8\pi\sin(\pi t)\right) \quad (4)$$

(5)

```
> nx := 80; ny := nx;  nt := nx;
```

$$nx := 80$$
$$ny := 80$$
$$nt := 80 \quad (6)$$

```
uT := ADI2D(a, b, c, d, T, nx, ny, nt, f, u0) :
```

a,b,c,d,T,nx,ny,nt,f,u0=, 0, 1, 0, 1, 1, 80, 80, 80, $\sin(2\pi x)\sin(2\pi y)\pi\left(\cos(\pi t) + 8\pi\sin(\pi t)\right)$, 0 (7)

```
U[nx] := Array(1 ..nx + 1, 1 ..ny + 1) : U[nx] := uT:

### Checking error
hx := (b − a) / nx : hy := (d − c) / ny :
err := 0 :
 for q from 1 to ny + 1 do
    yq := c + (q − 1)·hy;
    for p from 1 to nx + 1 do
       xp := a + (p − 1)·hx;
       eu := evalf( eval(u, [x=xp, y=yq, t=T]) );
       err := max(err, abs(uT[p, q] − eu) );
    end do;
end do:
```

```
print(`err=`, err);
```

```
                                err=, 0.00244437753383986                    (8)
```

```
## nx=10: err=, 0.1395588783
## nx=20: err=, 0.03913158771
## nx=40: err=, 0.009780608328
## nx=80: err=, 0.002444378205
```

Richardson Extrapolation

```
extrapolation := 1 :

if (extrapolation = 1) then
for it from 1 to 2 do
    nx := 2^(it-1)·20 : ny := nx :
    hx := (b-a)/nx : hy := (d-c)/ny :
    err := 0 :
    for q from 1 to ny+1 do
       yq := c+(q-1)·hy;
       for p from 1 to nx+1 do
          xp := a+(p-1)·hx;
          eu := evalf[14]( eval(u, [x=xp, y=yq, t=T]) );
          richardson := (4·U[2·nx][2·p-1, 2·q-1]-U[nx][p, q])/3;
          err := max(err, abs(richardson-eu));
       end do;
    end do:
    printf(" nx=ny=nt=%3d:  Richardson_Error = %g\n", nx, err);
end do:
end if;
```

```
 nx=ny=nt= 20:  Richardson_Error = 3.05212e-06
 nx=ny=nt= 40:  Richardson_Error = 1.03255e-06
```

Empty Page

CN-SOR for Parabolic PDE in 2D ($u = 0$ on ∂R, only)

Let

$((x, y), t) \in R \times [0, T] = [0, 1]^2 \times [0, 1]$

Use the ADI method to approximate the solution to

$u_t - u_{xx} - u_{yy} = \sin(2\pi x)\ \sin(2\pi y)\ \pi\ (\cos(\pi t) + 8\pi \sin(\pi t))$

$u(x, y, t = 0) = 0,\ \ (x, y) \in R$

$u(x, y, t) = 0,\ \ ((x, y), t) \in \partial R \times [0, T]$

with $n_x = n_y = n_t = 10$, 20, and 40.

Then analyze the error. The actual solution

$u(x, y, t) = \sin(2\pi x) \cdot \sin(2\pi y) \cdot \sin(\pi t)$

restart

```
SOR2D := proc(nx, ny, A, W, r, ω, tol, itmax)
    local p, q, it, err, wGS, wnew;

    err := 1;
    for it from 1 to itmax while (err > tol) do
       err := 0;
       for q from 2 to ny do
       for p from 2 to nx do
          wGS := (r[p, q] - ( A[2, p, q]·W[p, q + 1]
                + A[-1, p, q]·W[p - 1, q] + A[1, p, q]·W[p + 1, q]
                                + A[-2, p, q]·W[p, q - 1] ) )/A[0, p, q];
          wnew := (1 - ω)·W[p, q] + ω·wGS;
          err := max(err, abs(wnew - W[p, q]));
          W[p, q] := wnew;
       end do; end do;
    end do;
    if (it ≥ itmax) then
       printf("SOR: err=%g in %d iterations\n", err, it);
    end if;
    return W;
end proc:
```

```
##=======================================
## The Main Code
##=======================================
ParabolicSOR2D := proc(a, b, c, d, T, nx, ny, nt, f, u0, ω, tol, itmax)
    local i, p, q, n, hx, hy, k, halfk, dig;
    local xp, yq, tn;
    local KF, W, A, B, r;

    print(a,b,c,d,T,nx,ny,nt,f,u0,ω,tol,itmax=, a, b, c, d, T, nx, ny, nt, f, u0, ω, tol, itmax);
    with(LinearAlgebra) :
    hx := (b − a) / nx;
    hy := (d − c) / ny;
    k  := T / nt;
    halfk := k / 2;
    dig := 14;

    KF := Matrix(nx + 1, ny + 1);
    W  := Matrix(nx + 1, ny + 1);
    A  := Array( −2 ..2, 1 ..nx + 1, 1 ..ny + 1);
    B  := Array( −2 ..2, 1 ..nx + 1, 1 ..ny + 1);
    r  := Matrix(nx + 1, ny + 1);

   # Set the initial solution vector w^0
    for q from 1 to ny + 1 do
       yq := c + (q − 1) · hy;
       for p from 1 to nx + 1 do
          xp := a + (p − 1) · hx;
          W[p, q] := evalf[dig](eval(u0, [x = xp, y = yq]));
       end do;
    end do;

   # Get the matrix A  and B
    for q from 2 to ny do
    for p from 2 to nx do
       A[ −2, p, q] := −1 / hy^2;
       A[ −1, p, q] := −1 / hx^2;
       A[  0, p, q] :=  2 / hx^2 + 2 / hy^2;
       A[  1, p, q] := −1 / hx^2;
```

```
    A[ 2,p,q] := -1/hy^2;
 end do; end do;

 for q from 1 to ny+1 do
 for p from 1 to nx+1 do
    for i from -2 to 2 do
       A[i,p,q] := halfk*A[i,p,q];
       B[i,p,q] := -A[i,p,q];
    end do;
    A[0,p,q] := 1+A[0,p,q]; B[0,p,q] := 1+B[0,p,q]
 end do; end do;

# ======================================
# Now, Marching with SOR
# ======================================
 for n from 1 to nt do
    tn := (n-1/2)*k;

   # Set KF = k*f(tn)
    for q from 2 to ny do
       yq := c+(q-1)*hy;
       for p from 2 to nx do
          xp := a+(p-1)*hx;
          KF[p,q] := evalf[dig]( k*eval(f,[x=xp,y=yq,t=tn]) );
       end do;
    end do;

   # Get the RHS
    for q from 2 to ny do
    for p from 2 to nx do
       r[p,q] := KF[p,q]              +B[2,p,q]*W[p,q+1]
           +B[-1,p,q]*W[p-1,q]+B[0,p,q]*W[p,q]+B[1,p,q]*W[p+1,q]
                                      +B[-2,p,q]*W[p,q-1];
    end do; end do;

   # Call SOR
    W := SOR2D(nx,ny,A,W,r,ω,tol,itmax);

 end do;
```

```
    return W;
end proc:

##=========================================
## Problem Setting & Solve
##=========================================
a := 0 : b := 1 : c := 0 : d := 1 : T := 1 :
ω := 1.7 : tol := 10^(-8) : itmax := 1000 :
u := sin(2 π x)·sin(2 π y)·sin(π t) :
diff(u, t) − diff(u, x, x) − diff(u, y, y)
```

$$\sin(2\pi x)\sin(2\pi y)\cos(\pi t)\pi + 8\sin(2\pi x)\pi^2\sin(2\pi y)\sin(\pi t) \quad (1)$$

```
simplify(%)
```

$$\sin(2\pi x)\sin(2\pi y)\pi\left(\cos(\pi t) + 8\pi\sin(\pi t)\right) \quad (2)$$

```
> u0 := eval(u, t=0);
```

$$u0 := 0 \quad (3)$$

```
> f := sin(2 π x) sin(2 π y) π (cos(π t) + 8 π sin(π t))
```

$$f := \sin(2\pi x)\sin(2\pi y)\pi\left(\cos(\pi t) + 8\pi\sin(\pi t)\right) \quad (4)$$

```
>
```

(5)

```
> nx := 80; ny := nx;  nt := nx;
```

$$nx := 80$$
$$ny := 80$$
$$nt := 80 \quad (6)$$

```
uT := ParabolicSOR2D(a, b, c, d, T, nx, ny, nt, f, u0, ω, tol, itmax) :
a,b,c,d,T,nx,ny,nt,f,u0,ω,tol,itmax=, 0, 1, 0, 1, 1, 80, 80, 80,
```

(7)

$$\sin(2\pi x)\sin(2\pi y)\pi\left(\cos(\pi t) + 8\pi\sin(\pi t)\right), 0, 1.7, \frac{1}{100000000}, 1000$$

```
U[nx] := Array(1 ..nx + 1, 1 ..ny + 1) : U[nx] := uT:

errorcheck := 1 :
### Checking error
if errorcheck = 1 then
hx := (b − a) / nx : hy := (d − c) / ny :
```

```
err := 0 :
 for q from 1 to ny + 1 do
    yq := c + (q − 1) ·hy;
    for p from 1 to nx + 1 do
       xp := a + (p − 1) ·hx;
       eu := evalf[14]( eval(u, [x=xp, y=yq, t=T]) );
       err := max(err, abs(uT[p, q] − eu) );
    end do;
end do:
 printf(" nx=%3d: error=%g\n", nx, err);
end if:
 nx= 80: error=2.58319e-05

##  nx= 10: error=0.00159191
##  nx= 20: error=0.000415578
##  nx= 40: error=0.000102565
## nx= 80: error=2.58319e-05

## Result from ADI:
## nx=10: err=, 0.1395588783
## nx=20: err=, 0.03913158771
## nx=40: err=, 0.009780608328
## nx=80: err=, 0.002444378205
```

Richardson Extrapolation

```
extrapolation := 1 :

if (extrapolation=1) then
for it from 1 to 2 do
    nx := 2^(it − 1) ·20 : ny := nx :
    hx := (b − a) /nx : hy := (d − c) /ny :
    err := 0 :
    for q from 1 to ny + 1 do
       yq := c + (q − 1) ·hy;
       for p from 1 to nx + 1 do
          xp := a + (p − 1) ·hx;
          eu := evalf[14]( eval(u, [x=xp, y=yq, t=T]) );
```

```
            richardson := (4·U[2·nx][2·p − 1, 2·q − 1] − U[nx][p, q]) /3;
            err := max(err, abs(richardson − eu) );
        end do;
    end do:
    printf(" nx=ny=nt=%3d:  Richardson_Error = %g\n", nx, err);
end do:
end if;
```

```
 nx=ny=nt= 20:  Richardson_Error = 1.80033e-06
 nx=ny=nt= 40:  Richardson_Error = 5.47026e-07

## Result from ADI
 nx=ny=nt= 20:  Richardson_Error = 3.05321e-06
 nx=ny=nt= 40:  Richardson_Error = 1.03328e-06
```

Note:

- For 4th-order scheme, the CN-ADI produces a relatively small splitting error.
- When *nx* = *ny* = 80, CN-ADI takes about 2 minutes, while CN-SOR takes more than 30 minutes.
- There is a *simple* remedy for the elimination of the splitting error (with 3 -5% more computation time); see **J. Douglas, Jr. and S. Kim** (2001). "*Improved accuracy for locally one-dimensional methods for parabolic equations*". Math. Models and Methods in Appl. Sci. 11, pp. 1563-1579.

CN-ADI-II for Parabolic PDE in 2D ($u = 0$ on ∂R, only)

Let

$((x, y), t) \in R \times [0, T] = [0, 1]^2 \times [0, 1]$

Use the ADI method to approximate the solution to

$u_t - u_{xx} - u_{yy} = \sin(2\pi x)\ \sin(2\pi y)\ \pi\ (\cos(\pi t) + 8\pi \sin(\pi t))$

$u(x, y, t = 0) = 0,\ (x, y) \in R$

$u(x, y, t) = 0,\ ((x, y), t) \in \partial R \times [0, T]$

with $n_x = n_y = n_t = 10$, 20, and 40.

Then analyze the error. The actual solution

$u(x, y, t) = \sin(2\pi x) \cdot \sin(2\pi y) \cdot \sin(\pi t)$

```
restart
Digits := 15 :

Matrix3Vector := proc(A, b)
    local i, n, x;
    n := LinearAlgebra[RowDimension](A);
    x := Vector(n);
    ## Forward Elimination
    i := 1;
       x[i] := A[i, 2]·b[i] + A[i, 3]·b[i + 1];
    for i from 2 to n - 1 do
       x[i] := A[i, 1]·b[i - 1] + A[i, 2]·b[i] + A[i, 3]·b[i + 1];
    end do;
    i := n;
       x[i] := A[i, 1]·b[i - 1] + A[i, 2]·b[i];
    return x;
end proc:
```

```
TridiagLU := proc(A)
    local i, j, k, n, r, B;
    n := LinearAlgebra[RowDimension](A);
    #B:=A;  ## Do not use this line for Matrix copy
    B := Matrix(n, 3);
    for j to 3 do; for i to n do
       B[i, j] := A[i, j];
    end do; end do;
    for k from 1 to n - 1 do
       r := B[k + 1, 1]/B[k, 2];
       B[k + 1, 1] := r;
       B[k + 1, 2] := B[k + 1, 2] - r·B[k, 3];
    end do;
    return B;
end proc:

TridiagSOLVE := proc(A, b)
    local k, n, x, y;
    n := LinearAlgebra[RowDimension](A);
    x := Vector(n);
    y := Vector(n);
    ## Forward Elimination
    y[1] := b[1];
    for k from 2 to n do
       y[k] := b[k] - A[k, 1]·y[k - 1];
    end do;
    ## Backward Substitution
    x[n] := y[n]/A[n, 2];
    for k from n - 1 by -1 to 1 do
       x[k] := (y[k] - A[k, 3]·x[k + 1]) /A[k, 2];
    end do;
    return x;
end proc:
```

```
##=========================================
## The Main Code
##=========================================
ADI2D := proc(a, b, c, d, T, nx, ny, nt, f, u0)
    local p, q, n, hx, hy, k, halfk, it, itmax;
    local xp, yq, tn, k2o4;
    local v1, v2, vb, kf, A1, A2, B1, B2, B1LU, B2LU, A1v, A2v;
    local dv2, A2vs;
    local vec1, vec2, ws1, ws2;

    print(`a,b,c,d,T,nx,ny,nt,f,u0=`, a, b, c, d, T, nx, ny, nt, f, u0);
    with(LinearAlgebra) :
    hx := (b - a) / nx;
    hy := (d - c) / ny;
    k  := T / nt;
    halfk := k / 2;
    k2o4 := k^2 / 4;
    itmax := 10 :

    A1 := Matrix(nx + 1, 3);  B1 := Matrix(nx + 1, 3);
    A2 := Matrix(ny + 1, 3);  B2 := Matrix(ny + 1, 3);
    B1LU := Matrix(nx + 1, 3);
    B2LU := Matrix(ny + 1, 3);
    A1v := Matrix(nx + 1, ny + 1);
    A2v := Matrix(ny + 1, nx + 1);
    v1 := Matrix(nx + 1, ny + 1);
    v2 := Matrix(ny + 1, nx + 1);
    vb := Matrix(nx + 1, ny + 1);
    kf := Matrix(nx + 1, ny + 1);

    ws1 := Matrix(nx + 1, ny + 1);
    ws2 := Matrix(ny + 1, nx + 1);
    vec1 := Vector(nx + 1);
    vec2 := Vector(ny + 1);
    A2vs := Matrix(ny + 1, nx + 1); # added for ADI-II
    dv2 := Matrix(ny + 1, nx + 1);

   # Set the initial solution vector v^0
    for q from 1 to ny + 1 do
       yq := c + (q - 1) · hy;
```

```
    for p from 1 to nx + 1 do
        xp := a + (p − 1)·hx;
        v1[p, q] := evalf(eval(u0, [x = xp, y = yq]));
    end do;
end do;
v2 := v1^%T;

# Obtain the Directional matrices
for p from 2 to nx do
    A1[p, 1] := -1/hx^2; A1[p, 2] := 2/hx^2; A1[p, 3] := -1/hx^2;
end do;
for q from 2 to ny do
    A2[q, 1] := -1/hy^2; A2[q, 2] := 2/hy^2; A2[q, 3] := -1/hy^2;
end do;

B1 := halfk·A1; B2 := halfk·A2;
for p from 1 to nx + 1 do B1[p, 2] := 1 + B1[p, 2]; end do;
for q from 1 to ny + 1 do B2[q, 2] := 1 + B2[q, 2]; end do;
B1LU := TridiagLU(B1); B2LU := TridiagLU(B2);

# ======================================
# Now, Marching with the ADI
# ======================================
for n from 1 to nt do
    tn := (n − 1/2)·k;

   # Set kf = k·f(tn)
    for q from 2 to ny do
        yq := c + (q − 1)·hy;
        for p from 2 to nx do
            xp := a + (p − 1)·hx;
            kf[p, q] := evalf( k·eval(f, [x = xp, y = yq, t = tn]));
        end do;
    end do;

   # The correction term
    if (n ≥ 2) then
        for p from 2 to nx do
            vec2 := Matrix3Vector(A2, Vector[column](dv2[1 ..ny + 1, p]));
```

```
      for q from 1 to ny + 1 do ws1[p, q] := vec2[q]; end do;
    end do;
    for q from 2 to ny do
      vec1 := Matrix3Vector(A1, Vector[column](ws1[1 ..nx + 1, q]));
      for p from 1 to nx + 1 do
        kf[p, q] := kf[p, q] + k2o4·vec1[p];
      end do;
    end do;
 end if;

# Compute "A1v"
 for q from 2 to ny do
    vec1 := Matrix3Vector(A1, Vector[column](v1[1 ..nx + 1, q]));
    for p from 1 to nx + 1 do A1v[p, q] := vec1[p]; end do;
 end do;

# Compute "A2v"
 for p from 2 to nx do
    vec2 := Matrix3Vector(A2, Vector[column](v2[1 ..ny + 1, p]));
    for q from 1 to ny + 1 do A2v[q, p] := vec2[q]; end do;
 end do;

# X-sweep
#====================
 for q from 1 to ny + 1 do; for p from 1 to nx + 1 do
    ws1[p, q] := v1[p, q] − halfk·A1v[p, q] − k·A2v[q, p] + kf[p, q];
 end do; end do;

 for q from 2 to ny do
    vec1 := TridiagSOLVE(B1LU, Vector[column](ws1[1 ..nx + 1, q]));
    for p from 1 to nx + 1 do
      vb[p, q] := vec1[p];
    end do;
 end do;

# Y-sweep
#====================
 for p from 1 to nx + 1 do; for q from 1 to ny + 1 do
    ws2[q, p] := vb[p, q] + halfk·A2v[q, p];
 end do; end do;
```

```
 for p from 2 to nx do
    vec2 := TridiagSOLVE(B2LU, Vector[column](ws2[1 ..ny + 1, p]));
    for q from 1 to ny + 1 do
       v2[q, p] := vec2[q];
    end do;
 end do;

#================================
# Begin the Second round of Sweeps
#================================
 if (n = 1) then; for it from 2 to itmax do
#================================
 for p from 2 to nx do
    vec2 := Matrix3Vector(A2, Vector[column](v2[1 ..ny + 1, p]));
    for q from 1 to ny + 1 do A2vs[q, p] := vec2[q]; end do;
 end do;
# X-sweep
#====================
 for q from 1 to ny + 1 do; for p from 1 to nx + 1 do
    ws1[p, q] := v1[p, q] − halfk·(A1v[p, q] + A2v[q, p] + A2vs[q, p]) + kf[p,
q];
 end do; end do;
 for q from 2 to ny do
    vec1 := TridiagSOLVE(B1LU, Vector[column](ws1[1 ..nx + 1, q]));
    for p from 1 to nx + 1 do
       vb[p, q] := vec1[p];
    end do;
 end do;
# Y-sweep
#====================
 for p from 1 to nx + 1 do; for q from 1 to ny + 1 do
     ws2[q, p] := vb[p, q] + halfk·A2vs[q, p];
 end do; end do;
 for p from 2 to nx do
    vec2 := TridiagSOLVE(B2LU, Vector[column](ws2[1 ..ny + 1, p]));
    for q from 1 to ny + 1 do
       v2[q, p] := vec2[q];
    end do;
 end do;
```

```
    #================================
     end do; end if;
    #================================

     for p from 1 to nx + 1 do; for q from 1 to ny + 1 do
        dv2[q, p] := v2[q, p] − v1[p, q];
     end do; end do;

    # Another copy of the solution
    #====================
     v1 := v2^%T;
  end do;

  return v1;
end proc:
```

```
##=========================================
## Problem Setting & Solve
##=========================================
a := 0 : b := 1 : c := 0 : d := 1 : T := 1 :
u := sin(2 π x)·sin(2 π y)·sin(π t) :
diff(u, t) − diff(u, x, x) − diff(u, y, y)
```

$$\sin(2\pi x)\sin(2\pi y)\cos(\pi t)\pi + 8\sin(2\pi x)\pi^2\sin(2\pi y)\sin(\pi t) \quad (1)$$

```
simplify(%)
```

$$\sin(2\pi x)\sin(2\pi y)\pi\left(\cos(\pi t) + 8\pi\sin(\pi t)\right) \quad (2)$$

```
> u0 := eval(u, t=0);
```

$$u0 := 0 \quad (3)$$

```
> f := sin(2 π x) sin(2 π y) π (cos(π t) + 8 π sin(π t))
```

$$f := \sin(2\pi x)\sin(2\pi y)\pi\left(\cos(\pi t) + 8\pi\sin(\pi t)\right) \quad (4)$$

(5)

```
> nx := 40; ny := nx;  nt := nx;
```

$$nx := 40$$
$$ny := 40$$
$$nt := 40 \quad (6)$$

```
uT := ADI2D(a, b, c, d, T, nx, ny, nt, f, u0) :
```

a,b,c,d,T,nx,ny,nt,f,u0=, 0, 1, 0, 1, 1, 40, 40, 40, $\sin(2\pi x)\sin(2\pi y)\pi\left(\cos(\pi t) + 8\pi\sin(\pi t)\right)$, 0 (7)

```
U[nx] := Array(1 ..nx + 1, 1 ..ny + 1) : U[nx] := uT:

### Checking error
hx := (b − a) / nx : hy := (d − c) / ny :
err := 0 :
 for q from 1 to ny + 1 do
    yq := c + (q − 1)·hy;
    for p from 1 to nx + 1 do
       xp := a + (p − 1)·hx;
       eu := evalf( eval(u, [x=xp, y=yq, t=T]) );
       err := max(err, abs(uT[p, q] − eu) );
    end do;
end do:
```

printf(" nx=%3d: error=%g\n", *nx*, *err*);
nx= 40: error=0.000162894

Result from CN-ADI-II:
nx= 10: error=0.01163946
nx= 20: error=0.001150287
nx= 40: error=0.000162894
nx= 80: error=3.12040e-05

Result from CN-SOR:
nx= 10: error=0.00159191
nx= 20: error=0.000415578
nx= 40: error=0.000102565
nx= 80: error=2.58319e-05

Result from CN-ADI:
nx=10: err=, 0.1395588783
nx=20: err=, 0.03913158771
nx=40: err=, 0.009780608328
nx=80: err=, 0.002444378205

Empty Page

12.3. Finite Element Methods for Elliptic Problems in 2D

The model problem we will consider is the elliptic problem in two dimensions of the form

$$
\begin{gathered}
-\nabla\cdot(K\nabla u) + r\,u = f,\ \ (x,y)\in\Omega \\
K=\begin{bmatrix} p(x,y) & 0 \\ 0 & q(x,y) \end{bmatrix} \\
u(x,y)=g_1(x,y),\ \ (x,y)\in\Gamma_1;\quad (\boldsymbol{K}\nabla\boldsymbol{u})\cdot\boldsymbol{\nu}=\boldsymbol{g_2}(\boldsymbol{x},\boldsymbol{y}),\ \ (x,y)\in\Gamma_2
\end{gathered}
\tag{1}
$$

where p and q are positive functions, ν is the outward unit vector, and $\partial\Omega = \Gamma_1 \cup \Gamma_2$.

The PDE in detail reads

$$
-\frac{\partial}{\partial x}\left(p(x,y)\frac{\partial}{\partial x}u\right) - \frac{\partial}{\partial y}\left(q(x,y)\ \frac{\partial}{\partial y}\ u\right) + r(x,y)\ u = f(x,y)
$$

Variational Formulation: begins with defining a linear space

$$
V=\left\{w : w\in C^0(\Omega),\ \ \nabla w \text{ is piecewise continuous on } \Omega, \text{ and } w(x,y)=0 \text{ on } \Gamma_1\right\}
$$

The essential BC can be incorporated algebraically, as the last part of the assembly. So we may assume $g_1 = 0$ momentarily.

The integration by parts (Green's theorem):

$$\int_{\Omega} -\nabla\cdot(K\nabla u)\, w\, dxdy = -\int_{\partial\Omega} (K\nabla u)\cdot\nu\, w\, d\sigma + \int_{\Omega} K\nabla u\cdot\nabla w\, dxdy$$

$$= -\int_{\Gamma_2} g_2\, w\, d\sigma + \int_{\Omega} K\nabla u\cdot\nabla w\, dxdy, \quad w \in V$$

Thus, the weak form is formulated as

$$B(u, w) = \ell(w), \; w \in V \tag{2}$$

where

$$B(u, w) = \int_{\Omega} (K\nabla u\cdot\nabla w + r\, u\, w)\, dxdy$$

$$= \int_{\Omega} \left(p\, u_x w_x + q\, u_y w_y + r\, u\, w\right) dxdy$$

and

$$\ell(w) = \int_{\Omega} f w\, dxdy + \int_{\Gamma_2} g_2\, w\, d\sigma$$

Formulation of FEM:

- **Partitioning/Triangulation**:
- **Subspace $V_h \subset V$ and basis functions $\{\phi_j\}$**: the numerical solution is represented as a linear combination of basis functions

$$u_h(x, y) = \sum_{j=1}^{n} c_j \phi_j(x, y) \tag{3}$$

- **Application of variational principles**: Here we will consider the Galerkin method.
- **Assembly for a linear system**: the unknown $= \left(c_1, c_2, \cdots, c_n\right)^T$.

The Linear Galerkin FEM

Let the domain be partitioned into a collection of triangular elements $\{E_m\}$. We denote the FEM solution as

$$u_h(x,y) = \sum_{j=1}^{n} c_j \phi_j(x,y) \tag{4}$$

where the *j*-th basis function ϕ_j is linear on each of the elements and satisfies $\phi_j(x_i, y_i) = \delta_{ij}$ for all nodal points $\{(x_i, y_i)\}$.

Then, the linear Galerkin FEM is formulated as

$$B(u_h, \phi_i) = \ell(\phi_i),\ i = 1..n \tag{5}$$

Note that

$$B(u_h, \phi_i) = \int_\Omega \left(p (u_h)_x (\phi_i)_x + q (u_h)_y (\phi_i)_y + r u_h \phi_i \right) dxdy$$

$$= \sum_{j=1}^{n} \left(\int_\Omega \left(p (\phi_j)_x (\phi_i)_x + q (\phi_j)_y (\phi_i)_y + r \phi_j \phi_i \right) dxdy \right) c_j$$

$$= \sum_{j=1}^{n} \left(\sum_E \int_E \left(p (\phi_j)_x (\phi_i)_x + q (\phi_j)_y (\phi_i)_y + r \phi_j \phi_i \right) dxdy \right) c_j$$

and

$$\ell(\phi_i) = \int_\Omega f \phi_i \, dxdy + \int_{\Gamma_2} g_2 \phi_i \, d\sigma = \sum_E \left(\int_E f \phi_i \, dxdy + \int_{E \cap \Gamma_2} g_2 \phi_i \, d\sigma \right)$$

Thus

$$a_{ij} = B\left(\phi_j, \phi_i\right) = \sum_E \int_E \left(p \left(\phi_j\right)_x \left(\phi_i\right)_x + q \left(\phi_j\right)_y \left(\phi_i\right)_y + r \phi_j \phi_i \right) dxdy$$

$$b_i = \ell\left(\phi_i\right) = \sum_E \left(\int_E f \phi_i \, dxdy + \int_{E \cap \Gamma_2} g_2 \phi_i \, d\sigma \right)$$

You may incorporate the Dirichlet boundary condition at the end.

Construction of Basis Functions:

Let the triangle (an element) have three vertices (x_i, y_i), $i = 1, 2, 3$. Then the linear basis function defined on the triangle can be expressed as

$$\phi_i(x, y) = a_i + b_i x + c_i y, \; i = 1, 2, 3 \tag{6}$$

with a property that

$$\phi_i(x_j, y_j) = \delta_{ij} \tag{7}$$

For example, for ϕ_1,

$$\begin{aligned}
\phi_1(x_1, y_1) &= a_1 + b_1 x_1 + c_1 y_1 = 1 \\
\phi_1(x_2, y_2) &= a_1 + b_1 x_2 + c_1 y_2 = 0 \\
\phi_1(x_3, y_3) &= a_1 + b_1 x_3 + c_1 y_3 = 0
\end{aligned}$$

which can be written as the following algebraic system

$$\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

In general, for ϕ_i, the corresponding linear system reads

$$T C_i = e_i \tag{8}$$

where

$$T = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}, \quad C_i = \begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix}$$

Thus, C_i, the coefficients of ϕ_i, is the i-th column of T^{-1}.

Basis Functions for the Element: (x_i, y_i), $i = 1, 2, 3$.

$T\left[a_i, b_i, c_i\right]^T = e_i,\ i = 1, 2, 3$, where $\phi_i = a_i + b_i x + c_i y$.

$$T := \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}:$$

The i-th column of T^{-1} is $\left[a_i, b_i, c_i\right]^T$. However, T^{-1} itself is quite complicated to be used as a formula. Let's manipulate it a bit.

```
##==============================
with(LinearAlgebra) :
detT := Determinant(T)
```

$$x_2 y_3 - y_2 x_3 + x_3 y_1 - x_1 y_3 + x_1 y_2 - x_2 y_1 \tag{1.1}$$

of which the absolute value is known to be twice the area of the triangle.

```
C := simplify(detT T^(-1))
```

$$\begin{bmatrix} x_2 y_3 - x_3 y_2 & -x_1 y_3 + x_3 y_1 & x_1 y_2 - x_2 y_1 \\ -y_3 + y_2 & y_3 - y_1 & y_1 - y_2 \\ -x_2 + x_3 & x_1 - x_3 & x_2 - x_1 \end{bmatrix} \tag{1.2}$$

```
> for i from 1 to 3 do
    ϕ[i] := 1/DeterminantT (C[1, i] + C[2, i] x + C[3, i] y);
  end do;
```

$$\phi_1 := \frac{x_2 y_3 - x_3 y_2 + \left(-y_3 + y_2\right) x + \left(-x_2 + x_3\right) y}{DeterminantT}$$

$$\phi_2 := \frac{-x_1 y_3 + x_3 y_1 + \left(y_3 - y_1\right) x + \left(x_1 - x_3\right) y}{DeterminantT}$$

$$\phi_3 := \frac{x_1 y_2 - x_2 y_1 + \left(y_1 - y_2\right) x + \left(x_2 - x_1\right) y}{DeterminantT} \qquad \textbf{(1.3)}$$

where $DeterminantT = \left(x_1 - x_2\right)\left(y_1 - y_3\right) - \left(x_1 - x_3\right)\left(y_1 - y_2\right)$.

Basis Functions for the Element: (x_i, y_i), $i = 1, 2, 3, 4$.

$Q\,[a_i, b_i, c_i, d_i]^T = e_i,\ \ i = 1, 2, 3, 4$, where

$$\boldsymbol{\phi_i(x, y) = a_i + b_i x + c_i y + d_i x y \quad (i \le i \le 4)}$$

$$Q := \begin{bmatrix} 1 & x_1 & y_1 & x_1 y_1 \\ 1 & x_2 & y_2 & x_2 y_2 \\ 1 & x_3 & y_3 & x_3 y_3 \\ 1 & x_4 & y_4 & x_4 y_4 \end{bmatrix} :$$

The i-column of Q^{-1} is $[a_i, b_i, c_i, d_i]^T$. Let's manipulate it a bit.

```
##==============================
with(LinearAlgebra) :
detQ := Determinant(Q)
```

$$x_4 y_4 x_1 y_2 - x_1 x_4 y_1 y_2 + y_4 x_2 x_1 y_1 + x_1 y_2 y_3 x_2 - x_1 x_4 y_4 y_3 - x_2 y_1 y_3 x_1 - y_4 x_1 y_2 x_2 \quad \textbf{(2.1)}$$
$$- y_4 x_1 x_3 y_1 + x_1 y_4 y_3 x_3 + x_4 y_1 y_3 x_1 - y_2 y_3 x_3 x_1 + x_1 y_2 x_3 y_1 - x_4 y_4 x_2 y_1 - x_4 y_2 y_3 x_2$$
$$+ y_4 x_2 x_3 y_2 + y_1 y_3 x_3 x_2 + x_4 y_4 x_2 y_3 + x_4 y_4 x_3 y_1 - x_4 y_4 x_3 y_2 - y_1 x_2 x_3 y_2 - y_4 y_3 x_3 x_2$$
$$- x_4 y_1 y_3 x_3 + x_4 y_1 y_2 x_2 + x_4 y_2 y_3 x_3$$

$$C := simplify\left(detQ\, Q^{-1}\right)$$

$$\left[\left[-x_4 y_2 y_3 x_2 + y_4 x_2 x_3 y_2 + x_4 y_4 x_2 y_3 - x_4 y_4 x_3 y_2 - y_4 y_3 x_3 x_2 + x_4 y_2 y_3 x_3,\ -x_1 x_4 y_4 y_3 \right.\right. \quad \textbf{(2.2)}$$
$$+ x_1 y_4 y_3 x_3 - x_4 y_1 y_3 x_3 + x_4 y_4 x_3 y_1 + x_4 y_1 y_3 x_1 - y_4 x_1 x_3 y_1,\ x_4 y_4 x_1 y_2 - y_4 x_1 y_2 x_2$$
$$- x_4 y_4 x_2 y_1 + x_4 y_1 y_2 x_2 + y_4 x_2 x_1 y_1 - x_1 x_4 y_1 y_2,\ x_1 y_2 y_3 x_2 - y_2 y_3 x_3 x_1 + y_1 y_3 x_3 x_2$$
$$\left.\left. - y_1 x_2 x_3 y_2 - x_2 y_1 y_3 x_1 + x_1 y_2 x_3 y_1 \right]\right.,$$
$$\left[x_4 y_4 y_2 - y_4 y_2 x_2 + y_2 x_2 y_3 - y_2 y_3 x_3 - x_4 y_4 y_3 + y_4 y_3 x_3,\ -x_4 y_4 y_1 + y_4 x_1 y_1 - y_1 y_3 x_1 \right.$$
$$+ y_3 x_3 y_1 + x_4 y_4 y_3 - y_4 y_3 x_3,\ -y_4 x_1 y_1 + x_1 y_2 y_1 + x_4 y_4 y_1 - x_4 y_4 y_2 - y_1 x_2 y_2$$
$$\left. + y_4 y_2 x_2,\ y_1 y_3 x_1 - x_1 y_2 y_1 - y_3 x_3 y_1 - y_2 x_2 y_3 + y_1 x_2 y_2 + y_2 y_3 x_3 \right],$$

$$\left[x_4 y_2 x_2 - x_3 y_2 x_2 - x_4 y_4 x_2 + x_4 y_4 x_3 - x_4 y_3 x_3 + y_3 x_3 x_2,\ x_4 y_4 x_1 + x_3 x_1 y_1 - x_4 x_1 y_1 - y_3 x_3 x_1 + x_4 y_3 x_3 - x_4 y_4 x_3,\ -x_4 y_4 x_1 + x_4 y_4 x_2 - x_1 y_1 x_2 - x_4 y_2 x_2 + x_4 x_1 y_1 + x_1 y_2 x_2,\ y_3 x_3 x_1 - y_3 x_3 x_2 + x_1 y_1 x_2 - x_1 y_2 x_2 + x_3 y_2 x_2 - x_3 x_1 y_1\right],$$
$$\left[-x_4 y_2 + y_4 x_2 - x_2 y_3 - y_4 x_3 + x_3 y_2 + x_4 y_3,\ -y_4 x_1 + x_1 y_3 - x_4 y_3 + y_4 x_3 + x_4 y_1 - x_3 y_1,\ y_4 x_1 - y_4 x_2 + x_2 y_1 - x_4 y_1 + x_4 y_2 - x_1 y_2,\ -x_1 y_3 - x_2 y_1 + x_2 y_3 + x_1 y_2 + x_3 y_1 - x_3 y_2\right]\Big]$$

Too complicated! Okay. Let's give up finding a compact formula for the basis functions. However, you can construct them directly as follows.

Example:

$xy := Matrix(4, 2, [0, 0, 1, 0, 0, 1, 1, 1])$

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \textbf{(2.1.1)}$$

$$Q := \begin{bmatrix} 1 & xy[1,1] & xy[1,2] & xy[1,1]\cdot xy[1,2] \\ 1 & xy[2,1] & xy[2,2] & xy[2,1]\cdot xy[2,2] \\ 1 & xy[3,1] & xy[3,2] & xy[3,1]\cdot xy[3,2] \\ 1 & xy[4,1] & xy[4,2] & xy[4,1]\cdot xy[4,2] \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \textbf{(2.1.2)}$$

$C := Q^{-1}$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{2.1.3}$$

```
> for i from 1 to 4 do
    ϕ[i] := factor(C[1, i] + C[2, i] x + C[3, i] y + C[4, i] x y);
  end do;
```

$$\phi_1 := (y-1)\,(x-1)$$

$$\phi_2 := -(y-1)\,x$$

$$\phi_3 := -(x-1)\,y$$

$$\phi_4 := x\,y \tag{2.1.4}$$

Construction of Basis Functions: for n-gonal elements

a. Select n monomial components for the basis functions
b. Express basis functions as linear combinations of those monomial components
c. Build linear systems $N\,C = e_i$, $1 \le i \le n$, which represent $\phi_i(x_j, y_j) = \delta_{ij}$
d. The i-th column of N^{-1} is the coefficients for ϕ_i

See the attached for an example code for the linear Galerkin FEM.

The Stiffness Matrix: Construction and Access

The stiffness matrix can be saved with two arrays:

$$A[nN, nC],\ AI[nN, nC],\quad nC = 10 \sim 15$$

For each element E, local nodes must be related with global node ordering as follows

Local Node Number	1	2	3
Global Node Number	$gN[1]$	$gN[2]$	$gN[3]$

Let A^E be the local stiffness matrix obtained from the element E:

$$A^E = \begin{bmatrix} a^e_{11} & a^e_{12} & a^e_{13} \\ a^e_{21} & a^e_{22} & a^e_{23} \\ a^e_{21} & a^e_{32} & a^e_{33} \end{bmatrix} \tag{9}$$

##==========

Construction of A

##==========

for i **from** 1 **to** 3 **do**
for j **from** 1 **to** 3 **do**
    $k := 1$:
    while $k \leq cn$ **do**
        if $(AI[gN[i], k] = gN[j])$ **then**
            break;
        elif $(AI[gN[i], k] = 0)$ **then**
            $AI[gN[i], k] := gN[j]$;
            break;
        end if;
        $k := k + 1$;
    end if
    $A[gN[i], k] := A[gN[i], k] + a_{ij}^{e}$;
end do; **end do**;

##==========

Access of A

##==========

$AA :=$**proc**(A, AI, cn, i, j)
    local k;
    for k **from** 1 **to** cn **do**
        if $(AI[i, k] = j)$ **then break**; **end if**;
    end do;
    return $A[i, k]$;
end proc:
$aij := AA(A, AI, cn, i, j)$;

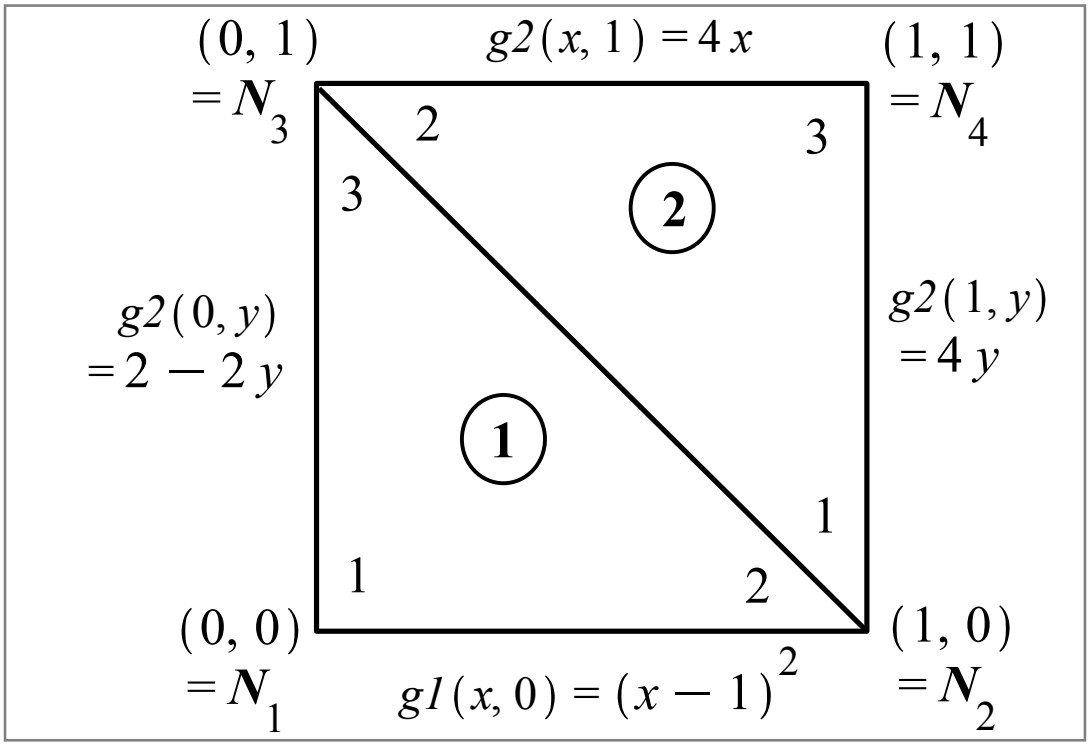
Note: the indices k found in the procedure "AA" can be saved, as inverse indices for the entry a_{ij}, during the first access; with the inverse indices, you can obtain entries of A easily without searching.

Maple Code:
Linear Galerkin FEM for 2D Elliptic Problems
(Second-order Quadrature)

Example: For $\Omega = [0, 1]^2$, consider the problem

$$-\frac{\partial}{\partial x}\left((1+x)\frac{\partial}{\partial x}u\right) - \frac{\partial}{\partial y}\left((1+y)\frac{\partial}{\partial y}u\right) = -6x - 6y$$

with boundary conditions given as in the figure:



Use the linear Galerkin method using the two elements as given in the figure. Compare the accuracy; the actual solution is $u(x, y) = (x + y - 1)^2$.

Note:

$$g_2(x, 1) = \left((K\nabla u)\cdot \nu\right)(x, 1) = \left((1+y)\frac{\partial}{\partial y}u\right)(x, 1)$$
$$= \left((1+y)\,2(x+y-1)\right)(x, 1) = 4x$$

Elementary Stiffness Matrix (ESM) for Linear FEM

```
##=========================================
LinearGalerkin2D := proc(p, q, r, f, b1, b2, b3, eN, conn)
    local i, j, k, x, y, xb, yb, xm, ym, detT, area, lengBE;
    local x1, x2, y1, y2, eA, eb, C, ϕ;
    #print(p(x,y), q(x,y), r(x,y), f(x,y), b1(x,y), b2(x,y), b3(x,y), eN, conn);

    x[1] := eN[1,1]; x[2] := eN[2,1]; x[3] := eN[3,1];
    y[1] := eN[1,2]; y[2] := eN[2,2]; y[3] := eN[3,2];

    detT := x[1]·(y[2] − y[3]) + x[2]·(y[3] − y[1]) + x[3]·(y[1] − y[2]);
    area := abs(detT) / 2;

   ## Construction of ϕ[i]
```

$$C := \frac{1}{detT} \cdot \begin{bmatrix} x_2 y_3 - x_3 y_2 & x_3 y_1 - x_1 y_3 & x_1 y_2 - x_2 y_1 \\ y_2 - y_3 & y_3 - y_1 & y_1 - y_2 \\ -x_2 + x_3 & -x_3 + x_1 & -x_1 + x_2 \end{bmatrix};$$

```
    for i from 1 to 3 do
        ϕ[i] := C[1,i] + C[2,i] xb + C[3,i] yb;
    end do;


    ## Elementary Stiffness Matrix "eA" and the corresponding rhs "eb", with 2nd-order
    quadrature
    eA := Matrix(3,3);
    eb := Vector(3);
    for i to 3 do
        for j to 3 do
            eA[i,j] := (area/3)·add( p(x[k], y[k])·C[2,j]·C[2,i]
                                   + q(x[k], y[k])·C[3,j]·C[3,i]
                                   + r(x[k], y[k])·eval(ϕ[j]·ϕ[i], [xb = x[k], yb = y[k]]), k = 1..3 );
        end do;
        eb[i] := (area/3)·add( f(x[k], y[k])·eval(ϕ[i], [xb = x[k], yb = y[k]]), k = 1..3 );
```

end do;
Neumann Boundary Condition
if $(conn[1]=2)$ **then**
$x1 := x_1;\ x2 := x_2;\quad y1 := y_1;\ y2 := y_2;$
$xm := (x1+x2)/2;\ \ ym := (y1+y2)/2;$
$lengBE := \mathrm{sqrt}\left((x1-x2)^2+(y1-y2)^2\right);$
$k := 1;\ eb[k] := eb[k] + (lengBE/6)\cdot\left(b1(x1,y1)\cdot eval\left(\phi[k],[xb=x1,yb=y1]\right)\right.$
$\quad + 4\cdot b1(xm,ym)\cdot eval\left(\phi[k],[xb=xm,yb=ym]\right)$
$\quad \left. + b1(x2,y2)\cdot eval\left(\phi[k],[xb=x2,yb=y2]\right)\right);$
$k := 2;\ eb[k] := eb[k] + (lengBE/6)\cdot\left(b1(x1,y1)\cdot eval\left(\phi[k],[xb=x1,yb=y1]\right)\right.$
$\quad + 4\cdot b1(xm,ym)\cdot eval\left(\phi[k],[xb=xm,yb=ym]\right)$
$\quad \left. + b1(x2,y2)\cdot eval\left(\phi[k],[xb=x2,yb=y2]\right)\right);$
end if;
if $(conn[2]=2)$ **then**
$x1 := x_2;\ x2 := x_3;\quad y1 := y_2;\ y2 := y_3;$
$xm := (x1+x2)/2;\ \ ym := (y1+y2)/2;$
$lengBE := \mathrm{sqrt}\left((x1-x2)^2+(y1-y2)^2\right);$
$k := 2;\ eb[k] := eb[k] + (lengBE/6)\cdot\left(b2(x1,y1)\cdot eval\left(\phi[k],[xb=x1,yb=y1]\right)\right.$
$\quad + 4\cdot b2(xm,ym)\cdot eval\left(\phi[k],[xb=xm,yb=ym]\right)$
$\quad \left. + b2(x2,y2)\cdot eval\left(\phi[k],[xb=x2,yb=y2]\right)\right);$
$k := 3;\ eb[k] := eb[k] + (lengBE/6)\cdot\left(b2(x1,y1)\cdot eval\left(\phi[k],[xb=x1,yb=y1]\right)\right.$
$\quad + 4\cdot b2(xm,ym)\cdot eval\left(\phi[k],[xb=xm,yb=ym]\right)$
$\quad \left. + b2(x2,y2)\cdot eval\left(\phi[k],[xb=x2,yb=y2]\right)\right);$
end if;
if $(conn[3]=2)$ **then**
$x1 := x_3;\ x2 := x_1;\quad y1 := y_3;\ y2 := y_1;$
$xm := (x1+x2)/2;\ \ ym := (y1+y2)/2;$
$lengBE := \mathrm{sqrt}\left((x1-x2)^2+(y1-y2)^2\right);$
$k := 3;\ eb[k] := eb[k] + (lengBE/6)\cdot\left(b3(x1,y1)\cdot eval\left(\phi[k],[xb=x1,yb=y1]\right)\right.$
$\quad + 4\cdot b3(xm,ym)\cdot eval\left(\phi[k],[xb=xm,yb=ym]\right)$
$\quad \left. + b3(x2,y2)\cdot eval\left(\phi[k],[xb=x2,yb=y2]\right)\right);$
$k := 1;\ eb[k] := eb[k] + (lengBE/6)\cdot\left(b3(x1,y1)\cdot eval\left(\phi[k],[xb=x1,yb=y1]\right)\right.$
$\quad + 4\cdot b3(xm,ym)\cdot eval\left(\phi[k],[xb=xm,yb=ym]\right)$
$\quad \left. + b3(x2,y2)\cdot eval\left(\phi[k],[xb=x2,yb=y2]\right)\right);$

```
    end if;

    return eA, eb;
end proc:
```

$$u := (x, y) \to (x + y - 1)^2 :$$
$$p := (x, y) \to 1 + x :$$
$$q := (x, y) \to 1 + y :$$
$$r := (x, y) \to 0 :$$
$$f := (x, y) \to -6\,x - 6\,y :$$
$$g1 := (x, y) \to (x - 1)^2 :$$
$$g21 := (x, y) \to 2 - 2\,y :$$
$$g22 := (x, y) \to 4\,x :$$
$$g23 := (x, y) \to 4\,y :$$
$$-diff(p(x, y) \cdot diff(u(x, y), x), x) - diff(q(x, y) \cdot diff(u(x, y), y), y) = -6\,x - 6\,y$$

```
with(LinearAlgebra) :
nn := 4 : ne := 2 :
N := Matrix(4, 2, [0, 0, 1, 0, 0, 1, 1, 1])
```

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \textbf{(1)}$$

```
A := Matrix(nn, nn) :  b := Vector(nn) :

for m from 1 to ne do
    eN := N[m ..m + 2, 1 ..2];
    if (m = 1) then
        conn := ⟨1, 0, 2⟩;
        b1 := g1; b2 := 0; b3 := g21;
        gN := ⟨1, 2, 3⟩;
    elif (m = 2) then
        conn := ⟨0, 2, 2⟩;
        b1 := 0; b2 := g22; b3 := g23;
        gN := ⟨2, 3, 4⟩;
    end if;
    eA, eb := LinearGalerkin2D(p, q, r, f, b1, b2, b3, eN, conn) :
    #print(`m,eA,eb=`, m, eA, eb);

   # Add to the Global Matrix
    for i to 3 do
        for j to 3 do
            A[gN[i], gN[j]] := A[gN[i], gN[j]] + eA[i, j];
        end do;
        b[gN[i]] := b[gN[i]] + eb[i];
    end do;
end do:
print(A, b);
```

$$\left[\begin{bmatrix} \frac{4}{3} & -\frac{2}{3} & -\frac{2}{3} & 0 \\ -\frac{2}{3} & \frac{3}{2} & 0 & -\frac{5}{6} \\ -\frac{2}{3} & 0 & \frac{3}{2} & -\frac{5}{6} \\ 0 & -\frac{5}{6} & -\frac{5}{6} & \frac{5}{3} \end{bmatrix}, \begin{bmatrix} \frac{2}{3} \\ -\frac{4}{3} \\ -1 \\ \frac{2}{3} \end{bmatrix}\right] \tag{2}$$

```
A[1, 1] := 1 : A[1, 2] := 0 :  A[1, 3] := 0 : A[1, 4] := 0 :  b[1] := 1 :
A[2, 1] := 0 : A[2, 2] := 1 :  A[2, 3] := 0 : A[2, 4] := 0 :  b[2] := 0 :
```

$$A^{-1}.b$$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \frac{2}{5} \end{bmatrix} \quad \textbf{(3)}$$

$$U := Vector([seq(u(N[i, 1], N[i, 2]), i = 1..4)])$$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \textbf{(4)}$$

Check If the Code is Correct:

```
u  := (x, y) → 1 + y :
p  := (x, y) → 1 :
q  := (x, y) → 1 :
r  := (x, y) → 0 :
f  := (x, y) → 0 :
g1 := (x, y) → 1 :
g21 := (x, y) → 0 :
g22 := (x, y) → 1 :
g23 := (x, y) → 0 :

nn := 4 : ne := 2 :
N := Matrix(4, 2, [0, 0, 1, 0, 0, 1, 1, 1]) :

A := Matrix(nn, nn) : b := Vector(nn) :
for m from 1 to ne do
    eN := N[m ..m + 2, 1 ..2];
    if (m = 1) then
        conn := ⟨1, 0, 2⟩;
        b1 := g1; b2 := 0; b3 := g21;
        gN := ⟨1, 2, 3⟩;
    elif (m = 2) then
        conn := ⟨0, 2, 2⟩;
        b1 := 0; b2 := g22; b3 := g23;
        gN := ⟨2, 3, 4⟩;
    end if;
    eA, eb := LinearGalerkin2D(p, q, r, f, b1, b2, b3, eN, conn) :
    #print(`m,eA,eb=`, m, eA, eb);

   # Add to the Global Matrix
    for i to 3 do
        for j to 3 do
            A[gN[i], gN[j]] := A[gN[i], gN[j]] + eA[i, j];
        end do;
        b[gN[i]] := b[gN[i]] + eb[i];
    end do;
end do:
```

$print(A, b);$

$$\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 1 & 0 & -\frac{1}{2} \\ -\frac{1}{2} & 0 & 1 & -\frac{1}{2} \\ 0 & -\frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} \tag{5}$$

$A[1, 1] := 1 : A[1, 2] := 0 : A[1, 3] := 0 : A[1, 4] := 0 : b[1] := 1 :$
$A[2, 1] := 0 : A[2, 2] := 1 : A[2, 3] := 0 : A[2, 4] := 0 : b[2] := 1 :$

$A^{-1}.b$

$$\begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix} \tag{6}$$

$U := Vector([seq(u(N[i, 1], N[i, 2]), i = 1 ..4)])$

$$\begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix} \tag{7}$$

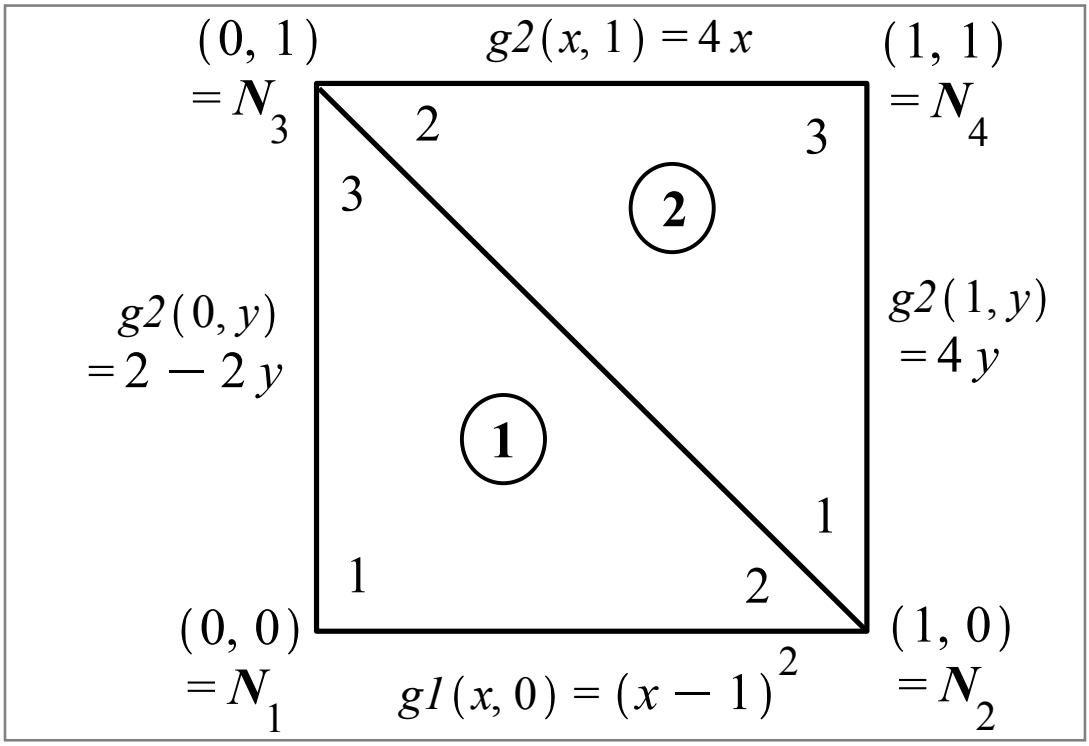
Maple Code:
Linear Galerkin FEM for 2D Elliptic Problems
(Higher-order Quadrature)

)

Example: For $\Omega = [0, 1]^2$, consider the problem

$$-\frac{\partial}{\partial x}\left((1+x)\frac{\partial}{\partial x}u\right) - \frac{\partial}{\partial y}\left((1+y)\frac{\partial}{\partial y}u\right) = -6x - 6y$$

with boundary conditions given as in the figure:



Use the linear Galerkin method using the two elements as given in the figure. Compare the accuracy; the actual solution is $u(x, y) = (x + y - 1)^2$.

Elementary Stiffness Matrix (ESM) for Linear FEM

```
##=========================================
LinearGalerkin2DHQ := proc(p, q, r, f, b1, b2, b3, eN, conn)
  local i, j, k, x, y, xb, yb, xm, ym, detT, area, lengBE;
  local x1, x2, y1, y2, eA, eb, C, ϕ;
  #print(p(x,y), q(x,y), r(x,y), f(x,y), b1(x,y), b2(x,y), b3(x,y), eN, conn);

  x[1] := eN[1,1]; x[2] := eN[2,1]; x[3] := eN[3,1]; x[4] := (x[1]+x[2]+x[3])/3;
  y[1] := eN[1,2]; y[2] := eN[2,2]; y[3] := eN[3,2]; y[4] := (y[1]+y[2]+y[3])/3;

  detT := x[1]·(y[2]-y[3]) + x[2]·(y[3]-y[1]) + x[3]·(y[1]-y[2]);
  area := abs(detT)/2;

 ## Construction of ϕ[i]
  C := 1/detT · [ x[2]y[3]-x[3]y[2]   x[3]y[1]-x[1]y[3]   x[1]y[2]-x[2]y[1] ]
                [ y[2]-y[3]           y[3]-y[1]           y[1]-y[2]         ];
                [ -x[2]+x[3]          -x[3]+x[1]          -x[1]+x[2]        ]
  for i from 1 to 3 do
    ϕ[i] := C[1,i] + C[2,i]·xb + C[3,i]·yb;
  end do;


  ## Elementary Stiffness Matrix "eA" and the corresponding rhs "eb", with higher-
  order quadrature
  eA := Matrix(3,3);
  eb := Vector(3);
  for i to 3 do
    for j to 3 do
      eA[i,j] := (area/12)·add( p(x[k],y[k])·C[2,j]·C[2,i]
                              + q(x[k],y[k])·C[3,j]·C[3,i]
                              + r(x[k],y[k])·eval( ϕ[j]·ϕ[i], [xb=x[k], yb=y[k]] ), k=1..3 );
      eA[i,j] := eA[i,j] + (area·9/12)·( p(x[4],y[4])·C[2,j]·C[2,i]
```

$$+ q\left(x_4, y_4\right) \cdot C[3,j] \cdot C[3,i]$$
$$+ r\left(x_4, y_4\right) \cdot eval\left(\phi_j \cdot \phi_i, \left[xb = x_4, yb = y_4\right]\right)\Big);$$

end do;

$$eb[i] := (area/12) \cdot add\left(f\left(x_k, y_k\right) \cdot eval\left(\phi_i, \left[xb = x_k, yb = y_k\right]\right), k = 1..3\right);$$

$$eb[i] := eb[i] + (area \cdot 9/12) \cdot \left(f\left(x_4, y_4\right) \cdot eval\left(\phi_i, \left[xb = x_4, yb = y_4\right]\right)\right);$$

end do;

Neumann Boundary Condition

if $(conn[1] = 2)$ **then**

$$x1 := x_1;\ x2 := x_2;\quad y1 := y_1;\ y2 := y_2;$$

$$xm := (x1 + x2)/2;\ \ ym := (y1 + y2)/2;$$

$$lengBE := \text{sqrt}\left((x1 - x2)^2 + (y1 - y2)^2\right);$$

$$k := 1;\ eb[k] := eb[k] + (lengBE/6) \cdot \left(b1(x1, y1) \cdot eval\left(\phi[k], [xb = x1, yb = y1]\right)\right.$$
$$+ 4 \cdot b1(xm, ym) \cdot eval\left(\phi[k], [xb = xm, yb = ym]\right)$$
$$\left. + b1(x2, y2) \cdot eval\left(\phi[k], [xb = x2, yb = y2]\right)\right);$$

$$k := 2;\ eb[k] := eb[k] + (lengBE/6) \cdot \left(b1(x1, y1) \cdot eval\left(\phi[k], [xb = x1, yb = y1]\right)\right.$$
$$+ 4 \cdot b1(xm, ym) \cdot eval\left(\phi[k], [xb = xm, yb = ym]\right)$$
$$\left. + b1(x2, y2) \cdot eval\left(\phi[k], [xb = x2, yb = y2]\right)\right);$$

end if;

if $(conn[2] = 2)$ **then**

$$x1 := x_2;\ x2 := x_3;\quad y1 := y_2;\ y2 := y_3;$$

$$xm := (x1 + x2)/2;\ \ ym := (y1 + y2)/2;$$

$$lengBE := \text{sqrt}\left((x1 - x2)^2 + (y1 - y2)^2\right);$$

$$k := 2;\ eb[k] := eb[k] + (lengBE/6) \cdot \left(b2(x1, y1) \cdot eval\left(\phi[k], [xb = x1, yb = y1]\right)\right.$$
$$+ 4 \cdot b2(xm, ym) \cdot eval\left(\phi[k], [xb = xm, yb = ym]\right)$$
$$\left. + b2(x2, y2) \cdot eval\left(\phi[k], [xb = x2, yb = y2]\right)\right);$$

$$k := 3;\ eb[k] := eb[k] + (lengBE/6) \cdot \left(b2(x1, y1) \cdot eval\left(\phi[k], [xb = x1, yb = y1]\right)\right.$$
$$+ 4 \cdot b2(xm, ym) \cdot eval\left(\phi[k], [xb = xm, yb = ym]\right)$$
$$\left. + b2(x2, y2) \cdot eval\left(\phi[k], [xb = x2, yb = y2]\right)\right);$$

end if;

if $(conn[3] = 2)$ **then**

$$x1 := x_3;\ x2 := x_1;\quad y1 := y_3;\ y2 := y_1;$$

$$xm := (x1 + x2)/2;\ \ ym := (y1 + y2)/2;$$

$$lengBE := \text{sqrt}\left((x1 - x2)^2 + (y1 - y2)^2\right);$$

```
        k := 3; eb[k] := eb[k] + (lengBE/6)·(b3(x1, y1)·eval(ϕ[k], [xb=x1, yb=y1])
                    + 4·b3(xm, ym)·eval(ϕ[k], [xb=xm, yb=ym])
                    + b3(x2, y2)·eval(ϕ[k], [xb=x2, yb=y2]));
        k := 1; eb[k] := eb[k] + (lengBE/6)·(b3(x1, y1)·eval(ϕ[k], [xb=x1, yb=y1])
                    + 4·b3(xm, ym)·eval(ϕ[k], [xb=xm, yb=ym])
                    + b3(x2, y2)·eval(ϕ[k], [xb=x2, yb=y2]));
    end if;

    return eA, eb;
end proc:
```

$$u := (x, y) \to (x + y - 1)^2 :$$
$$p := (x, y) \to 1 + x :$$
$$q := (x, y) \to 1 + y :$$
$$r := (x, y) \to 0 :$$
$$f := (x, y) \to -6\,x - 6\,y :$$
$$g1 := (x, y) \to (x - 1)^2 :$$
$$g21 := (x, y) \to 2 - 2\,y :$$
$$g22 := (x, y) \to 4\,x :$$
$$g23 := (x, y) \to 4\,y :$$

```
with(LinearAlgebra) :
nn := 4 : ne := 2 :
N := Matrix(4, 2, [0, 0, 1, 0, 0, 1, 1, 1])
```

$$N = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \textbf{(1)}$$

```
A := Matrix(nn, nn) :  b := Vector(nn) :

for m from 1 to ne do
    eN := N[m..m + 2, 1..2];
    if (m = 1) then
```

```
    conn := ⟨1, 0, 2⟩;
    b1 := g1; b2 := 0; b3 := g21;
    gN := ⟨1, 2, 3⟩;
  elif (m = 2) then
    conn := ⟨0, 2, 2⟩;
    b1 := 0; b2 := g22; b3 := g23;
    gN := ⟨2, 3, 4⟩;
  end if;
  eA, eb := LinearGalerkin2DHQ(p, q, r, f, b1, b2, b3, eN, conn) :
  #print(`m,eA,eb=`, m, eA, eb);

 # Add to the Global Matrix
  for i to 3 do
    for j to 3 do
      A[gN[i], gN[j]] := A[gN[i], gN[j]] + eA[i, j];
    end do;
    b[gN[i]] := b[gN[i]] + eb[i];
  end do;
end do:
print(A, b);
```

$$\left[\begin{bmatrix} \frac{4}{3} & -\frac{2}{3} & -\frac{2}{3} & 0 \\ -\frac{2}{3} & \frac{3}{2} & 0 & -\frac{5}{6} \\ -\frac{2}{3} & 0 & \frac{3}{2} & -\frac{5}{6} \\ 0 & -\frac{5}{6} & -\frac{5}{6} & \frac{5}{3} \end{bmatrix}, \begin{bmatrix} \frac{1}{6} \\ -\frac{4}{3} \\ -1 \\ \frac{7}{6} \end{bmatrix}\right] \tag{2}$$

```
A[1, 1] := 1 : A[1, 2] := 0 : A[1, 3] := 0 : A[1, 4] := 0 : b[1] := 1 :
A[2, 1] := 0 : A[2, 2] := 1 : A[2, 3] := 0 : A[2, 4] := 0 : b[2] := 0 :

evalf(A^(-1).b)
```

$$\begin{bmatrix} 1. \\ 0. \\ 0.2307692308 \\ 0.8153846154 \end{bmatrix} \quad \textbf{(3)}$$

$$U := Vector([seq(u(N[i, 1], N[i, 2]), i = 1..4)])$$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \textbf{(4)}$$

Check If the Code is Correct:

```
u  := (x, y) → 1 + y :
p  := (x, y) → 1 :
q  := (x, y) → 1 :
r  := (x, y) → 0 :
f  := (x, y) → 0 :
g1 := (x, y) → 1 :
g21 := (x, y) → 0 :
g22 := (x, y) → 1 :
g23 := (x, y) → 0 :

nn := 4 : ne := 2 :
N := Matrix(4, 2, [0, 0, 1, 0, 0, 1, 1, 1]) :

A := Matrix(nn, nn) : b := Vector(nn) :
for m from 1 to ne do
    eN := N[m ..m + 2, 1 ..2];
    if (m = 1) then
        conn := ⟨1, 0, 2⟩;
        b1 := g1; b2 := 0; b3 := g21;
        gN := ⟨1, 2, 3⟩;
    elif (m = 2) then
        conn := ⟨0, 2, 2⟩;
        b1 := 0; b2 := g22; b3 := g23;
        gN := ⟨2, 3, 4⟩;
    end if;
    eA, eb := LinearGalerkin2DHQ(p, q, r, f, b1, b2, b3, eN, conn) :
    #print(`m,eA,eb=`, m, eA, eb);

   # Add to the Global Matrix
    for i to 3 do
        for j to 3 do
            A[gN[i], gN[j]] := A[gN[i], gN[j]] + eA[i, j];
        end do;
        b[gN[i]] := b[gN[i]] + eb[i];
    end do;
end do:
```

$print(A, b);$

$$\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 1 & 0 & -\frac{1}{2} \\ -\frac{1}{2} & 0 & 1 & -\frac{1}{2} \\ 0 & -\frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} \tag{5}$$

$A[1, 1] := 1 : A[1, 2] := 0 : A[1, 3] := 0 : A[1, 4] := 0 : b[1] := 1 :$
$A[2, 1] := 0 : A[2, 2] := 1 : A[2, 3] := 0 : A[2, 4] := 0 : b[2] := 1 :$

$A^{-1}.b$

$$\begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix} \tag{6}$$

$U := Vector([seq(u(N[i, 1], N[i, 2]), i = 1 ..4)])$

$$\begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \end{bmatrix} \tag{7}$$

Homework:
12. Numerical Solutions to Partial Differential Equations

You may consider this last homework as another project.

Problem #2 is optional, and you may get a headache when you decide to try it. However, if you can get through the maze, you clearly deserve an ***A*** *for the class and you do not have to take the final.*

#1. Use the **Crank-Nicolson method** to approximate the solution to

$$\frac{\partial}{\partial t}\,u(x,\,t) - \frac{\partial^2}{\partial x^2}\,u(x,\,t) = 2, \quad (x,\,t) \in [0,\,1/2] \times [0,\,1/2]$$

$$u(0,\,t) = 0, \quad u_x(1/2,\,t) = 0, \quad u(x,\,0) = \sin(\pi x) - x^2 + x$$

a. Modify the code for the Neumann boundary condition.
b. Carry out the computation with $n_x = 10, n_t = 20$ and $n_x = 20, n_t = 40$
c. Compute the Richardson extrapolation for the solutions at $t = T = 1/2$ and verify accuracy improvement; the actual solution is

$$u(x,\,t) = \mathrm{e}^{-\pi^2 t}\sin(\pi x) - x^2 + x.$$

#2 (optional). Let $((x, y), t) \in R \times [0, T] = [0, 1]^2 \times [0, 1]$.
Modify the **CN-ADI** code to approximate the solution to

$$u_t - u_{xx} - u_{yy} = f, \quad f = 0$$

$$u_0(x, y) = \begin{cases} 1, & x = y = 0.5 \\ 0, & \text{otherwise} \end{cases}$$

$$u_\nu(x, y, t) = 0, \quad ((x, y), t) \in \partial R \times [0, T]$$

with $n_x = n_y = 10$, 20, and 40. Note that the problem involves the no-flux BC; you have to modify the code appropriately.

a. Verify that your modifications are correct, by setting an example of smooth solution which satisfies the no-flux boundary condition.

For example:

$u := \cos(\pi x) - \cos(\pi y) + \sin(\pi t)$:

$diff(u, x)$

$$-\sin(\pi x)\,\pi \tag{2.1}$$

$diff(u, y)$

$$\sin(\pi y)\,\pi \tag{2.2}$$

$f := diff(u, t) - diff(u, x, x) - diff(u, y, y)$

$$\cos(\pi t)\,\pi + \cos(\pi x)\,\pi^2 - \cos(\pi y)\,\pi^2 \tag{2.3}$$

$u0 := eval(u, t = 0)$

$$\cos(\pi x) - \cos(\pi y) \tag{2.4}$$

Clearly, **(2.1)** and **(2.2)** satisfy the no-flux BC on the boundary. For the verification, you have to use **(2.3)** and **(2.4)** for f and u_0, respectively.

b. For each case of spatial grid sizes, find n_t such that the solution to the point-source problem is not oscillatory.