

Deep Neural Networks and Finite Elements

In this chapter, we will give a brief introduction to a special function class related to deep neural networks (DNN) used in machine learning. We then explore the relationship between DNN (with ReLU as activation function) and linear finite element methods.

21.1 Introduction

Given $n, m \geq 1$, the first ingredient in defining a deep neural network (DNN) is (vector) linear functions of the form

$$(21.1) \quad \Theta : \mathbb{R}^n \rightarrow \mathbb{R}^m,$$

as $\Theta(x) = Wx + b$ where $W = (w_{ij}) \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$.

The second main ingredient is a nonlinear activation function, usually denoted as

$$(21.2) \quad \sigma : \mathbb{R} \rightarrow \mathbb{R}.$$

By applying the function to each component, we can extend this naturally to

$$\sigma : \mathbb{R}^n \mapsto \mathbb{R}^n.$$

Given $d, c, k \in \mathbb{N}^+$ and

$$n_1, \dots, n_k \in \mathbb{N} \text{ with } n_0 = d, n_{k+1} = c,$$

a general DNN from \mathbb{R}^d to \mathbb{R}^c is given by

$$\begin{aligned} f(x) &= f^k(x), \\ f^\ell(x) &= [\Theta^\ell \circ \sigma](f^{\ell-1}(x)) \quad \ell = 1 : k, \end{aligned}$$

with $f^0(x) = \Theta(x)$.

The following more concise notation is often used in computer science literature:

$$(21.3) \quad f(x) = \Theta^k \circ \sigma \circ \Theta^{k-1} \circ \sigma \cdots \circ \Theta^1 \circ \sigma \circ \Theta^0(x),$$

here $\Theta^i : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^{n_{i+1}}$ are linear functions as defined in (21.1). Such a DNN is called a $(k + 1)$ -layer DNN, and is said to have k hidden layers. The size of this DNN is $n_1 + \cdots + n_k$.

In this chapter, we mainly consider a special activation function, known as the *rectified linear unit* (ReLU), and defined as $\text{ReLU} : \mathbb{R} \mapsto \mathbb{R}$,

$$(21.4) \quad \text{ReLU}(x) = \max(0, x), \quad x \in \mathbb{R}.$$

A ReLU DNN with k hidden layers might be written as:

$$(21.5) \quad f(x) = \Theta^k \circ \text{ReLU} \circ \Theta^{k-1} \circ \text{ReLU} \cdots \circ \Theta^1 \circ \text{ReLU} \circ \Theta^0(x).$$

We note that ReLU is a continuous piecewise linear (CPWL) function. Since the composition of two CPWL functions is still a CPWL function, we have the following observation[?].

Lemma 129. *Every ReLU DNN: $\mathbb{R}^d \rightarrow \mathbb{R}^c$ is a continuous piecewise linear function. More specifically, given any ReLU DNN, there is a polyhedral decomposition of \mathbb{R}^d such that this ReLU DNN is linear on each polyhedron in such a decomposition.*

Here is a simple example for the “grid” created by some 2-layer ReLU DNNs in \mathbb{R}^2 .

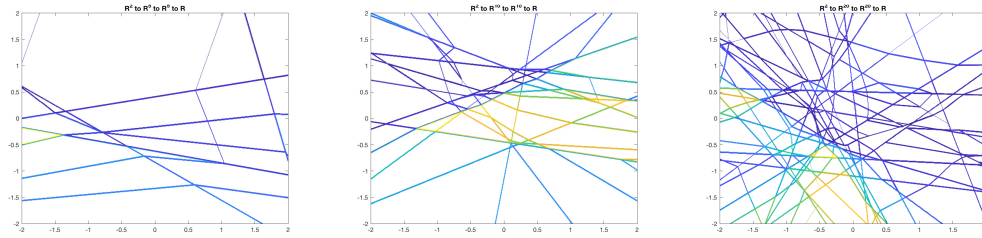


Fig. 21.1. Projections of the domain partitions formed by 2-layer ReLU DNNs with sizes $(n_0, n_1, n_2, n_3) = (2, 5, 5, 1), (2, 10, 10, 1)$ and $(2, 20, 20, 1)$ with random parameters.

For convenience of exposition, we introduce the following notation:

$$(21.6) \quad \text{DNN}_J := \{f : f = \Theta^J \circ \text{ReLU} \circ \Theta^{J-1} \cdots \circ \text{ReLU} \circ \Theta^0(x), \\ \Theta^\ell \in \mathbb{R}^{n^{\ell+1} \times (n^\ell+1)}, \quad n^0 = d, \quad n^{J+1} = 1, \quad n^\ell \in \mathbb{N}^+\}.$$

Namely DNN_J represents the DNN model with J hidden layers and ReLU activation function with arbitrary size.

21.2 Linear finite element as a DNN

In this section, we consider a special CPWL function space, namely the space of linear simplicial finite element functions.

21.2.1 Notation

Recall in Chapter 2, we introduce the finite element spaces. Assuming that $\mathcal{Q} \subset \mathbb{R}^d$ is a bounded domain. Given a simplicial finite element grid \mathcal{T}_h consisting of a set of simplexes $\{\tau_k\}$ and the corresponding set of nodal points \mathcal{N}_h , the finite element space is given by

$$(21.7) \quad V_h = \{v \mid v \in C(\overline{\mathcal{Q}}), \text{ and } v|_{\tau_k} \in P_1(\tau_k), \forall \tau_k \in \mathcal{T}_h\}.$$

Obviously any $v \in V_h$ can be uniquely represented in terms of these nodal basis functions:

$$(21.8) \quad v(x) = \sum_{i=1}^N v_i \varphi_i(x),$$

where N is the degrees of freedom and $\{\varphi_i, i = 1, \dots, N\}$ are the standard nodal basis functions as defined in previous chapters.

Given $x_i \in \mathcal{N}_h$, let $N(i)$ denote all the indices j such that τ_j contains the nodal point x_i , namely

$$N(i) = \{j : x_i \in \tau_j\},$$

and k_h denote the maximum number of neighboring elements in the grid

$$(21.9) \quad k_h = \max_{x_i \in \mathcal{N}_h} |N(i)|.$$

Let $G(i)$ denote the support of the nodal basis φ_i :

$$G(i) = \bigcup_{k \in N(i)} \tau_k.$$

We say that the grid \mathcal{T}_h is locally convex if $G(i)$ is convex for each i .

21.2.2 DNN representation of finite element functions

As an illustration, we will now demonstrate how a linear finite element function associated with a locally convex grid \mathcal{T}_h can be represented by a ReLU DNN.

Thanks to (21.8), it suffices to show that each basis function φ_i can be represented by a ReLU DNN. We first note that the case where $d = 1$ is trivial as the basis function φ_i with support in $[x_{i-1}, x_{i+1}]$ can be easily written as

$$(21.10) \quad \varphi_i(x) = \frac{1}{h_{i-1}} \text{ReLU}(x - x_{i-1}) - \left(\frac{1}{h_{i-1}} + \frac{1}{h_i}\right) \text{ReLU}(x - x_i) + \frac{1}{h_i} \text{ReLU}(x - x_{i+1}),$$

where $h_i = x_{i+1} - x_i$.

In order to consider the cases where $d > 1$, we first prove the following lemma.

Lemma 130. *Given $x_i \in \mathcal{N}_h$, if $G(i)$ is convex, then the corresponding basis function can be written as*

$$(21.11) \quad \varphi_i(x) = \max \left\{ 0, \min_{k \in N(i)} g_k(x) \right\},$$

where, for each $k \in N(i)$, g_k is the global linear function such that $g_k = \varphi_i$ on τ_k .

Proof. To show (21.11) holds for all $x \in \mathbb{R}^d$, we first consider the case $x \in G(i)$, namely $x \in \tau_{k_0}$ for some $k_0 \in N(i)$. Thus

$$(21.12) \quad \varphi_i(x) = g_{k_0}(x) \geq 0.$$

Let P_k be the hyperplane that passes through the $d - 1$ subsimplex (of τ_k) that does not contain x_i (see the left figure in Figure 21.2). Since $G(i)$ is convex by assumption, all points in τ_{k_0} should be on the same side of the hyperplane P_k . As a result, for all $k \in N(i)$,

$$g_k(y) \geq 0 \equiv g_{k_0}(y), \quad y \in P_k \cap \tau_{k_0}.$$

By combining the above inequality with the following obvious inequality that

$$g_k(x_i) = 1 \geq 1 = g_{k_0}(x_i), \quad k \in N(i),$$

and the fact that all g_k are linear, we conclude that

$$g_k(y) \geq g_{k_0}(y), \quad \forall y \in \tau_{k_0}, k \in N(i).$$

In particular

$$g_k(x) \geq g_{k_0}(x), \quad k \in N(i).$$

This, together with (21.12), proves that (21.11) holds for all $x \in G(i)$. Thus

$$(21.13) \quad \max \left\{ 0, \min_{k \in N(i)} g_k(x) \right\} = g_{k_0}(x).$$

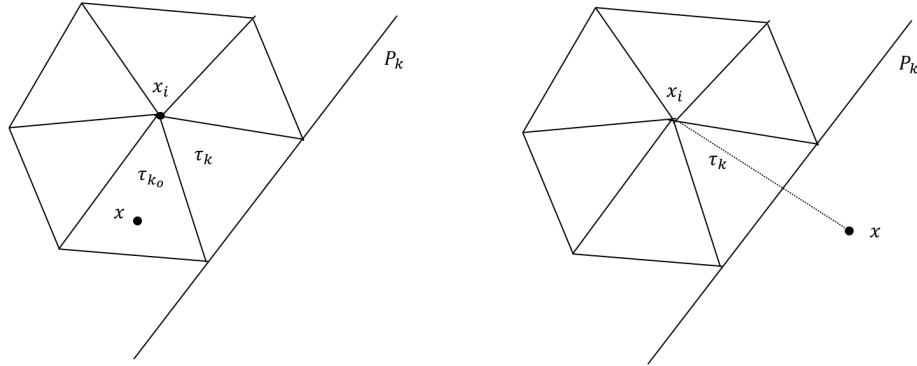


Fig. 21.2. Left: $x \in G(i)$, right: $x \notin G(i)$

On the other hand, if $x \notin G(i)$, there exists a $\tau_k \subset G(i)$ such that τ_k contains a segment of the straight line that pass through x and x_i (see the right figure in Figure 21.2). Again let P_k be the hyperplane associated with τ_k as defined above. We note that x and x_i are on the different sides of P_k . Since

$$g_k(x_i) \geq 0, \quad g_k(y) = 0, \quad y \in P_k,$$

we then have

$$\min_{k \in N(i)} g_k(x) \leq g_k(x) \leq 0,$$

which implies

$$\max \left\{ 0, \min_{k \in N(i)} g_k(x) \right\} = 0 = \varphi_i(x), \quad x \notin G(i).$$

This finishes the proof of Lemma 130. \square

Theorem 130. *Given a locally convex finite element grid \mathcal{T}_h , any linear finite element function with N degrees of freedom, can be written as a ReLU-DNN with at most $k = \lceil \log_2 k_h \rceil + 1$ hidden layers and at most $O(k_h N)$ number of the neurons.*

Proof. We have the following identity,

$$(21.14) \quad \min\{a, b\} = \frac{a+b}{2} - \frac{|a-b|}{2} = v \cdot \text{ReLU}(W \cdot [a, b]^T),$$

where

$$v = \frac{1}{2}[1, -1, -1, -1], \quad W = \begin{bmatrix} 1 & 1 \\ -1 & -1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}.$$

By Lemma 130, the basis function $\varphi_i(x)$ can be written as:

$$\varphi_i(x) = \max \left\{ 0, \min_{k \in N(i)} g_k(x) \right\}.$$

For convenience, we assume that

$$N(i) = \{r_1, r_2, \dots, r_{|N(i)|}\}.$$

Then we have

$$\begin{aligned} \min_{k \in N(i)} g_k(x) &= \min \{g_{r_1}(x), \dots, g_{r_{|N(i)|}}(x)\} \\ &= \min \left\{ \min\{g_{r_1}, \dots, g_{r_{\lceil |N(i)|/2 \rceil}}\}, \min\{g_{r_{\lceil |N(i)|/2 \rceil+1}}, \dots, g_{r_{|N(i)|}}\} \right\} \\ &= v \cdot \text{ReLU} \left(W \cdot \begin{bmatrix} \min\{g_{r_1}, \dots, g_{r_{\lceil |N(i)|/2 \rceil}}\} \\ \min\{g_{r_{\lceil |N(i)|/2 \rceil+1}}, \dots, g_{r_{|N(i)|}}\} \end{bmatrix} \right). \end{aligned}$$

According to this procedure, we get the minimum of $|N(i)|$ terms by splitting them in two, each taking the minimum over at most $\lceil |N(i)|/2 \rceil$ terms. This contributes to one ReLU hidden layer. Then we can further split the terms

$$\min\{g_{r_1}, \dots, g_{r_{\lceil |N(i)|/2 \rceil}}\}, \quad \min\{g_{r_{\lceil |N(i)|/2 \rceil+1}}, \dots, g_{r_{|N(i)|}}\}$$

until all the minimum functions contain only 1 or 2 terms.

1. If there is one term

$$\min\{a\} = a.$$

2. If there are two terms

$$\min\{a, b\} = v \cdot \text{ReLU}(W \cdot [a, b]^T).$$

which is also a ReLU DNN with 1 hidden layer. So we can write a basis function as a $1 + \lceil \log_2 k_h \rceil$ -hidden-layer DNN. Considering the binary-tree structure, a k -layer full binary-tree has $2^k - 1$ nodes. We can see the number of neurons is at most

$$\mathcal{O}(2^k) = \mathcal{O}(2^{1+\lceil \log_2 k_h \rceil}) = \mathcal{O}(k_h).$$

By (21.8), the piecewise linear function can be represented as a DNN with $k = 1 + \lceil \log_2 k_h \rceil$ hidden layers. The number of neurons is at most $\mathcal{O}(k_h N)$. \square

Example 12. Let us consider a 2D uniform grid.

Define global linear functions g_i such that

$$g_i(x_0) = 1 \quad g_i(x_i) = 0 \quad g_i(x_{i+1}) = 0,$$

here g_i is linear in Domain i , and $x_7 = x_1$.

For example, if $g_1 = W_1 x + b_1$, with $x_0 = (0, 0)$, $x_1 = (-1, 0)$ and $x_2 = (0, 1)$ then

$$g_1 = x - y + 1 = [1, -1] \begin{bmatrix} x \\ y \end{bmatrix} + 1$$

The basis function φ is

$$\varphi(x) = \begin{cases} g_i(x), & x \in \text{Domain } i \\ 0, & x \in \mathbb{R}^2 - \overline{x_1 x_2 x_3 x_4 x_5 x_6} \end{cases}$$

Then by Lemma 130 we have

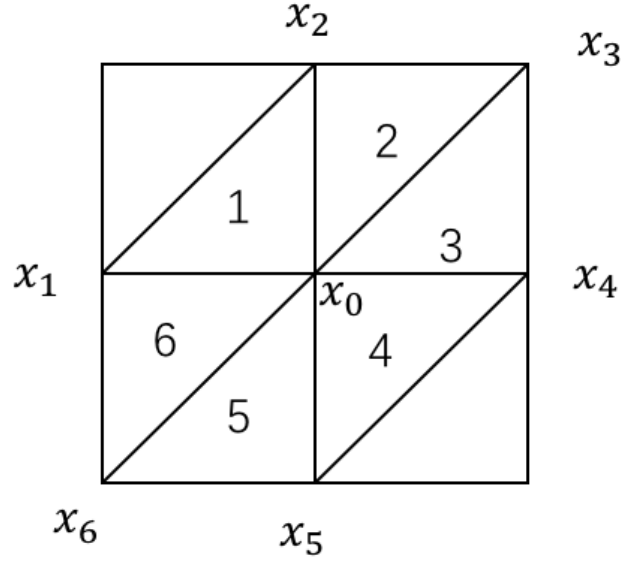


Fig. 21.3. 2D uniform FEM basis function

$$\varphi(x) = \max(0, \min(g_1, g_2, g_3, g_4, g_5, g_6)).$$

Use (21.14) we can get

$$\begin{aligned} \min(a, b, c) &= \min(\min(a, b), c) \\ &= v \cdot \text{Relu}\left(W \cdot \begin{pmatrix} \min(a, b) \\ c \end{pmatrix}\right), \\ &= v \cdot \text{Relu}\left(W \cdot \begin{pmatrix} v \cdot \text{Relu}(W \cdot [a, b]^T) \\ \text{Relu}(c) - \text{Relu}(-c) \end{pmatrix}\right), \\ &= v \cdot \text{Relu}\left(W \cdot \begin{pmatrix} v \cdot \text{Relu}(W \cdot [a, b]^T) \\ [1, -1] \cdot \text{Relu}([1, -1]^T c) \end{pmatrix}\right), \\ &= v \cdot \text{Relu}(W_2 \cdot \text{Relu}(W_1 \cdot [a, b, c]^T)), \end{aligned}$$

where

$$W_2 = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & 1 & -1 \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -1 & 1 \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -1 & 1 \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 & -1 \end{bmatrix}, \quad W_1 = \begin{bmatrix} 1 & 1 & 0 \\ -1 & -1 & 0 \\ 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix}.$$

It follows that

$$\begin{aligned}
 g &\equiv \min(g_1, g_2, g_3, g_4, g_5, g_6) = \min(\min(g_1, g_2, g_3), \min(g_4, g_5, g_6)) \\
 &= v \cdot \text{Relu}\left(W \cdot \begin{bmatrix} \min(g_1, g_2, g_3) \\ \min(g_4, g_5, g_6) \end{bmatrix}\right) \\
 &= v \cdot \text{Relu}\left(W \cdot \begin{bmatrix} v \cdot \text{Relu}(W_2 \cdot \text{Relu}(W_1 \cdot [g_1, g_2, g_3]^T)) \\ v \cdot \text{Relu}(W_2 \cdot \text{Relu}(W_1 \cdot [g_4, g_5, g_6]^T)) \end{bmatrix}\right)
 \end{aligned}$$

Thus

$$\varphi = \max(0, g) = \text{Relu}(g).$$

and φ can be represented by a ReLU DNN with 4 hidden layers, which is exactly $1 + \lceil \log_2 6 \rceil$.

We now consider a special class of the so-called shape regular finite element grid \mathcal{T}_h which satisfies

$$(21.15) \quad \kappa_1 \leq \frac{r_\tau}{R_\tau} \leq \kappa_2, \quad \forall \tau \in \mathcal{T}_h,$$

for some constants κ_1 and κ_2 independent of h and d , where r_τ (R_τ) is the radius of the largest (smallest) ball contained in (containing) τ .

Corollary 11. *Given a locally convex and shape regular finite element grid \mathcal{T}_h , any linear finite element function with N degrees of freedom (DOFs), can be written as a ReLU-DNN with at most $O(d)$ hidden layers. The number of neurons is at most $O(\kappa^d N)$ for some constant $\kappa \geq 2$ depending on the shape-regularity of \mathcal{T}_h . The number of non-zero parameters is at most $O(d\kappa^d N)$.*

We note that, using the approach described in this section, a finite element function with N DOFs can be represented by a DNN with $O(N)$ number of weights. This property is expected to be useful when DNNs are used in adaptive mesh-less or vertex-less numerical discretization methods for partial differential equation, which is a subject of further study.

21.3 Linear finite element cannot be recovered by DNN_1 for $d \geq 2$

In the previous section, we show that a finite element function can be represented by a ReLU DNN with $\log_2 k_h + 1$ hidden layers.

In view of Lemma 129 and the fact that $\text{DNN}_J \subseteq \text{DNN}_{J+1}$, it is natural to ask that how many layers are needed at least to recover all linear finite element functions in \mathbb{R}^d . In this section, we will show that

$$(21.16) \quad J_d \geq 2, \quad \text{if } d \geq 2,$$

where J_d is the minimal J such that all linear finite element functions in \mathbb{R}^d can be recovered by DNN_J .

In particular, we will show the following theorem.

Theorem 131. *If $\Omega \subset \mathbb{R}^d$ is either a bounded domain or $\Omega = \mathbb{R}^d$, DNN_1 can not be used to recover all linear finite element functions in Ω .*

Proof. We prove it by contradiction. Let us assume that for any continuous piecewise linear function $f : \Omega \rightarrow \mathbb{R}$, we can find finite $N \in \mathbb{N}$, $w_i \in \mathbb{R}^{1,d}$ as row vector and $\alpha_i, b_i, \beta \in \mathbb{R}$ such that

$$f = \sum_{i=1}^N \alpha_i \text{ReLU}(w_i x + b_i) + \beta,$$

with $f_i = \alpha_i \text{ReLU}(w_i x + b_i)$, $\alpha_i \neq 0$ and $w_i \neq 0$. Consider the finite element functions, if this one hidden layer ReLU DNN can recover any basis function of FEM, then it can recover the finite element space. Thus let us assume f is a locally supported basis function for FEM. Furthermore, if Ω is a bounded domain, we assume that

$$(21.17) \quad d(\text{supp}(f), \partial\Omega) > 0,$$

with

$$d(A, B) = \inf_{x \in A, y \in B} \|x - y\|,$$

as the distance of two closed sets.

A more important observation is that $\nabla f : \Omega \rightarrow \mathbb{R}^d$ is a piecewise constant vector function. The key point is to consider the discontinuous points for $g := \nabla f = \sum_{i=1}^N \nabla f_i$.

For more general case, we can define the set of discontinuous points of a function by

$$D_g := \{x \in \Omega \mid x \text{ is a discontinuous point of } g\}.$$

Because of the property that

$$(21.18) \quad D_{f+g} \supseteq D_f \cup D_g \setminus (D_f \cap D_g),$$

we have

$$(21.19) \quad D_{\sum_{i=1}^N g_i} \supseteq \bigcup_{i=1}^N D_{g_i} \setminus \bigcup_{i \neq j} (D_{g_i} \cap D_{g_j}).$$

Note that

$$(21.20) \quad g_i = \nabla f_i(x) = \nabla (\alpha_i \text{ReLU}(w_i x + b_i)) = (\alpha_i H(w_i x + b_i)) w_i \in \mathbb{R}^d,$$

for $i = 1 : N$ with H be the Heaviside function defined as:

$$H(x) = \begin{cases} 0 & \text{if } x \leq 0, \\ 1 & \text{if } x > 0. \end{cases}$$

This means that

$$(21.21) \quad D_{g_i} = \{x \mid w_i x + b_i = 0\}$$

is a $d - 1$ dimensional affine space in \mathbb{R}^d .

Without loss of generality, we can assume that

$$(21.22) \quad D_{g_i} \neq D_{g_j}.$$

When the other case occurs, i.e. $D_{g_{\ell_1}} = D_{g_{\ell_2}} = \dots = D_{g_{\ell_k}}$, by the definition of g_i in (21.20) and D_{g_i} in (21.21), this happens if and only if there is a row vector (w, b) such that

$$(21.23) \quad c_{\ell_i} (w \ b) = (w_{\ell_i} \ b_{\ell_i}),$$

with some $c_{\ell_i} \neq 0$ for $i = 1 : k$. We combine those g_{ℓ_i} as

$$\begin{aligned} \tilde{g}_\ell &= \sum_{i=1}^k g_{\ell_i} = \sum_{i=1}^k \alpha_{\ell_i} H(w_{\ell_i} x + b_{\ell_i}) w_{\ell_i}, \\ &= \sum_{i=1}^k (c_{\ell_i} \alpha_{\ell_i} H(c_{\ell_i} (wx + b))) w, \\ &= \begin{cases} \left(\sum_{i=1}^k c_{\ell_i} \alpha_{\ell_i} H(c_{\ell_i}) \right) w & \text{if } wx + b > 0, \\ \left(\sum_{i=1}^k c_{\ell_i} \alpha_{\ell_i} H(-c_{\ell_i}) \right) w & \text{if } wx + b \leq 0. \end{cases} \end{aligned}$$

Thus, if

$$\left(\sum_{i=1}^k c_{\ell_i} \alpha_{\ell_i} H(c_{\ell_i}) \right) = \left(\sum_{i=1}^k c_{\ell_i} \alpha_{\ell_i} H(-c_{\ell_i}) \right),$$

\tilde{g}_ℓ is a constant vector function, that is to say $D_{\sum_{i=1}^k g_{\ell_i}} = D_{\tilde{g}_\ell} = \emptyset$. Otherwise, \tilde{g}_ℓ is a piecewise constant vector function with the property that

$$D_{\sum_{i=1}^k g_{\ell_i}} = D_{\tilde{g}_\ell} = D_{g_{\ell_i}} = \{x \mid wx + b = 0\}.$$

This means that we can use condition (21.23) as an equivalence relation and split $\{g_i\}_{i=1}^N$ into some groups, and we can combine those g_{ℓ_i} in each group as what we do above. After that, we have

$$\sum_{i=1}^N g_i = \sum_{\ell=1}^{\tilde{N}} \tilde{g}_\ell,$$

with $D_{\tilde{g}_s} \neq D_{\tilde{g}_t}$. Finally, we can have that $D_{\tilde{g}_s} \cap D_{\tilde{g}_t}$ is an empty set or a $d-2$ dimensional affine space in \mathbb{R}^d . Since $\tilde{N} \leq N$ is a finite number,

$$D := \bigcup_{i=1}^N D_{g_i} \setminus \bigcup_{s \neq t} (D_{\tilde{g}_s} \cap D_{\tilde{g}_t})$$

is an unbounded set.

- If $\Omega = \mathbb{R}^d$,

$$\text{supp}(f) \supseteq D_g = D_{\sum_{i=1}^N g_i} = D_{\sum_{\ell=1}^{\tilde{N}} \tilde{g}_\ell} \supseteq D,$$

is contradictory to the assumption that f is locally supported.

- If Ω is a bounded domain,

$$d(D, \partial\Omega) = \begin{cases} s > 0 & \text{if } D_{\tilde{g}_i} \cap \Omega = \emptyset, \forall i \\ 0 & \text{otherwise.} \end{cases}$$

Note again that all $D_{\tilde{g}_i}$'s are $d-1$ dimensional affine spaces, while $D_{\tilde{g}_i} \cap D_{\tilde{g}_j}$ is either an empty set or a $d-2$ dimensional affine space. If $d(D, \partial\Omega) > 0$, this implies that ∇f is continuous in Ω , which contradicts the assumption that f is a basis function in FEM. If $d(D, \partial\Omega) = 0$, this contradicts the previous assumption in (21.17).

Hence DNN₁ cannot recover any piecewise linear function in Ω for $d \geq 2$. \square

Following the proof above, we have the following theorem.

Theorem 132. $\{\text{ReLU}(w_i x + b_i)\}_{i=1}^m$ are linearly independent if (w_i, b_i) and (w_j, b_j) are linearly independent in $\mathbb{R}^{1 \times (d+1)}$ for any $i \neq j$.

21.4 General CPWL as a DNN

Assume that $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is a continuous function that are piecewise linear on m subdomains

$$\Omega_i, \quad i = 1 : m.$$

Namely, on each Ω_i , f is a linear function:

$$f(x) = f_i(x) = a_i \cdot x + b_i, \quad x \in \Omega_i,$$

with some $a_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$.

For the relationship between ReLU DNNs and general CPWL functions, we have the next theorem with some estimation.[?]

Theorem 133. A continuous function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ that are piecewise linear on m subdomains $\{\Omega_j\}_{j=1}^m$ can be represented by a ReLU DNN. Furthermore,

1. the number of hidden layers is bounded by

$$(21.24) \quad N_{\text{layer}} \leq \lceil \log_2(d+1) \rceil.$$

2. the number of neurons

$$(21.25) \quad N_{\text{neuron}} = \begin{cases} O\left(d2^{mM+(d+1)(m-d-1)}\right) & \text{if } m \geq d+1, \\ O\left(d2^{mM}\right) & \text{if } m < d+1. \end{cases}$$

here M , satisfying $m \leq M \leq m!$, is the number of subdomains in which $f_i - f_j$ does not change sign.

Combine Theorem 133 with Theorem 131, we have the following corollary regarding the minimal number of layers needed to recover all piecewise linear functions.

Corollary 12.

$$(21.26) \quad 2 \leq J_d \leq \lceil \log_2(d+1) \rceil.$$

This also indicates that $\lceil \log_2(d+1) \rceil$ is “optimal” for $d = 2, 3$.

Appendix: Implementation of Finite Element Methods

21.5 On the Implementation of Linear Finite Elements

In this note, we discuss some details for the implementation of piecewise linear elements in two dimensions. Although, the technique to be described can be applied to more general problems, we will mainly focus on the Poisson equation in a polygonal domain Ω with Dirichlet boundary conditions and homogeneous Neumann conditions. The model differential equation we are dealing with is:

$$\begin{cases} -\Delta u = f(x, y), & (x, y) \in \Omega, \\ u = g(x, y), & (x, y) \in \Gamma_D, \\ \frac{\partial u}{\partial n} = 0, & (x, y) \in \Gamma_N. \end{cases}$$

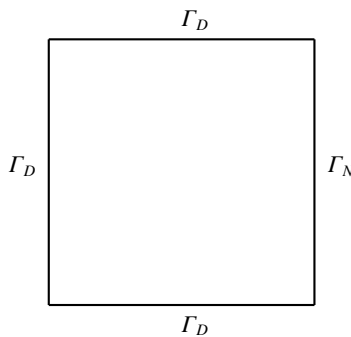


Fig. 21.4. Square domain

We triangulate Ω and we with n_e triangles and n number of nodes (vertices of the triangles) and as usual we denote the triangulation with \mathcal{T}_h .

21.5.1 Variational formulation

Define space

$$H_{0,\Gamma_D}^1 = \{v \in H^1(\Omega) | v|_{\Gamma_D} = 0\}$$

and

$$H_{g,\Gamma_D}^1 = \{v \in H^1(\Omega) | v|_{\Gamma_D} = g\}.$$

The variational formulation for this model problem is: Find $u \in H_{g,\Gamma_D}^1$, s.t.

$$(\nabla u, \nabla v) = (f, v), \quad \forall v \in H_{0,\Gamma_D}^1.$$

21.5.2 Finite element approximation

Define the triangulation $T_h = \{\tau_i\}_i^{n_e}$ and the finite element subspaces $V_{0,h} \subset H_{0,\Gamma_D}$ and $V_{g,h} \subset H_{g,\Gamma_D}$. Now the finite element problem is: Find $u_h \in V_{g,h}$, s.t.

$$(\nabla u_h, \nabla v_h) = (f, v_h), \quad \forall v_h \in V_{0,h}.$$

Now we show some details about assembling FEM matrices. Define $u_h = \sum_{i=1}^3 u_i \phi_i$, where ϕ_i is P1 nodal basis such that

$$\phi_i(a_j) = \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Now in the bilinear form $(\nabla u_h, \nabla v_h) = \int_{\Omega} \nabla u_h \cdot \nabla v_h dx$, we choose $v_h = \phi_j$, ($j = 1, 2, 3$). On the triangular of linear finite element, the barycentric coordinates are defined by

$$\lambda_i = \frac{S_i}{\text{Area}(\tau)}, \quad i = 1, 2, 3.$$

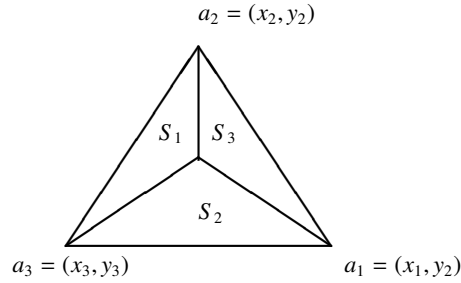


Fig. 21.5. triangular element

The area of the triangular can be calculated by

$$\text{Area}(\tau) = \frac{1}{2} \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix} = \frac{1}{2} \begin{vmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{vmatrix},$$

$$S_1 = \frac{1}{2} \begin{vmatrix} x - x_3 & x_2 - x_3 \\ y - y_3 & y_2 - y_3 \end{vmatrix}, \quad S_2 = \frac{1}{2} \begin{vmatrix} x - x_1 & x_3 - x_1 \\ y - y_1 & y_3 - y_1 \end{vmatrix}, \quad S_3 = \frac{1}{2} \begin{vmatrix} x - x_2 & x_1 - x_2 \\ y - y_2 & y_1 - y_2 \end{vmatrix}.$$

21.5.3 Basic arrays for defining the triangular grid

We define the basic arrays handling the triangulation. In what follows the dimension of the array will be in parenthesis after the array name. For example, an array named nt of dimension $3 \times n_E$ will be denoted by $NT(3, n_E)$.

- n_E = number of elements
- n = number of nodes
- Integer array $IB(n, 1)$ identifying the boundary nodes as follows:

$$IB(k) = \begin{cases} 0 & \text{if node } k \text{ is internal} \\ 1 & \text{if node } k \text{ is on } \Gamma_D \\ -1 & \text{if node } k \text{ is on } \Gamma_N \end{cases}$$

We note that the nodes, where the boundary condition changes its type are assumed to be grid nodes. For example, if the condition changes from Dirichlet to Neumann at some point $(x, y) \in \partial\Omega$ then (x, y) is vertex of a triangle of our triangulation. It is important to note that in the actual computations such nodes are always treated as nodes on Dirichlet boundary (in some sense this matches with saying that Γ_D is a closed set in $\partial\Omega$ and Γ_N is open).

- Real array $xy(n, 2)$ for the (x, y) coordinates of the nodes, namely $(x_i, y_i) \ i = 1 : n$.
- Integer array $NT(3, n_E)$ for the global numbering of three vertices of each element, namely a triangle numbered $\{ne\}$ has as a vertices

$$(NT(1, ne), NT(2, ne), NT(3, ne)).$$

For example, $NT(3, 6) = 8$ means that the 6-th triangle has as a 3-rd vertex node with coordinates x_8, y_8 .

21.5.4 Local stiffness matrix

The implementation of finite element method is very easy to carry out because the computations can be done locally (element by element). To obtain the resulting system of discrete equations we only need to assemble all these local computations together. Roughly speaking, this is the main idea and we will follow it throughout the implementation. First we have to compute the local stiffness matrices A_τ . To do this we

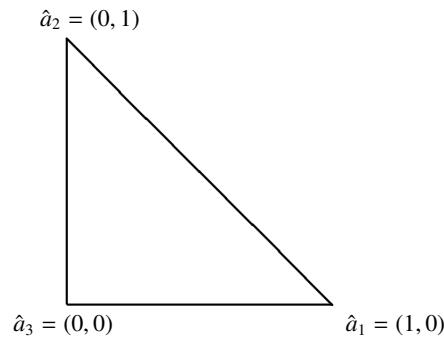


Fig. 21.6. reference element

shall use the so-called *reference element*. The reference element is the triangle $\hat{\tau}$ with vertices $\hat{a}_1 = (1, 0)$, $\hat{a}_2 = (0, 1)$, $\hat{a}_3 = (0, 0)$. The advantage of this approach is that the integrals we need to evaluate, can be computed on the reference triangle (only once!). Then we use these values many times in obtaining different local stiffness matrices for the elements on the actual grid. In other words, if we want to compute an integral on some triangle $\tau \in \mathcal{T}_h$, we only need to evaluate the *affine mapping* which maps the reference triangle $\hat{\tau}$ on τ . This procedure is described in more detail below.

- Affine mapping:
Assume $\tau \in \mathcal{T}_h$ is the ne -th element. Let To simplify the notation, we use (x_i, y_i) , ($i = 1, 2, 3$) as the vertices of current element.
We define an affine mapping $F(\hat{x}) = B_\tau \hat{x} + c : \hat{\tau} \mapsto \tau$:

$$B_\tau = \begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix}$$

and

$$c = \begin{pmatrix} x_3 \\ y_3 \end{pmatrix}.$$

- Real array $A_\tau(3, 3)$ for local stiffness matrix. If we take ne -th element τ then the local stiffness matrix is defined as follows.

$$A_\tau(i, j) = a(\lambda_i, \lambda_j),$$

Here

$$\hat{\lambda}_1 = \hat{x}, \hat{\lambda}_2 = \hat{y}, \hat{\lambda}_3 = 1 - \hat{x} - \hat{y}.$$

On each element, we have by change of variable

$$\int_\tau \nabla_x u_h : \nabla_x v_h dx = \int_{\hat{\tau}} \nabla_x \hat{u}_h : \nabla_x \hat{v}_h \det(B_\tau) d\hat{x}.$$

We use ∇_x to stress that it is the gradient in current coordinate. We know that for the affine mapping

$$\nabla_{\hat{x}} x = \nabla_{\hat{x}} F = B_\tau.$$

So by Chain rule

$$\int_\tau \nabla_x \hat{u}_h : \nabla_x \hat{v}_h \det(B_\tau) d\hat{x} = \int_{\hat{\tau}} B_\tau^{-1} \nabla_{\hat{x}} \hat{u}_h : B_\tau^{-1} \nabla_{\hat{x}} \hat{v}_h \det(B_\tau) d\hat{x}.$$

Here $\nabla_{\hat{x}}$ is the gradient w.r.t reference coordinate. For B_τ^{-1} , we have

$$B_\tau^{-1} = \frac{1}{\det(B_\tau)} \begin{pmatrix} y_2 - y_3 & x_3 - x_2 \\ y_3 - y_1 & x_1 - x_3 \end{pmatrix}.$$

Therefore, for $i, j = 1, 2, 3$, we have

$$\begin{aligned} A_\tau(i, j) &= \int_\tau \nabla_x \lambda_i : \nabla_x \lambda_j dx \\ &= \int_\tau \nabla_x \lambda_i : \nabla_x \lambda_j dx \\ (21.27) \quad &= \det(B_\tau) \int_{\hat{\tau}} \nabla_x \hat{\lambda}_i : \nabla_x \hat{\lambda}_j d\hat{x} \\ &= \det(B_\tau) \int_{\hat{\tau}} B_\tau^{-1} \nabla_{\hat{x}} \hat{\lambda}_i : B_\tau^{-1} \nabla_{\hat{x}} \hat{\lambda}_j d\hat{x}. \end{aligned}$$

On reference element, for P1 element it is easy to compute that

$$\nabla_{\hat{x}} \hat{\lambda}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \nabla_{\hat{x}} \hat{\lambda}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \nabla_{\hat{x}} \hat{\lambda}_3 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}.$$

For the local right hand side, similarly we have

$$f_\tau(j) = \int_\tau f \lambda_j dx = |\det(B_\tau)| \int_{\hat{\tau}} \hat{f} \hat{\lambda}_j d\hat{x}.$$

At this point, numerical quadrature has to be used.

In general, given M quadrature points $\{q_i\}$, we can approximate the integral by sum

$$\int_\tau f(x) dx = \sum_{i=1}^M f(q_i) \omega_i.$$

Since we are using P1 element, we can take the three mid points of edges to be quadrature points.

$$\int_{\hat{\tau}} \hat{f} \hat{\lambda}_j d\hat{x} \approx \frac{1}{6} \sum_{i=1}^3 \hat{f}(q_i) \hat{\lambda}_j(q_i).$$

with

$$q_1 = \left(0, \frac{1}{2}\right), \quad q_2 = \left(\frac{1}{2}, 0\right), \quad q_3 = \left(\frac{1}{2}, \frac{1}{2}\right).$$

And the weights are $\omega_1 = \omega_2 = \omega_3 = 1/6$.

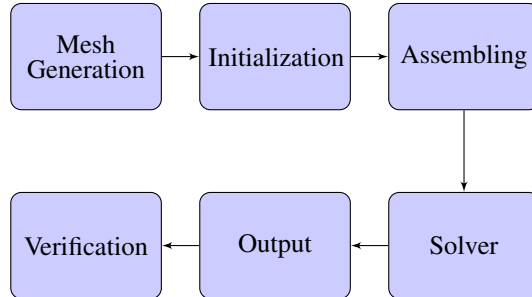
21.5.5 Assembling the global stiffness matrix and the right hand side

The main goal in doing all these computations is to form the resulting system of linear equation of the form:

$$Au = f$$

Here $A = A(n, n)$ is the global stiffness matrix and $f(n)$ is the right hand side and after solving this system, the vector $u(n)$ will contain the nodal values of the discrete solution.

The global stiffness matrix is assembled from the local ones as follows: For every particular element we first compute the local stiffness matrix $A_\tau(3, 3)$ and local right hand side $f_\tau(3)$. Then we add them in $A(n, n)$, $f(n)$ respectively. The place, where to add them is uniquely determined by the local-global correspondence we have stored in $NT(3, n_E)$. Thus, for the local matrices and right hand sides only two small arrays $A_\tau(3, 3)$ and $f_\tau(3)$ are used (but in the values in them are changed). More precisely the procedure is as follows:



- **Mesh Generation.**

From the mesh, we can get the following information.

- n : # of nodes, n_E :# of elements.
- $xy(2, n)$: 2-by- n array of coordinates of all the nodes.
- $NT(3, n_E)$: the global number of three vertices of each element.
- $IB(n)$: types of nodes as follows:

$$IB(k) = \begin{cases} 0 & \text{if node } k \text{ is internal} \\ 1 & \text{if node } k \text{ is on } \Gamma_D \\ -1 & \text{if node } k \text{ is on } \Gamma_N \end{cases} \quad 1 \leq k \leq n.$$

- **Initialization**

- $A \leftarrow 0, f \leftarrow 0$
- For $k = 1 : n$, if the node k is on the Dirichlet boundary (namely $IB(k) > 0$), then

$$A(k, k) = 1, f(k) = u_D(x_k, y_k), \text{ the Dirichlet value at node } k.$$

- **Assembling**

```

for  $ne = 1, \dots, n_e$  do
  Form the local stiffness matrix  $A_\tau(3, 3)$  for element  $ne$  (denoted by  $\tau_{ne}$ )
  for  $i = 1, 2, 3$  do
    Let  $ie = NT(i, ne)$ 
    if  $IB(ie) \leq 0$  then
       $f(ie) \leftarrow f(ie) + f_\tau(i)$ 
    end if
    for  $j = 1, 2, 3$  do
      Let  $je = NT(j, ne)$ 
      if  $IB(je) \leq 0$  then
        if  $IB(ie) \leq 0$  then
           $A(ie, je) \leftarrow A(ie, je) + A_\tau(i, j)$ 
        else
           $f(je) \leftarrow f(je) - A_\tau(j, i) * f(ie)$ 
        end if
      end if
    end for
  end for
end for

```

In the procedure described above a special care is taken for the Dirichlet values. It has to be noted that the right hand side for a Dirichlet node always must contain the corresponding Dirichlet value (the value of u_D). The right hand sides of the immediate neighbors to such boundary node have to be updated in proper way. We will explain this in some more detail. Let us pick a node k lying on the Dirichlet boundary and let j be in the interior of Ω and $A(j, k) \neq 0$. Then the j -th equation in our resulting system looks like:

$$A(1, j)u(1) + \dots + A(k, j)u(k) + \dots = f(j).$$

The known value of $u(k) = u_D(x_k, y_k)$ has to be moved to the right hand side, i.e.

$$A(1, j)u(1) + \dots + 0 \cdot u(k) + \dots = f(j) - A(k, j)u_D(x_k, y_k).$$

As it is seen, we are able to do this procedure locally (in one and the same loop with the assembling), because $A(j, k) \neq 0$ only if k -th and j -th node are in one and the same element.

21.5.6 Nonhomogenous problem

For nonhomogenous problem, the Neumann boundary condition is nonzero.

$$\begin{cases} -\Delta u = f(x, y), & (x, y) \in \Omega, \\ u = g(x, y), & (x, y) \in \Gamma_D, \\ \frac{\partial u}{\partial n} = h(x, y), & (x, y) \in \Gamma_N. \end{cases}$$

It is easy to derive the corresponding weak form: Find $u \in H_{g, \Gamma_D}^1$, such that

$$(\nabla u, \nabla v) = (f, v) + \int_{\Gamma_N} h v ds, \quad \forall v \in H^1(0, \Gamma_D).$$

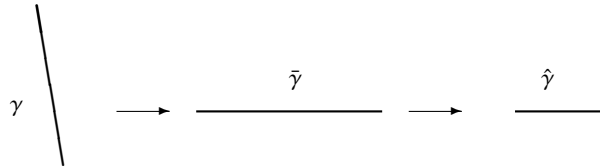


Fig. 21.7. line element

- $\hat{\gamma}$ is horizontal and has the same length with γ .
- $\hat{\gamma}$ is the reference line element $[0, 1]$ of $\hat{\gamma}$.
- base functions on reference element: $\hat{\phi}_1 = 1 - \hat{x}$, $\hat{\phi}_2 = \hat{x}$.
- $\int_{\gamma} h v_h ds = \int_{\hat{\gamma}} \det(F) \hat{h} \hat{v}_h$. The boundary integral can be approximated by

$$\int_{\hat{\gamma}} \hat{h} \hat{\phi}_1 \approx \frac{L}{2} \hat{h}(0), \quad \int_{\hat{\gamma}} \hat{h} \hat{\phi}_2 \approx \frac{L}{2} \hat{h}(1),$$

where $L = \bar{x}_2 - \bar{x}_1$. Here we use quadrature points $q_0 = 0$, $q_1 = 1$ and the weights $\omega_1 = 1/2$, $\omega_2 = 1/2$.

Then we can compute

$$h_{\tau}(1) = \frac{L}{2} \hat{h}(0), \quad h_{\tau}(2) = \frac{L}{2} \hat{h}(1).$$

In this case $h_{\tau}(3) = 0$.

So by the procedures above, we can obtain the local boundary integral $h_{\tau}(3)$. Now we show the assembling algorithm for nonhomogenous case.

```

for  $ne = 1, \dots, n_E$  do
    Form the local stiffness matrix  $A_{\tau}(3, 3)$  for element  $ne$  (denoted by  $\tau_{ne}$ )
    for  $i = 1, 2, 3$  do
        Let  $ie = NT(i, ne)$ 
        if  $IB(ie) \leq 0$  then
             $f(ie) \leftarrow f(ie) + f_{\tau}(i)$ 
        end if
        if  $IB(ie) < 0$  then
             $f(ie) \leftarrow f(ie) + h_{\tau}(i)$ 
        end if
        for  $j = 1, 2, 3$  do
            Let  $je = NT(j, ne)$ 
            if  $IB(je) \leq 0$  then
                if  $IB(ie) \leq 0$  then
                     $A(ie, je) \leftarrow A(ie, je) + A_{\tau}(i, j)$ ,
                else
                     $f(je) \leftarrow f(je) - A_{\tau}(j, i) * f(ie)$ .
                end if
            end if
        end for
    end for
end for
    
```

To conclude, we would like to mention two immediate generalizations of the implementation we have just described. Although the differential equation considered here was fairly simple, the implementation can

be extended very easy to handle more complicated equations and more complicated finite element methods. For example quadrature formulae for the evaluation of the integrals can be easily included if the simple model problem we consider is replaced by some more general elliptic PDE with non-constant coefficients. Another possible generalization is the use of higher order finite elements. In both cases only the computation of the integrals in the reference element has to be changed.

Homework

- Solve

$$\begin{cases} -\Delta u = -4, & (x, y) \in \Omega, \\ u = x^2 + y^2, & (x, y) \in \partial\Omega, \end{cases}$$

on squared domain $[0, 1] \times [0, 1]$ with both structured mesh and unstructured mesh.

- Solve

$$\begin{cases} -\Delta u = f(x, y), & (x, y) \in \Omega, \\ u = x^2 + y^2, & (x, y) \in \Gamma_D, \\ \frac{\partial u}{\partial n} = 2x, & (x, y) \in \Gamma_N. \end{cases}$$

on the following square domain $[0, 1] \times [0, 1]$.

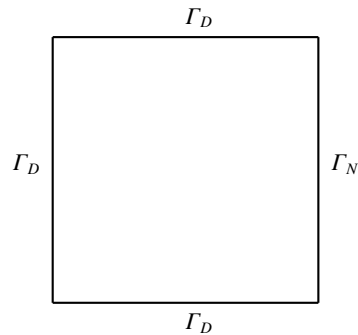


Fig. 21.8. Square domain