有限要素法プログラミング演習

渡辺浩志

2000 年 7 月 17 日

はじめに

このテキストは, 解析実務に供することができるような, 非線形有限要素解析コードをゼロから作成できるような自習書を目指しています. より具体的には, 久田俊明著「非線形有限要素法のためのテンソル解析の基礎」, 久田俊明, 野口裕久著「非線形有限要素法の基礎と応用」の内容を基に, 実際の有限要素解析コードを作成する手順を示しています.

歴史的に有限要素法の最初の論文は, アメリカのボーイング社のエンジニアによって発表されたものとされています. それ以来, 特に冷戦下でアメリカと旧ソ連の熾烈な航空機開発競争にともない, 大量の研究費と優秀な人材が投入され大きく発展してきました. そのため「有限要素法=構造解析ツール」というイメージが強いと思いますが, アプリケーションとして構造解析が多いだけで, 有限要素法自体は応用数学で構造解析以外にも応用できます. また数学ですから, じっくり考えれば理解できることが積み重ねられているだけで, 「怪しい工学的モデル化」とは無縁です.

有限要素法は, はじめて学ぶ人にとっては敷居の高いものだと思います. 世の中に有限要素法の教科書はたくさん出版されていますが, そのどれもが, やたらと数式が並んでいて, それだけでうんざりしてくるにも関わらず, プログラムリストが例示されていても, 構造化プログラミングを意識する以前に, インデントすらされていない, つまりプログラミングの講義で悪いプログラムの代表として教えられるような古色蒼然とした fortran のリストがのっているだけで, はっきりいって近寄りがたい雰囲気が漂っているように感じられます. 少なくとも, 著者のように学生時代, 授業にはたまにしか出てこない, レポートや製図の課題は期限内に出したことが無いような不良学生にはついて行けず, 途中で挫折したものばかりでした.

そんな, わかりにくい教科書で勉強しながら, 著者が感じていたことは,

(i) 有限要素法は数学的にはとても厳密で, 気合いを入れて取り組めば理解できる. しかし, この教科書の 10 倍ぐらいの分量で説明がなされていたら, 理解するのに必要な時間は, 10 分の 1 程度になるだろう.

(ii) 古色蒼然とした, 即ち構造化されていないのみならず, インデントすらしていない fortran のコーディング例ではなく, fortran にしても構造化を意識したもの, あるいは C などでコーディングしたものであれば, 遥かに分かりやすいコードになるだろう.

ということでした.

(i) に関しては,「本」という形態を取る以上ある程度仕方の無いことだと思います. いまでも十分分厚いものが, さらに 10 倍になったらそれは東京都 23 区の電話帳になってしまいます. しかし敢えてやってみる価値はあると思います.

(ii) については, これこそわれわれ若い世代の仕事だと思います. どのような言語でもコーディングできるように, まず規格化されたコーディング例を提示し, fortran でコーディングするにしても Pascal ライクなコーディング, 即ちプログラムの制御には, for 文, if then else 文しか使わない. また C でのコーディングも例示する. このとき, fortran と C を混在できるようなコーディングを採用します.

近年有限要素法は飛躍的な発展をとげ, 市販の汎用有限要素解析コードでも高度な非線形解析を行うことができるようになってきました. しかしながら, その中身までわかって使っている人はそれほど多いわけではなさそうで, 実際に問題意識を持っている人も多いようです. このテキストでは, 初心者のみならずそのような人の要望にも応えられるように最新の非線形解析手法についても解説します.

このテキストは, 著者が実際にコーディングする際に作成するノートと実際のコードを, ほぼそのまま例示しています. その意味では癖のあるものになっていると思いますが, どのコードもすべて同じ人間が書いたものであり, その意味での統一性はあると思います. また研究室の学生の質問事項に応えて細かなことまで書いていますので, 他の参考文献はほとんど必要ないと思います. したがって, かなり長い道程になっていますが, 長いだけで, そんなに山も谷も無いと思いますので, 皆さんがんばってください.

目 次

第1章　Gauss の消去法

プログラミングの観点から見た有限要素法は

(1) マトリックスの操作により連立 1 次方程式を作る.

(2) 得られた連立 1 次方程式を解く.

の繰り返しである. ここでは数値計算のプログラミングに慣れる意味も含めて連立 1 次方程式を解くプログラムを作成する. 解法には代表的な Gauss の消去法を用いる.

1.1　Gauss の消去法の例

以下のような連立 1 次方程式を例にとる. 係数マトリックスを $[A]$, 未知ベクトルを $\{b\}$, 右辺を $\{c\}$ と表す.

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{1.1}$$

この連立 1 次方程式を「ある決まった手順の繰り返し」で解く方法を考える. 便宜的に $[A^{(1)}] = [A]$, $\{c^{(1)}\} = \{c\}$ とおく.

1.1.1　前進消去

step 1. $[A^{(1)}]$ の 1 列目の対角項より下が 0 になるように 2 ∼ 4 行目の方程式を右辺を含めて下式のように操作する.

$$2\text{ 行目}-\{\underbrace{(-4)}_{A^{(1)}_{21}}/\underbrace{5}_{A^{(1)}_{11}}\}\times 1\text{ 行目}$$
$$3\text{ 行目}-\{\underbrace{1}_{A^{(1)}_{31}}/\underbrace{5}_{A^{(1)}_{11}}\}\times 1\text{ 行目}$$
$$4\text{ 行目}-\{\underbrace{0}_{A^{(1)}_{41}}/\underbrace{5}_{A^{(1)}_{11}}\}\times 1\text{ 行目}$$

この操作により $[A^{(1)}]\{b\}=\{c^{(1)}\}$ は以下のように変形され, これを $[A^{(2)}]\{b\}=\{c^{(2)}\}$ と表す.

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & \frac{14}{5} & -\frac{16}{5} & 1 \\ 0 & -\frac{16}{5} & \frac{29}{5} & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix}\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}=\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{1.2}$$

step 2. $[A^{(2)}]$ の 2 列目の対角項より下が 0 になるように次のように 3, 4 行目の方程式を右辺を含めて下式のように操作する.

$$3\text{ 行目}-\left\{\underbrace{\left(-\frac{16}{5}\right)}_{A^{(2)}_{23}}\Big/\underbrace{\frac{14}{5}}_{A^{(2)}_{22}}\right\}\times 2\text{ 行目}$$
$$4\text{ 行目}-\left\{\underbrace{1}_{A^{(2)}_{24}}\Big/\underbrace{\frac{14}{5}}_{A^{(2)}_{22}}\right\}\times 2\text{ 行目}$$

この操作により $[A^{(2)}]\{b\}=\{c^{(2)}\}$ は以下のように変形され, これを $[A^{(3)}]\{b\}=\{c^{(3)}\}$ と表す.

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & \frac{14}{5} & -\frac{16}{5} & 1 \\ 0 & 0 & \frac{15}{7} & -\frac{20}{7} \\ 0 & 0 & -\frac{20}{7} & \frac{65}{14} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ \frac{8}{7} \\ -\frac{5}{14} \end{bmatrix} \tag{1.3}$$

step 3. $[A^{(3)}]$ の 3 列目の対角項より下が 0 になるように 4 行目の方程式を右辺を含めて下式のように操作する.

$$4\text{ 行目} - \left\{ \underbrace{\left(-\frac{20}{7}\right)}_{A_{34}^{(3)}} \Big/ \underbrace{\frac{15}{7}}_{A_{44}^{(3)}} \right\} \times \text{ 3 行目}$$

この操作により $[A^{(3)}]\{b\} = \{c^{(3)}\}$ は以下のように変形され, これを $[A^{(4)}]\{b\} = \{c^{(4)}\}$ と表す.

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & \frac{14}{5} & -\frac{16}{5} & 1 \\ 0 & 0 & \frac{15}{7} & -\frac{20}{7} \\ 0 & 0 & 0 & \frac{5}{6} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ \frac{8}{7} \\ \frac{7}{6} \end{bmatrix} \tag{1.4}$$

ここまでの操作により式 (1.1) は, 上三角行列[1]を係数マトリックスに持つ連立 1 次方程式に変換された. 以上の操作を前進消去 (forward reduction) と呼ぶ.

1.1.2 後退代入

未知数 $\{b\}$ は b_4, b_3, b_2, b_1 の順に以下のように求める事が出来る. この操作を後退代入 (backward substitution) と呼ぶ.

[1]対角項より下の成分がすべて 0 であるような行列を上三角行列という. 同様に対角項より上の成分が全て 0 であるような行列を下三角行列と言う.

step 1.

$$b_4 = \underbrace{\frac{7}{6}}_{c_4^{(4)}} \Big/ \underbrace{\frac{5}{6}}_{A_{44}^{(4)}} = \frac{7}{5} \tag{1.5}$$

step 2.

$$b_3 = \left\{ \underbrace{\frac{8}{7}}_{c_3^{(4)}} - \underbrace{\left(-\frac{20}{7}\right)}_{A_{34}^{(4)}} \cdot \underbrace{\frac{7}{5}}_{b_4} \right\} \Big/ \underbrace{\frac{15}{7}}_{A_{33}^{(4)}} = \frac{12}{5} \tag{1.6}$$

step 3.

$$b_2 = \left\{ \underbrace{1}_{c_2^{(4)}} - \underbrace{\left(-\frac{16}{5}\right)}_{A_{23}^{(4)}} \cdot \underbrace{\frac{12}{5}}_{b_3} - \underbrace{1}_{A_{24}^{(4)}} \cdot \underbrace{\frac{7}{5}}_{b_4} \right\} \Big/ \underbrace{\frac{14}{5}}_{A_{22}^{(4)}} = \frac{13}{5} \tag{1.7}$$

step 4.

$$b_1 = \left\{ \underbrace{0}_{c_1^{(4)}} - \underbrace{(-4)}_{A_{12}^{(4)}} \cdot \underbrace{\frac{13}{5}}_{b_2} - \underbrace{1}_{A_{13}^{(4)}} \cdot \underbrace{\frac{12}{5}}_{b_3} - \underbrace{0}_{A_{14}^{(4)}} \cdot \underbrace{\frac{7}{5}}_{b_4} \right\} \Big/ \underbrace{5}_{A_{11}^{(4)}} = \frac{8}{5} \tag{1.8}$$

1.1.3　後退消去

あるいは, 以下のような方法でも出来る. この操作を後退消去 (backward reduction) と呼ぶ. これは, 左辺の上三角行列を対角化する時と同じ操作を右辺に加えていることになる. ここでは $c_i^{(4,0)} = c_i^{(4)}$ とおく.

(福成君はわからなかったらしい. 要, 詳しい説明, マトリックスが次元が小さくなっていくことがわからない.)

step 1.

$$b_4 = \underbrace{\frac{7}{6}}_{c_4^{(4,0)}} \Big/ \underbrace{\frac{5}{6}}_{A_{44}^{(4)}} = \frac{7}{5} \tag{1.9}$$

$$c_1^{(4,1)} = \underbrace{0}_{c_1^{(4,0)}} - \underbrace{0}_{A_{14}^{(4)}} \cdot \underbrace{\frac{7}{5}}_{b_4} = 0 \tag{1.10}$$

$$c_2^{(4,1)} = \underbrace{1}_{c_2^{(4,0)}} - \underbrace{1}_{A_{24}^{(4)}} \cdot \underbrace{\frac{7}{5}}_{b_4} = -\frac{2}{5} \tag{1.11}$$

$$c_3^{(4,1)} = \underbrace{\frac{8}{7}}_{c_3^{(4,0)}} - \underbrace{\Big(-\frac{20}{7}\Big)}_{A_{34}^{(4)}} \cdot \underbrace{\frac{7}{5}}_{b_4} = \frac{36}{7} \tag{1.12}$$

step 2.

$$b_3 = \underbrace{\frac{36}{7}}_{c_3^{(4,1)}} \Big/ \underbrace{\frac{15}{7}}_{A_{33}^{(4)}} = \frac{12}{5} \tag{1.13}$$

$$c_1^{(4,2)} = \underbrace{0}_{c_1^{(4,1)}} - \underbrace{1}_{A_{13}^{(4)}} \cdot \underbrace{\frac{12}{5}}_{b_3} = -\frac{12}{5} \tag{1.14}$$

$$c_2^{(4,2)} = \underbrace{-\frac{2}{5}}_{c_2^{(4,1)}} - \underbrace{\Big(-\frac{16}{5}\Big)}_{A_{23}^{(4)}} \cdot \underbrace{\frac{12}{5}}_{b_3} = \frac{182}{25} \tag{1.15}$$

step 3.

$$b_2 = \underbrace{\frac{182}{25}}_{c_2^{(4,2)}} \Big/ \underbrace{\frac{14}{5}}_{A_{22}^{(4)}} = \frac{13}{5} \tag{1.16}$$

$$c_1^{(4,3)} = \underbrace{-\frac{12}{5}}_{c_1^{(4,2)}} - \underbrace{(-4)}_{A_{12}^{(4)}} \cdot \underbrace{\frac{13}{5}}_{b_2} = 8 \tag{1.17}$$

step 4.

$$b_1 = \underbrace{8}_{c_1^{(4,3)}} \Big/ \underbrace{5}_{A_{11}^{(4)}} = \frac{8}{5} \tag{1.18}$$

後退代入と後退消去はかなり異なった手続きで, 手計算で解く際には後退消去は煩雑な印象を受けるが, 未知数を全て求めるのに必要な演算数はそれぞれ加減 6 回, 乗算 6 回, 除算 4 回と全く同じである. ここで各 step 内に現れる係数マトリックスの成分に着目すると, 後退代入では行が固定されており, 一方後退消去では列が固定されていることがわかる. 後退代入と後退消去の違いはコーディングを行う際には loop index のとり方として反映され, 係数マトリックスの記憶方法により使い分けられる. 実際の有限要素コードでは skyline 法と呼ばれる Gauss の消去法を基に効率化を図った手法が事実上の標準として用いられることが多いが, skyline 法は後退消去を採用している. 理由はやはり係数マトリックスの記憶方法である.

1.2　三角分解

Gauss の消去法で行われた一連の操作の最大の特徴は, 係数マトリックスを上三角行列に変換する操作には右辺の値を用いていないこと即ち, 右辺の値とは無関係に係数マトリックスの値だけで上三角行列に変換できることにある. そこでこの操作を以下のように基本行列の積で表してみる.

1.2.1　基本行列

まず, 以下のように対角成分はすべて 1, i, j 成分 $(i \neq j)$ は C, その他はすべて 0 という基本行列 $[P(i, j : C)]$ を定義する.

$$[P(i,j:C)] = \begin{array}{c} \\ \\ \\ \\ i\text{行} \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & & j\text{列} & & & \end{array} \\ \begin{bmatrix} 1 & & \vdots & & & & 0 \\ & 1 & \vdots & & & & \\ & & 1 & & & & \\ & & \vdots & 1 & & & \\ \cdots & \cdots & C & & \ddots & & \\ & & & & & 1 & \\ 0 & & & & & & 1 \end{bmatrix} \end{array} \tag{1.19}$$

$[P(i,j:C)]$ を行列 $[A]$ の左からかけると, $[A]$ の i 行に $[A]$ の j 行の C 倍が加えられる. 1.1 節で述べた上三角行列への変換プロセスは, すべて, この操作によって行われているので, 係数行列を $[A]$, $[A]$ から得られる上三角行列を $[S]$, $[P_n]$ を適当な基本行列として以下のような形式の関係が成立することになる.

$$[P_n]...[P_2][P_1][A] = [S] \tag{1.20}$$

例として, 式 (1.1) を式 (1.20) の形式で表すと以下のようになる.

$$\underbrace{\begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ 0 & 0 & -\frac{4}{3} & 1 \end{bmatrix}}_{step3} \underbrace{\begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ 0 & \frac{5}{14} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ 0 & -\frac{8}{7} & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{step2} \underbrace{\begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ \frac{1}{5} & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & 0 \\ -\frac{4}{5} & 1 & & \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{step1}$$

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix} = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & \frac{14}{5} & -\frac{16}{5} & 1 \\ 0 & 0 & \frac{15}{7} & -\frac{20}{7} \\ 0 & 0 & 0 & \frac{5}{6} \end{bmatrix} \tag{1.21}$$

ここで各ステップで用いる基本行列の積を計算してみると以下のようになる.

step 1.

$$\begin{bmatrix} 1 & & & 0 \\ -\frac{4}{5} & 1 & & \\ \frac{1}{5} & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.22}$$

step 2.

$$\begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ 0 & -\frac{8}{7} & 1 & \\ 0 & \frac{5}{14} & 0 & 1 \end{bmatrix} \tag{1.23}$$

step 3.

$$\begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ 0 & 0 & -\frac{4}{3} & 1 \end{bmatrix} \tag{1.24}$$

1.2.2　基本行列による三角分解

一般に基本行列 $[P(i,j:C)]$ の j を固定し, $i=j+1\sim n$ とした行列の積は

$$[P(i+1,i:C_{i+1})][P(i+2,i:C_{i+2})]\dots[P(u,i:C_n)]=$$

$$\begin{array}{c} \\ i\text{ 行} \end{array} \begin{array}{c} \begin{array}{cccccccc} & & & i\text{ 列} & & & & \end{array} \\ \begin{bmatrix} 1 & & \vdots & & & & 0 & \\ & \ddots & \vdots & & & & & \\ \dots & \dots & 1 & & & & & \\ & & C_{i+1} & 1 & & & & \\ & & C_{i+2} & & \ddots & & & \\ 0 & & \vdots & & & \ddots & & \\ & & C_n & 0 & & & 1 & \end{bmatrix} \end{array} \tag{1.25}$$

となり, 対角項は全て 1, i 列目の $i+1$ 行以下に $C_{i+1}, C_{i+2}, \ldots C_n$ が並び残りの要素は 0 という形式になる. また左辺の横の順序を入れ換えても結果はかわらない.

$[P(i,j:C)]$ の逆行列は $[P(i,j:-C)]$ であることから, この逆行列は以下のように $C_{i+1}, C_{i+2}, \ldots C_n$ の符合を換えたものになる.

$$\begin{bmatrix} 1 & & & & & 0 \\ & \ddots & & & & \\ & & 1 & & & \\ & & C_{i+1} & 1 & & \\ 0 & & \vdots & & \ddots & \\ & & C_n & 0 & & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & & & & & 0 \\ & \ddots & & & & \\ & & 1 & & & \\ & & -C_{i+1} & 1 & & \\ 0 & & \vdots & & \ddots & \\ & & -C_n & 0 & & 1 \end{bmatrix} \tag{1.26}$$

そこで, 後の定式化の便利のため, 下式に従い $[L_i]$ を定義する.

$$[L_i] = \begin{array}{c} \\ i\text{ 行} \end{array} \begin{array}{c} \overset{i\text{ 列}}{} \\ \begin{bmatrix} 1 & & \vdots & & 0 \\ & \ddots & \vdots & & \\ \ldots & \ldots & 1 & & \\ & & L_{i+1,i} & 1 & \\ 0 & & \vdots & & \ddots \\ & & L_{n,i} & 0 & & 1 \end{bmatrix} \end{array} \tag{1.27}$$

ただし,

$$L_{i+j,i} = \frac{A^{(i)}_{i+j,i}}{A^{(i)}_{i,i}} \tag{1.28}$$

$[L_i]$ の逆行列を $[L_i^{-1}]$ と表せば $[L_i^{-1}]$ は以下のようになる.

$$[L_i^{-1}] = \begin{array}{c} \\ \\ i \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} & & i & & & \end{array} \\ \begin{bmatrix} 1 & & \vdots & & 0 & \\ & \ddots & \vdots & & & \\ \dots & \dots & 1 & & & \\ & & -L_{i+1,i} & 1 & & \\ 0 & & \vdots & & \ddots & \\ & & -L_{n,i} & 0 & & 1 \end{bmatrix} \end{array} \tag{1.29}$$

係数マトリックス $[A]$ の次元を n とすると, この $[L_i^{-1}]$ を用い, 前節で説明した前進消去の操作は以下のように表すことができ, $[A^{(n)}]$ は上三角行列になっている.

$$\begin{aligned} [A^{(2)}] =& [L_1^{-1}][A^{(1)}] \\ & \vdots \\ [A^{(n)}] =& [L_{n-1}^{-1}][A^{(n-1)}] \end{aligned} \tag{1.30}$$

$[S] = [A^{(n)}]$ とすれば, 以下のように書くことができる.

$$[L_{n-1}^{-1}] \dots [L_2^{-1}][L_1^{-1}][A] = [S] \tag{1.31}$$

例として式 (1.1) について式 (1.29), (1.31) を適用することにより上三角行列 $[S]$ を計算してみると以下のようになる.

$$[A^{(1)}] = \begin{bmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix} \tag{1.32}$$

$$[L_1^{-1}] = \begin{bmatrix} 1 & & & 0 \\ -L_{2,1} & 1 & & \\ -L_{3,1} & 0 & 1 & \\ -L_{4,1} & 0 & 0 & 1 \end{bmatrix} \quad \text{但し} \begin{cases} L_{2,1} = \dfrac{A_{2,1}^{(1)}}{A_{1,1}^{(1)}} = -\dfrac{4}{5} \\ L_{3,1} = \dfrac{A_{3,1}^{(1)}}{A_{1,1}^{(1)}} = \dfrac{1}{5} \\ L_{4,1} = \dfrac{A_{4,1}^{(1)}}{A_{1,1}^{(1)}} = \dfrac{0}{5} = 0 \end{cases} \tag{1.33}$$

$$[L_1^{-1}][A^{(1)}] = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & \frac{14}{5} & -\frac{16}{5} & 1 \\ 0 & -\frac{16}{5} & \frac{29}{5} & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix} = [A^{(2)}] \tag{1.34}$$

$$[L_2^{-1}] = \begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ 0 & -L_{3,2} & 1 & \\ 0 & -L_{4,2} & 0 & 1 \end{bmatrix} \quad \text{但し} \begin{cases} L_{3,2} = \dfrac{A^{(2)}_{3,2}}{A^{(2)}_{2,2}} = -\dfrac{8}{7} \\ L_{4,2} = \dfrac{A^{(2)}_{4,2}}{A^{(2)}_{2,2}} = \dfrac{5}{14} \end{cases} \tag{1.35}$$

$$[L_2^{-1}][A^{(2)}] = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & \frac{14}{5} & -\frac{16}{5} & 1 \\ 0 & 0 & \frac{15}{7} & -\frac{20}{7} \\ 0 & 0 & -\frac{20}{7} & \frac{65}{14} \end{bmatrix} = [A^{(3)}] \tag{1.36}$$

$$[L_3^{-1}] = \begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ 0 & 0 & -L_{4,3} & 1 \end{bmatrix} \quad \text{但し } L_{4,3} = \frac{A^{(3)}_{4,3}}{A^{(3)}_{3,3}} = -\frac{4}{3} \tag{1.37}$$

$$[L_3^{-1}][A^{(3)}] = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & \frac{14}{5} & -\frac{16}{5} & 1 \\ 0 & 0 & \frac{15}{7} & -\frac{20}{7} \\ 0 & 0 & 0 & \frac{5}{6} \end{bmatrix} = [A^{(4)}] = [S] \tag{1.38}$$

式 (1.31) の右側から順に $[L_{n-1}], [L_{n-2}], \ldots, [L_2], [L_1]$ をかけることにより下式が得られる.

$$[A] = [L_1][L_2] \ldots [L_{n-1}][S] \tag{1.39}$$

ここで $[L]$ を以下のように定義する.

$$[L] = [L_1][L_2] \ldots [L_{n-1}] \tag{1.40}$$

$[L]$ を成分で書けば, 以下のようになる.

$$[L] = \begin{bmatrix} 1 & & & & & & \\ & & & & & \mathbf{0} & \\ L_{2,1} & 1 & & & & & \\ L_{3,1} & L_{3,2} & \ddots & & & & \\ & & & 1 & & & \\ \vdots & \vdots & & L_{i+1,i} & \ddots & & \\ & & & \vdots & & 1 & \\ L_{n,1} & L_{n,2} & \dots & L_{n,i} & \dots & L_{n,n-1} & 1 \end{bmatrix} \tag{1.41}$$

この $[L]$ を用いれば式 (1.39) は

$$[A] = [L][S] \tag{1.42}$$

と下三角行列 $[L]$ と上三角行列 $[S]$ の 積の形に分解できることが分かる. これを三角分解と呼ぶ.

$[S]$ の対角項は, 三角分解の途中や, 未知ベクトルを求めるときに, 割り算の分母として用いられるため, $[S]$ の対角項を並べた対角行列を $[D]$ として $[S] = [D][U]$ とさらに分解

する場合が多い. $[D], [U]$ を成分で書けば, 以下のようになる.

$$
\begin{aligned}
[S] &= \begin{bmatrix} S_{11} & S_{12} & \dots & S_{1n} \\ & S_{22} & & \vdots \\ & & \ddots & \vdots \\ \Large 0 & & & S_{nn} \end{bmatrix} \\
&= \begin{bmatrix} S_{11} & & & \Large 0 \\ & S_{22} & & \\ & & \ddots & \\ \Large 0 & & & S_{nn} \end{bmatrix} \begin{bmatrix} S_{11}/S_{11} & S_{12}/S_{11} & \dots & S_{1n}/S_{11} \\ & S_{22}/S_{22} & \dots & S_{2n}/S_{22} \\ & & \ddots & \vdots \\ \Large 0 & & & S_{nn}/S_{nn} \end{bmatrix} \\
&= \begin{bmatrix} S_{11} & & & \Large 0 \\ & S_{22} & & \\ & & \ddots & \\ \Large 0 & & & S_{nn} \end{bmatrix} \begin{bmatrix} 1 & S_{12}/S_{11} & \dots & S_{1n}/S_{11} \\ & 1 & & S_{2n}/S_{22} \\ & & \ddots & \vdots \\ \Large 0 & & & 1 \end{bmatrix}
\end{aligned} \tag{1.43}
$$

これより

$$
[D] = \begin{bmatrix} S_{11} & & & \Large 0 \\ & S_{22} & & \\ & & \ddots & \\ \Large 0 & & & S_{nn} \end{bmatrix} \tag{1.44}
$$

$$
[U] = \begin{bmatrix} 1 & S_{12}/S_{11} & \dots & S_{1n}/S_{11} \\ & 1 & & S_{2n}/S_{22} \\ & & \ddots & \vdots \\ \Large 0 & & & 1 \end{bmatrix} \tag{1.45}
$$

これからわかるように $[U]$ の対角項は $[L]$ 同様全て 1 である. この操作により $[A]$ は最終的に下式のように分解される.

$$
[A] = [L][D][U] \tag{1.46}
$$

このような形式にすることにより, コーディングをする際にも loop index をシンメトリックに構成することができ, また $[A]$ が対称行列ならば $[U] = [L^T]$ である. すなわち $[A]$ は

下式のように分解される. [2]

$$[A] = [L][D][L^T] = [U^T][D][U] \tag{1.47}$$

1.2.3　三角行列と連立一次方程式の求解

このように $[A] = [L][D][U]$ と分解することにより, 連立 1 次方程式 $[A]\{b\} = \{c\}$ は, 以下のように両辺に順次逆行列を作用させて行くようにして求解できる.

$$\begin{aligned} [L][D][U]\{b\} &= \{c\} \\ [D][U]\{b\} &= [L^{-1}]\{c\} \\ [U]\{b\} &= [D^{-1}][L^{-1}]\{c\} \\ \{b\} &= [U^{-1}][D^{-1}][L^{-1}]\{c\} \end{aligned} \tag{1.48}$$

まず $[L^{-1}]\{c\}$ を求める. これを $\{x\} = [L^{-1}]\{c\}$ とおけば, $\{x\}$ は以下に示すような下三角行列を係数マトリックスとする連立 1 次方程式の解である.

$$[L]\{x\} = \{c\} \tag{1.49}$$

このような連立 1 次方程式であれば, 前進代入 (forward substitution) すなわち先に説明した後退代入をちょうど逆にした操作により求解される. 具体的には以下のようになる.

$$x_1 = c_1 \tag{1.50}$$

$$x_2 = c_2 - L_{21}x_1 \tag{1.51}$$

$$x_3 = c_3 - L_{31}x_1 - L_{32}x_2 \tag{1.52}$$

$$\vdots$$

当然ながら後退消去と同様に前進消去を考えることも可能だが, skyline 法では用いられないのでここでは割愛する.

[2] なお慣用的に論文や雑誌などの文章中に出てくるときは $[L][D][L^T]$ と記述されている場合が多いが, そのような論文や書籍に掲載されているプログラム例では上三角行列を記憶し, $[U^T][D][U]$ と分解している場合が多い.

次に $[D^{-1}][L^{-1}]\{c\}$ を求める. 先に求めた $\{x\}$ を用いれば, $[D^{-1}][L^{-1}]\{c\} = [D^{-1}]\{x\}$ でありこれを $\{y\} = [D^{-1}]\{x\}$ とおけば $\{y\}$ は以下に示すような対角行列を係数マトリックスとする連立 1 次方程式の解である.

$$[D]\{y\} = \{x\} \tag{1.53}$$

このような連立 1 次方程式なら単なる割算で求解される. 具体的には以下のようになる.

$$y_i = x_i/D_{ii}\ (i = 1 \sim n) \tag{1.54}$$

最後に $[U^{-1}][D^{-1}][L^{-1}]\{c\}$ を求める. これは先に求めた $\{y\}$ を用いれば $\{b\} = [U^{-1}]\{y\}$ であり, $\{b\}$ は以下に示す上三角行列を係数マトリックスとする連立 1 次方程式の解である.

$$[U]\{b\} = \{y\} \tag{1.55}$$

このような連立 1 次方程式は後退消去あるいは後退代入により求解される.

三角分解が完了していれば, 式 (1.48) 以下の操作は三角分解を行うよりもはるかに少ない計算量で実行できる. また, 異なる右辺のベクトルが多数存在する場合では, 同じ係数マトリックスを用いて非常に効率良く解を求めることが可能である.

また 1.1 節で示した例では最終的に得られた係数マトリックス $[A^{(4)}]$ は $[S]$, 右辺のベクトル $\{c^{(4)}\}$ は $\{x\} = [L^{-1}]\{c\}$ に対応している.

式 (1.1) について実際に計算してみると, 以下のようになる.

$$[L] = \begin{bmatrix} 1 & & & 0 \\ -\frac{4}{5} & 1 & & \\ \frac{1}{5} & -\frac{8}{7} & 1 & \\ 0 & \frac{5}{14} & -\frac{4}{3} & 1 \end{bmatrix} \tag{1.56}$$

$$[D] = \begin{bmatrix} 5 & & & 0 \\ & \frac{14}{5} & & \\ & & \frac{15}{7} & \\ 0 & & & \frac{5}{6} \end{bmatrix} \tag{1.57}$$

$$[U] = [L^T] = \begin{bmatrix} 1 & -\frac{4}{5} & \frac{1}{5} & 0 \\ & 1 & -\frac{8}{7} & \frac{5}{14} \\ & & 1 & -\frac{4}{3} \\ \Large 0 & & & 1 \end{bmatrix} \tag{1.58}$$

$$[L^{-1}] = [L_{n-1}^{-1}] \cdots [L_2^{-1}][L_1^{-1}] = \begin{bmatrix} 1 & & & \\ \frac{4}{5} & 1 & & \Large 0 \\ \frac{5}{7} & \frac{8}{7} & 1 & \\ \frac{2}{3} & \frac{7}{6} & \frac{4}{3} & 1 \end{bmatrix} \tag{1.59}$$

$$[L^{-1}]\{c\} = \begin{bmatrix} 1 & & & \Large 0 \\ \frac{4}{5} & 1 & & \\ \frac{5}{7} & \frac{8}{7} & 1 & \\ \frac{2}{3} & \frac{7}{6} & \frac{4}{3} & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ \frac{8}{7} \\ \frac{7}{6} \end{bmatrix} = \{c^{(4)}\} \tag{1.60}$$

1.3　Gauss の消去法のコーディング — 基礎編

以上で説明した Gauss の消去法を実際にコーディングする. ここで注意しなければならないのは数値計算のプログラムはしっかりしたプランをつくってからコーディングを行わないとデバッグが事実上不可能だと言うことである. これは一般に数値計算のプログラムは変数が多く, またそのほとんどが実数型で手計算では $\frac{5}{17}$ であっても変数の値としては $0.2941176\ldots$ となりこれが $\frac{5}{17}$ であるとはにわかに認識できないので, 正しいのか間違っているのか一見では判定できないからである. [3]

[3]「プログラマーの法則」というのが知られている. これは, 素人プログラマーの場合, その人にあるプログラムを作るのにどれぐらい時間がかかるかを尋ねて, 例えば「5 分です」と答えたとする. そのとき本当に完成するまでに必要な時間は「5 分」を『3 倍して 1 桁上げる』つまり 5 (分) × 3 = 15 (時間) である. 5 分でできそうなプログラムに 15 時間かかるのはプランを立ててからコーディングしないからである. 5 分でタイプできてもデバッグに 15 時間かかることなどざらにある. エキスパートならまず 10 分ぐらいできっちりした仕様書を作り, それから 5 分でタイプし, コンパイルエラーをとり, 5 分で仕様を満たしているか確認作業を行う. こうすれば後から思わぬバグが発覚し, 苦しむことも滅多にない.

1.3.1 三角行列の積

Gauss の消去法のコーディングにはさまざまな流儀のようなものがある. ここで取り上げるものはその中の一つに過ぎないが

1. 何をどう計算しているか分かりやすい.
2. 後に導入する, skyline 法と呼ばれる手法と基本的に同じ構造である.

という特徴がある. 1.2 節では, 一般のマトリックスが三角分解できることを示した. ここでは逆に三角分解されたマトリックスから出発する. 例として以下の 4×4 のマトリックスを考える.

$$[L] = \begin{bmatrix} 1 & & & 0 \\ L_{21} & 1 & & \\ L_{31} & L_{32} & 1 & \\ L_{41} & L_{42} & L_{43} & 1 \end{bmatrix} \tag{1.61}$$

$$[D] = \begin{bmatrix} D_{11} & & & 0 \\ & D_{22} & & \\ & & D_{33} & \\ 0 & & & D_{44} \end{bmatrix} \tag{1.62}$$

$$[U] = \begin{bmatrix} 1 & U_{12} & U_{13} & U_{14} \\ & 1 & U_{23} & U_{24} \\ & & 1 & U_{34} \\ 0 & & & 1 \end{bmatrix} \tag{1.63}$$

この $[L]$, $[D]$, $[U]$ の積をとると $[A] = [L][D][U]$ として以下のようになる.

$$[A] = \begin{bmatrix} D_{11} & D_{11}U_{12} & D_{11}U_{13} & D_{11}U_{14} \\ L_{21}D_{11} & L_{21}D_{11}U_{12} + D_{22} & L_{21}D_{11}U_{13} + D_{22}U_{23} & L_{21}D_{11}U_{14} + D_{22}U_{24} \\ L_{31}D_{11} & L_{31}D_{11}U_{12} + L_{32}D_{22} & L_{31}D_{11}U_{13} + L_{32}D_{22}U_{23} + D_{33} & L_{31}D_{11}U_{14} + L_{32}D_{22}U_{24} + D_{33}U_{34} \\ L_{41}D_{11} & L_{41}D_{11}U_{12} + L_{42}D_{22} & L_{41}D_{11}U_{13} + L_{42}D_{22}U_{23} + L_{43}D_{33} & L_{41}D_{11}U_{14} + L_{42}D_{22}U_{24} + L_{43}D_{33}U_{34} + D_{44} \end{bmatrix} \tag{1.64}$$

これは一般に以下のようにインデックスを用いて表すことができる.

$$A_{11} = D_{11} \tag{1.65}$$

$$A_{1i} = D_{11}U_{1i} \qquad i = 2 \sim n \tag{1.66}$$

$$A_{i1} = L_{i1}D_{11} \qquad i = 2 \sim n \tag{1.67}$$

$i \geq 2$ で

$$A_{ij} = D_{ii}U_{ij} + \sum_{k=1}^{i-1} L_{ik}D_{kk}U_{kj} \quad (i < j : \text{上三角側}) \tag{1.68}$$

$$A_{ij} = L_{ij}D_{jj} + \sum_{k=1}^{j-1} L_{ik}D_{kk}U_{kj} \quad (i > j : \text{下三角側}) \tag{1.69}$$

$$A_{ii} = D_{ii} + \sum_{k=1}^{i-1} L_{ik}D_{kk}U_{ki} \qquad (i = j) \tag{1.70}$$

1.3.2　三角分解の手順

この関係を用いると $[L]$, $[D]$, $[U]$ は以下のように順次計算できる.

$j = 1$

$$D_{11} = A_{11} \tag{1.71}$$

$j = 2$

$$\begin{cases} U_{12} = A_{12}/D_{11} \\ L_{21} = A_{21}/D_{11} \end{cases} \tag{1.72}$$

$$D_{22} = A_{22} - L_{21}D_{11}U_{12} \tag{1.73}$$

$j = 3$

$$\begin{cases} U_{13} = A_{13}/D_{11} \\ L_{31} = A_{31}/D_{11} \end{cases} \tag{1.74}$$

$$\begin{cases} U_{23} = (A_{23} - L_{21}D_{11}U_{13})/D_{22} \\ L_{32} = (A_{32} - L_{31}D_{11}U_{12})/D_{22} \end{cases} \tag{1.75}$$

$$D_{33} = A_{33} - L_{31}D_{11}U_{13} - L_{32}D_{22}U_{23} \tag{1.76}$$

$$\vdots$$

これを実際にコーディングするようにインデックスで書けば以下のようになる.

for $j = 1$

$D_{11} = A_{11}$

for $j = 2$

$U_{12} = A_{12}/D_{11}$

$L_{21} = A_{21}/D_{11}$

$D_{22} = A_{22} - L_{21}D_{11}U_{12}$

for $j = 3 \sim n$

$U_{1j} = A_{1j}/D_{11}$

$L_{j1} = A_{j1}/D_{11}$

for $i = 2 \sim j - 1$

for $k = 1 \sim i - 1$

$\texttt{temp\_}U = \texttt{temp\_}U + L_{ik}D_{kk}U_{kj}$

$\texttt{temp\_}L = \texttt{temp\_}L + L_{jk}D_{kk}U_{ki}$

end for

$U_{ij} = (A_{ij} - \texttt{temp\_}U)/D_{ii}$

$L_{ji} = (A_{ji} - \texttt{temp\_}L)/D_{ii}$

end for

for $k = 1 \sim j-1$

$\texttt{temp} = \texttt{temp} + L_{jk}D_{kk}U_{kj}$

end for

$D_{jj} = A_{jj} - \texttt{temp}$

end for

以上で得られた三角行列から未知ベクトルを求める．ここでは下三角行列は前進代入, 上三角行列は後退消去を用いる．上, 下三角行列の対角項は 1 であることを考慮すると, 以下のようなコーディングになる．

下三角行列 (前進代入)

for $i = 1$

$x_1 = c_1$

for $i = 2 \sim n$

for $j = 1 \sim i-1$

$\texttt{temp} = \texttt{temp} + L_{ij}x_j$

end for

$x_i = c_i - \texttt{temp}$

end for

対角項

for $i = 1 \sim n$

$y_i = x_i/D_{ii}$

end for

上三角行列 (後退消去)

for $j = n$

$b_n = y_n$

for $j = n \sim 2$ **step** $= -1$

for $i = 1 \sim j - 1$

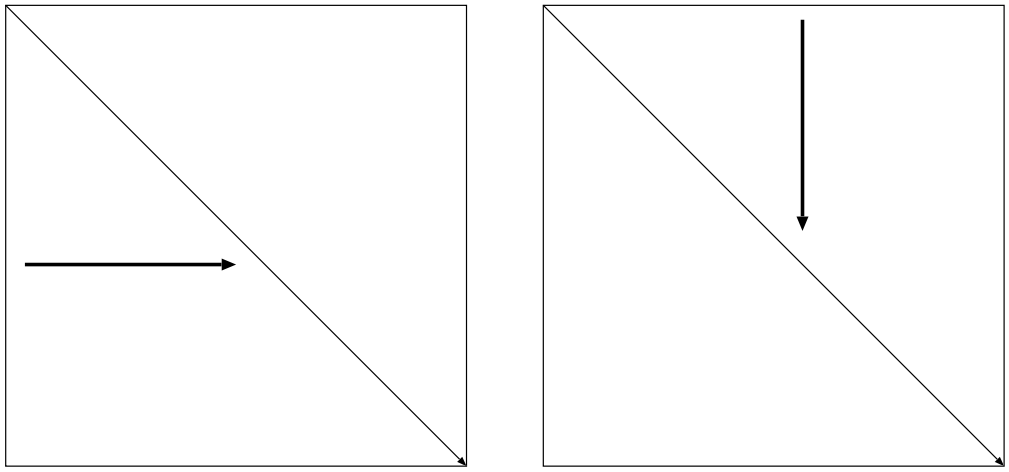
$y_i = y_i - U_{ij} b_j$

end for

$b_{j-1} = y_{j-1}$

end for

このようなコーディングを行うと図 1.1 に示すように下三角行列については行を, 上三角行列については列をスイープすることになり, 前述の三角分解の場合と同様になる. 後述する skyline 法ではこの特徴を用いて効率化を図る.



下三角行列に前進代入を用いる　　上三角行列に後退消去を用いる

図 1.1: スイープの方向

1.3.3　演習 Gauss の消去法 (バージョン 1)

以下のプログラムを作れ.

(1) 下式のような形式のマトリックスを作る. これを $[A]$ とする.

$$\begin{bmatrix} 1 & 2 & 3 & \dots & n \\ 2 & 2 & 3 & & n \\ 3 & 3 & 3 & & n \\ \vdots & & & \ddots & \vdots \\ n & n & n & \dots & n \end{bmatrix}$$

(2) $\{b\} = \{1, 2, \dots, n\}^T$ として

$$[A]\{b\} = \{c\}$$

により $\{c\}$ を求める.

(3) $[A]\{b\} = \{c\}$ の連立 1 次方程式を解いて $\{b\}$ を求め, $\{1, 2, \dots, n\}^T$ となっていることを確認する.

(4) 1000×1000 のマトリックスにして, cpu time を計測する. プログラム中で用いる変数は実数は全て倍精度にする. まずプログラムの先頭でマトリックスの次元 n を定数として宣言する. その他の変数名などは以下のようにする.

定式化	変数名	次元	大きさ
係数マトリックス A	a	2	n,n
未知ベクトル b	b	1	n
右辺ベクトル c	c	1	n
三角分解 L	al	2	n,n
三角分解 D	ad	2	n,n
三角分解 U	au	2	n,n
中間 x	x	1	n
中間 y	y	1	n

数値計算のプログラミングに慣れた人ならこの配列のとり方が非常に無駄が多くパフォーマンスを劣化させることが予想できると思うが, ここではコーディングを容易にするためにこのようにする. またバージョン 2 以降で, この配列などの合理化を行いプログラムを加速する.

1.3.4 fortran コーディング例

fortran でのコーディング例を示すと以下のようになる. サブルーチン init で係数マトリックスと右辺ベクトルを作成する gauss の消去法のサブルーチンは gauss_ver1 である.

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o misc.o gauss_ver1.o

TARGET = gauss_ver1

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
c gauss elimination ver. 1
c     by WATANABE Hiroshi (2000 June)
c
      implicit real*8(a-h,o-z)
      parameter(n = 1000)
      parameter(i_check = 1)
      dimension a(n,n),b(n),c(n),al(n,n),au(n,n),ad(n,n),x(n),y(n)
c
      call init(a,b,c,n)
c
      if (i_check .ge. 2) call check_matrix(a,b,c,n)
c
      call gauss_ver1(a,b,c,al,au,ad,x,y,n)
```

```
c
      if (i_check .ge. 1) call check_solution(b,n)
c
      stop
      end
c
```

misc.f

```
c#########################################################################
      subroutine init(a,b,c,n)
c#########################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),b(*),c(*)
c
      do i = 1,n
          do j = 1,i
              a(i,j) = dfloat(i)
              a(j,i) = dfloat(i)
          end do
      end do
c
      do i = 1,n
          b(i) = dfloat(i)
      end do
c
      do i = 1,n
          temp = 0.D0
          do j = 1,n
              temp = temp + a(i,j) * b(j)
         end do
         c(i) = temp
      end do
c
      return
      end
c
c#########################################################################
      subroutine check_matrix(a,b,c,n)
c#########################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),b(*),c(*)
c
      write(*,*) 'matrix A'
      do i = 1,n
```

```
          write(*,1000) (a(i,j),j=1,n)
      end do
      write(*,*) 'vector b'
      write(*,1000) (b(i),i=1,n)
      write(*,*) 'vector c'
      write(*,1000) (c(i),i=1,n)
c
 1000 format(8e10.3)

      return
      end
c
c#########################################################################
      subroutine check_solution(b,n)
c#########################################################################
      implicit real*8(a-h,o-z)
      dimension b(*)
c
      write(*,*) 'solution vector b'
      write(*,1000) (b(i),i=1,n)
c
 1000 format(8e10.3)
c
      return
      end
c
```

gauss_ver1.f

```
c#########################################################################
      subroutine gauss_ver1(a,b,c,al,au,ad,x,y,n)
c#########################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),b(*),c(*),al(n,*),au(n,*),ad(n,*),x(*),y(*)
c
      ad(1,1) = a(1,1)
c
      au(1,2) = a(1,2) / ad(1,1)
      al(2,1) = a(2,1) / ad(1,1)
c
      ad(2,2) = a(2,2) - al(2,1) * ad(1,1) * au(1,2)
c
      do j = 3,n
          au(1,j) = a(1,j) / ad(1,1)
          al(j,1) = a(j,1) / ad(1,1)
```

```
          do i = 2, j-1
              temp_u = 0.d0
              temp_l = 0.d0
              do k = 1, i-1
                  temp_u = temp_u
     1                   + al(i,k) * ad(k,k) * au(k,j)
                  temp_l = temp_l
     1                   + al(j,k) * ad(k,k) * au(k,i)
              end do
              au(i,j) = (a(i,j) - temp_u) / ad(i,i)
              al(j,i) = (a(j,i) - temp_l) / ad(i,i)
          end do
          temp = 0.d0
          do k = 1,j-1
              temp = temp + al(j,k) * ad(k,k) * au(k,j)
          end do
          ad(j,j) = a(j,j) - temp
      end do
c
c
      x(1) = c(1)
      do i = 2,n
          temp = 0.d0
          do j = 1,i-1
              temp = temp + al(i,j) * x(j)
          end do
          x(i) = c(i) - temp
      end do
c
      do i = 1,n
          y(i) = x(i) / ad(i,i)
      end do
c
      b(n) = y(n)
      do j = n,2,-1
          do i = 1,j-1
              y(i) = y(i) - au(i,j) * b(j)
          end do
          b(j-1) = y(j-1)
      end do
c
      return
      end
c
```

1.3.5 C コーディング例

C で数値計算のプログラミングをする際に問題になることは, 一般にベクトルやマトリックスの添字が 1 からはじまるように定式化を進めるのに対して, C の配列の添字は 0 から始まることである. 例えば, n 元のベクトル x_i を 配列 `x` を用いて表すとすれば, x_1 = `x[0]`, x_2 = `x[1]`, $\cdots$, x_n = `x[n-1]` と対応させることになり, 配列の添字もループインデックスも定式化で導いたものをそのまま用いることができない.

これを解決するには, たとえば以下のような方法が考えられる.

(1) もともとの定式化をする際に, 添字が 0 から始まるよう書き直す.

(2) 配列の領域を余分にとって 0 のところをアクセスしない.

(1) の方法は直接的で C らしいコードが書けるという利点があるが, ベクトルやマトリックスの添字が 0 から始まるような定式化は, あまり一般的ではない. 即ち参考文献などに掲載されているものをそのまま利用できないので, 定式化の段階でバグをつくる可能性がある. また, 定式化においてベクトル, マトリックスの添字を 1 から始め, コーディングを するときに配列の添字とループインデックスを 1 シフトするというやり方も あるにはあるが, よほどプログラミングに習熟した人ならいざ知らず, 初心者がこれを試みるのは前述の「プログラマーの法則」を実行するようなものである.

(2) の方法は定式化はやり直さなくて良いが当然ながら無駄なメモリーを使用するし, もし fortran のプログラムや, 他のライブラリーとリンクしたいときに, 配列の整合性がとれていないという問題を生じる.

ここでは多少テクニカルな方法であるがポインターの操作によって無駄なメモリーをとることなく, またループインデックスの変更もないようなコーディングを採用する. ポインターは初心者には難解で, バグの原因になりやすいが, C では配列とポインターは完全に同じもので, 下記にも示すようにポインターとして宣言した変数を配列として取り扱うことが可能である.

さて, 一般的な方法だと, 最初にポインター型の変数を宣言し, 次に, 記憶するのに必要なメモリーを確保し, 配列の初期化をする, という手順になる. 例えば, $x_i(x_1 = 1.0, x_2 = 2.0, x_3 = 3.0, \cdots, x_n = n)$ を記憶するために倍精度の 1 次元配列 `x` を用いるとする.

```
double *x;
x = (double *)malloc(sizeof(double)*n);
for (i = 0; i < n; ++i){
    x[i] = (double)(i+1);
}
```

前述のように, このようにコーディングすると $x_i = i =$ x[i-1] となり添字がずれることになりコードが分かりにくくなる. そこでポインターの利点を活用する. ポインターはその名のとおり実際にはあるアドレスを指し示しているだけの変数で上記のようにメモリーを確保してはじめて配列としての機能を果たす. malloc によって確保した領域の先頭のアドレスがx に代入され, x だけならアドレス, *(x), あるいは x[0] で, x が指している領域に格納されている値が参照できる.

ここで, もう一つ別にポインター変数 xx を宣言し, x が指しているアドレスより 1 だけ引いたアドレスを代入する. このとき *(xx+1) が *(x) と同じ値を指している. これは配列にした場合, xx[1] が, x[0] と同じ値を指していると言うことを意味する. 即ち配列 xx の添字の範囲は 1 から n である. これにもとづき上記のプログラムを書き直すと以下のようになる. このときプログラムで明示的に値を代入している配列 xx 以外にもx にも値が代入されており, $x_i = i =$ x[i-1] = xx[i] となっている.

```
double *x, *xx;
x = (double *)malloc(sizeof(double)*n);
xx = x - 1;
for (i = 1; i <= n; ++i){
    xx[i] = (double)(i);
}
```

マトリックスは 2 次元配列ではなく, 多少見辛くなるが 1 次元配列を用いて記憶する. たとえば以下のようなマトリックスを記憶する場合を考える.

$$[A] = \begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{bmatrix}$$

ここでは, fortran で記述されたプログラムや数値演算ライブラリーなどとのリンクを想定し, マトリックスを列ベクトルの集合と考えて, まず第 1 列の要素を`a[0]` から `a[n-1]` に記憶し, 次に第 2 列の要素を`a[0 + n]` から `a[n-1 + n]` に記憶する. これを以下順に繰り返えせば, i 行 j 列の成分は配列の $i-1+(j-1)\times n$ 番目の要素, すなわち `a[i-1+(j-1)*n]` として記憶する.

```
double *a;
a = (double *)malloc(sizeof(double)*n*n);
for (j = 0; j < n; ++j){
    for (i = 0; i < n; ++i){
        a[i + j * n] = (double)((i+1)*10 + (j+1));
    }
}
```

ベクトルと同様に, もう一つ別にポインター変数 `aa` を宣言し, `a` が指しているアドレスより `1+n` だけ引いたアドレスを代入する. このとき `*(aa+1+1*n)` が `*(a)` と同じ値を指しており, 配列にした場合, `aa[i+j*n]` と, `a[(i-1)+(j-1)*n]` が対応していると言うことを意味する. これにもとづき上記のプログラムを書き直すと以下のようになる.

```
double *a, *aa;
a = (double *)malloc(sizeof(double)*n*n);
aa = a - n - 1;
for (j = 1; j <= n; ++j){
    for (i = 1; i <= n; ++i){
        aa[i + j * n] = (double)(i*10 + j);
```

```
        }
    }
```

この方法でベクトル, マトリックスを表したコーディング例を示すと以下のようになる. 関数 init で係数マトリックスと右辺ベクトルを作成する gauss の消去法の関数は gauss_ver1 である.

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o misc.o gauss_ver1.o

TARGET = gauss_ver1

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
/* gauss elimination ver. 1
     by WATANABE Hiroshi (2000 July) */

#include <stdio.h>
#define N 1000
#define I_CHECK 1

void init(double *a, double *b, double *c, int n);
void check_matrix(double *a, double *b, double *c, int n);
void gauss_ver1(double *a, double *b, double *c, double *al,
                double *au, double *ad, double *x, double *y, int n);
void check_solution(double *b, int n);

main()
{
    double *a,*b,*c,*al,*au,*ad,*x,*y;
    int n;
```

```
    n = N;

    a  = (double *)malloc(sizeof(double)*n*n);
    al = (double *)malloc(sizeof(double)*n*n);
    au = (double *)malloc(sizeof(double)*n*n);
    ad = (double *)malloc(sizeof(double)*n*n);
    b  = (double *)malloc(sizeof(double)*n);
    c  = (double *)malloc(sizeof(double)*n);
    x  = (double *)malloc(sizeof(double)*n);
    y  = (double *)malloc(sizeof(double)*n);

    init(a,b,c,n);

    if (I_CHECK >= 2) check_matrix(a,b,c,n);

    gauss_ver1(a,b,c,al,au,ad,x,y,n);

    if (I_CHECK >= 1) check_solution(b,n);
}
```

misc.c

```
void init(double *arg_a, double *arg_b, double *arg_c, int n)
{
    double *a, *b, *c;
    int i,j, ij, ji;
    double temp;

    a = arg_a-1-n;
    b = arg_b-1;
    c = arg_c-1;

    for (i = 1; i <= n; ++i){
        for (j = 1; j <= n; ++j){
            a[i+j*n] = (double)(i);
            a[j+i*n] = (double)(i);
        }
    }

    for (i = 1; i <= n; ++i){
        b[i] = (double)(i);
    }

    for (i = 1; i <= n; ++i){
        temp = 0.0;
```

```
        for (j = 1; j <= n; ++j){
            temp = temp + a[i+j*n] * b[j];
        }
        c[i] = temp;
    }
}

void check_matrix(double *arg_a, double *arg_b, double *arg_c, int n)
{
    double *a, *b, *c;
    int i,j;

    a = arg_a-1-n;
    b = arg_b-1;
    c = arg_c-1;

    printf("matrix A\n");
    for (i = 1; i <= n; ++i){
        for (j = 1; j <= n; ++j) printf("% 10.3E",a[i+j*n]); printf("\n");
    }

    printf("vector b\n");
    for (i = 1; i <= n; ++i) printf("% 10.3E",b[i]); printf("\n");

    printf("vector c\n");
    for (i = 1; i <= n; ++i) printf("% 10.3E",c[i]); printf("\n");
}

void check_solution(double *arg_b, int n)
{
    double *b;
    int i;

    b = arg_b-1;

    printf("solution vector b\n");
    for (i = 1; i <= n; ++i) printf("% 10.3E",b[i]); printf("\n");
}
```

gauss_ver1.c

```
void gauss_ver1(double *arg_a, double *arg_b, double *arg_c,
                double *arg_al, double *arg_au, double *arg_ad,
                double *arg_x, double *arg_y, int n)
{
```

```
    double *a, *b, *c, *al, *au, *ad, *x, *y;
    int i, j, k;
    double temp_u, temp_l, temp;

    a  = arg_a -1-n;
    al = arg_al-1-n;
    au = arg_au-1-n;
    ad = arg_ad-1-n;

    b = arg_b-1;
    c = arg_c-1;
    x = arg_x-1;
    y = arg_y-1;

    ad[1+1*n] = a[1+1*n];
    au[1+2*n] = a[1+2*n] / ad[1+1*n];
    al[2+1*n] = a[2+2*n] / ad[1+1*n];

    ad[2+2*n] = a[2+2*n] - al[2+1*n] * ad[1+1*n] * au[1+2*n];

    for (j = 3; j <= n; ++j){
        au[1+j*n] = a[1+j*n] / ad[1+1*n];
        al[j+1*n] = a[j+1*n] / ad[1+1*n];
        for (i = 2; i <= j-1; ++i){
            temp_u = 0.0;
            temp_l = 0.0;
            for (k = 1; k <= i-1; ++k){
                temp_u = temp_u + al[i+k*n] * ad[k+k*n] * au[k+j*n];
                temp_l = temp_l + al[j+k*n] * ad[k+k*n] * au[k+i*n];
            }
            au[i+j*n] = (a[i+j*n] - temp_u) / ad[i+i*n];
            al[j+i*n] = (a[j+i*n] - temp_l) / ad[i+i*n];
        }
        temp = 0.0;
        for (k = 1; k <= j-1; ++k){
            temp = temp + al[j+k*n] * ad[k+k*n] * au[k+j*n];
        }
        ad[j+j*n] = a[j+j*n] - temp;
    }

    x[1] = c[1];
    for (i = 2; i <= n; ++i){
        temp = 0.0;
        for (j = 1; j <= i-1; ++j){
            temp = temp + al[i+j*n] * x[j];
```

```
        }
        x[i] = c[i] - temp;
    }

    for (i = 1; i <= n; ++i){
        y[i] = x[i] / ad[i+i*n];
    }

    b[n] = y[n];
    for (j = n; j >= 2; --j){
        for (i = 1; i <= j-1; ++i){
            y[i] = y[i] - au[i+j*n] * b[j];
        }
        b[j-1] = y[j-1];
    }
}
```

1.4　Gauss の消去法のコーディング — 配列の合理化 (1)

数値計算のプログラムで最も大切なことはもちろん正しい解を求める事であるが, それと同時になるべく効率よく, 短時間で計算を終ることも重要である. そのために多少わかりにくいコードを書くことがある. その代表とも言えるものが計算で使用する配列を共用して使用するメモリーを節約する, と言うものであろう.

1.4.1　右辺のメモリーの節約

例えば, バージョン 1 の前進代入のループの部分を展開してみると以下のようになる.

$$x(1) = c(1) \tag{1.77}$$
$$\texttt{temp} = \texttt{0.D0} \tag{1.78}$$
$$\texttt{temp} = al(2,1)x(1) \tag{1.79}$$
$$x(2) = c(2) - \texttt{temp} \tag{1.80}$$
$$\texttt{temp} = \texttt{0.D0} \tag{1.81}$$
$$\texttt{temp} = al(3,1)x(1) + al(3,2)x(2) \tag{1.82}$$
$$x(3) = c(3) - \texttt{temp} \tag{1.83}$$
$$\vdots$$

これで分かるように $x(i)$ を求めるのに必要な値は, $x(1), \ldots, x(i-1)$ 及び $c(i)$ であり, $c(1), \ldots, c(i-1)$ は必要ない. そこで, $x(i)$ としてもとめられた量を順次 $c(i)$ に上書きするようにして記憶するようにすれば配列 x を用いずにコーディングすることができる. 後退消去の部分も配列 y を用いることなく計算することができ最終的には左辺の値を記憶する配列が一つあれば, その配列に計算結果を代入するようにコーディングできる.

バージョン 2.1 として左辺値を記憶した配列 (たとえば $\{c\}$) を Gauss の消去法のサブルーチン (または関数) にわたし, 計算終了後はその配列に解が入っているようにコーディングする. また 1000×1000 のマトリックスにしたときの cpu time を計測する.

用いる変数はバージョン 1 と同じとする. またこのとき係数マトリクスと左辺をつくるサブルーチンはバージョン 1 と同じものを用いることができる. 用いる OS やコンパイラにもよるがバージョン 1 より少しだけ加速しているはずである.

1.4.2 fortran でのコーディング例

fortran でのコーディング例を示すと以下のようになる. gauss_ver1 の前進代入などの部分を変更したサブルーチンが gauss_ver21 である.

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o misc.o gauss_ver21.o

TARGET = gauss_ver21

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
c gauss elimination ver. 2.1
c     by WATANABE Hiroshi (2000 June)
c
      implicit real*8(a-h,o-z)
      parameter(n = 1000)
      parameter(i_check = 1)
      dimension a(n,n),b(n),c(n),al(n,n),au(n,n),ad(n,n)
c
      call init(a,b,c,n)
c
      if (i_check .ge. 2) call check_matrix(a,b,c,n)
c
      call gauss_ver21(a,c,al,au,ad,n)
c
      if (i_check .ge. 1) call check_solution(c,n)
c
      stop
      end
c
```

misc.f　gauss_ver1 と共通

gauss_ver21.f

```
c************************************************************************
      subroutine gauss_ver21(a,c,al,au,ad,n)
c************************************************************************
```

```
      implicit real*8(a-h,o-z)
      dimension a(n,*),c(*),al(n,*),au(n,*),ad(n,*)
c
      ad(1,1) = a(1,1)
c
      au(1,2) = a(1,2) / ad(1,1)
      al(2,1) = a(2,1) / ad(1,1)
c
      ad(2,2) = a(2,2) - al(2,1) * ad(1,1) * au(1,2)
c
      do j = 3,n
          au(1,j) = a(1,j) / ad(1,1)
          al(j,1) = a(j,1) / ad(1,1)
          do i = 2, j-1
              temp_u = 0.d0
              temp_l = 0.d0
              do k = 1, i-1
                  temp_u = temp_u
     1                 + al(i,k) * ad(k,k) * au(k,j)
                  temp_l = temp_l
     1                 + al(j,k) * ad(k,k) * au(k,i)
              end do
              au(i,j) = (a(i,j) - temp_u) / ad(i,i)
              al(j,i) = (a(j,i) - temp_l) / ad(i,i)
          end do
          temp = 0.d0
          do k = 1,j-1
              temp = temp + al(j,k) * ad(k,k) * au(k,j)
          end do
          ad(j,j) = a(j,j) - temp
      end do
c
c
      do i = 2,n
          temp = 0.d0
          do j = 1,i-1
              temp = temp + al(i,j) * c(j)
          end do
          c(i) = c(i) - temp
      end do
c
      do i = 1,n
          c(i) = c(i) / ad(i,i)
      end do
c
```

```
      do j = n,2,-1
          do i = 1,j-1
              c(i) = c(i) - au(i,j) * c(j)
          end do
      end do
c
      return
      end
c
```

1.4.3　C でのコーディング例

C でのコーディング例を示すと以下のようになる. gauss_ver1 の前進代入などの部分を変更した関数が gauss_ver21 である.

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o misc.o gauss_ver21.o

TARGET = gauss_ver21

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
/* gauss elimination ver. 2.1
     by WATANABE Hiroshi (2000 July) */

#include <stdio.h>
#define N 1000
#define I_CHECK 1

void init(double *a, double *b, double *c, int n);
void check_matrix(double *a, double *b, double *c, int n);
```

```
void gauss_ver21(double *a, double *c, double *al,
                 double *au, double *ad, int n);
void check_solution(double *b, int n);

main()
{
    double *a,*b,*c,*al,*au,*ad;
    int n;

    n = N;

    a  = (double *)malloc(sizeof(double)*n*n);
    al = (double *)malloc(sizeof(double)*n*n);
    au = (double *)malloc(sizeof(double)*n*n);
    ad = (double *)malloc(sizeof(double)*n*n);
    b  = (double *)malloc(sizeof(double)*n);
    c  = (double *)malloc(sizeof(double)*n);

    init(a,b,c,n);

    if (I_CHECK >= 2) check_matrix(a,b,c,n);

    gauss_ver21(a,c,al,au,ad,n);

    if (I_CHECK >= 1) check_solution(b,n);
}
```

misc.c gauss_ver1 と共通

gauss_ver21.c

```
void gauss_ver21(double *arg_a, double *arg_c, double *arg_al,
                 double *arg_au, double *arg_ad, int n)
{
    double *a, *c, *al, *au, *ad;
    int i, j, k;
    double temp_u, temp_l, temp;

    a  = arg_a -1-n;
    al = arg_al-1-n;
    au = arg_au-1-n;
    ad = arg_ad-1-n;

    c = arg_c-1;
```

```
    ad[1+1*n] = a[1+1*n];
    au[1+2*n] = a[1+2*n] / ad[1+1*n];
    al[2+1*n] = a[2+2*n] / ad[1+1*n];

    ad[2+2*n] = a[2+2*n] - al[2+1*n] * ad[1+1*n] * au[1+2*n];

    for (j = 3; j <= n; ++j){
        au[1+j*n] = a[1+j*n] / ad[1+1*n];
        al[j+1*n] = a[j+1*n] / ad[1+1*n];
        for (i = 2; i <= j-1; ++i){
            temp_u = 0.0;
            temp_l = 0.0;
            for (k = 1; k <= i-1; ++k){
                temp_u = temp_u + al[i+k*n] * ad[k+k*n] * au[k+j*n];
                temp_l = temp_l + al[j+k*n] * ad[k+k*n] * au[k+i*n];
            }
            au[i+j*n] = (a[i+j*n] - temp_u) / ad[i+i*n];
            al[j+i*n] = (a[j+i*n] - temp_l) / ad[i+i*n];
        }
        temp = 0.0;
        for (k = 1; k <= j-1; ++k){
            temp = temp + al[j+k*n] * ad[k+k*n] * au[k+j*n];
        }
        ad[j+j*n] = a[j+j*n] - temp;
    }

    for (i = 2; i <= n; ++i){
        temp = 0.0;
        for (j = 1; j <= i-1; ++j){
            temp = temp + al[i+j*n] * c[j];
        }
        c[i] = c[i] - temp;
    }

    for (i = 1; i <= n; ++i){
        c[i] = c[i] / ad[i+i*n];
    }

    for (j = n; j >= 2; --j){
        for (i = 1; i <= j-1; ++i){
            c[i] = c[i] - au[i+j*n] * c[j];
        }
    }
}
```

1.5 Gauss の消去法のコーディング — 配列の合理化 (2)

1.5.1 係数マトリックス用配列の節約

Gauss の消去法では計算時間のほとんどは三角分解に費やされるので, 三角分解で用いられる配列を合理化し加速を試みる. 今回コーディングしたようなループインデックスだと図 1.2(a) のように左上すなわち A_{11} 成分からはじまり, 順に三角分解済みの領域を正方形状に広げるように行なわれる.

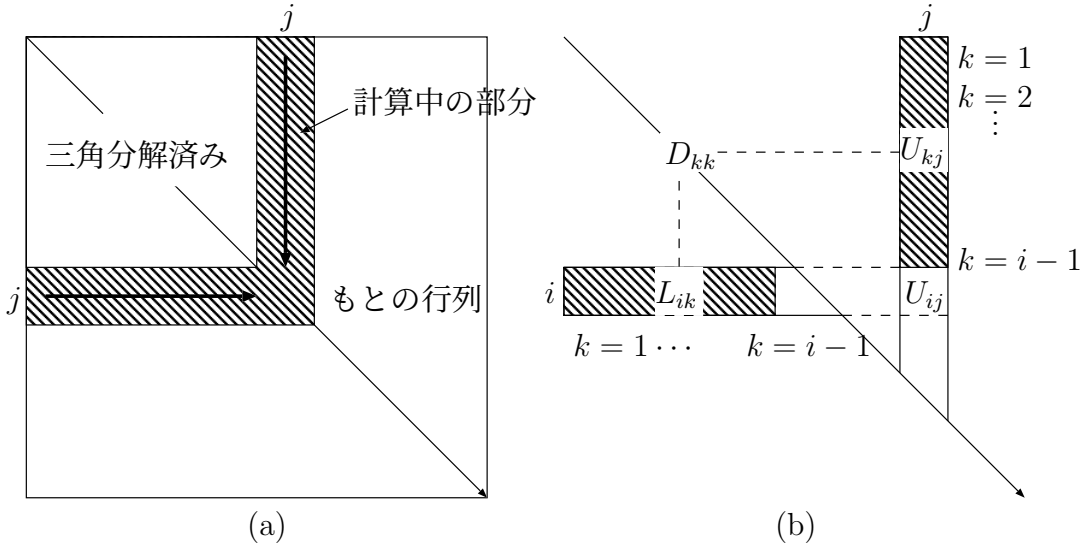


図 1.2: 三角分解の進み方

ある $U_{ij}(i \neq j)$ を求めるとき必要な値は A_{ij} と, 図 1.2(b) にハッチングで示された $U_{kj}(k = 1 \sim i-1), L_{ik}(k = 1 \sim i-1), D_{kk}(k = 1 \sim i-1)$ であり, 三角分解された部分の $[A]$ の値は参照されることはない. そのため, この領域を有効に活用する. 記憶しないといけない U_{kj}, L_{ik}, D_{kk} は全てこの領域に含まれており, $[U], [L]$ の対角項は参照されることはない. したがって $[A]$ の対角項に $[D]$, 上下三角部分に $[U], [L]$ の対角項以外を記憶す

ることにより $[U], [D], [L]$ のためだけの配列を用いることなく三角分解を行うことができる. 対角項の計算も同様である.

バージョン 2.2 として係数マトリックス を記憶した配列のみを用いて三角分解を行うようにコーディングする. また 1000×1000 のマトリックスにしたときの cpu time を計測する.

用いる変数はバージョン 2.1 と同じとする. 用いる OS やコンパイラにもよるがバージョン 2.1 よりもかなり加速するはずである.

1.5.2 fortran コーディング例

fortran でのコーディング例を示すと以下のようになる. gauss_ver21 の三角分解の配列を合理化したサブルーチンが gauss_ver22 である.

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o misc.o gauss_ver22.o

TARGET = gauss_ver22

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
c gauss elimination ver. 2.2
c     by WATANABE Hiroshi (2000 June)
c
      implicit real*8(a-h,o-z)
      parameter(n = 1000)
      parameter(i_check = 1)
c
      dimension a(n,n),b(n),c(n)
```

```
c
      call init(a,b,c,n)
c
      if (i_check .ge. 2) call check_matrix(a,b,c,n)
c
      call gauss_ver22(a,c,n)
c
      if (i_check .ge. 1) call check_solution(c,n)
c
      stop
      end
c
```

misc.f gauss_ver1 と共通

gauss_ver22.f

```
c#########################################################################
      subroutine gauss_ver22(a,c,n)
c#########################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),c(*)
c
      a(1,1) = a(1,1)
c
      a(1,2) = a(1,2) / a(1,1)
      a(2,1) = a(2,1) / a(1,1)
c
      a(2,2) = a(2,2) - a(2,1) * a(1,1) * a(1,2)
c
      do j = 3,n
          a(1,j) = a(1,j) / a(1,1)
          a(j,1) = a(j,1) / a(1,1)
          do i = 2, j-1
              temp_u = 0.d0
              temp_l = 0.d0
              do k = 1, i-1
                  temp_u = temp_u
     1                 + a(i,k) * a(k,k) * a(k,j)
                  temp_l = temp_l
     1                 + a(j,k) * a(k,k) * a(k,i)
              end do
              a(i,j) = (a(i,j) - temp_u) / a(i,i)
              a(j,i) = (a(j,i) - temp_l) / a(i,i)
```

```
          end do
          temp = 0.d0
          do k = 1,j-1
              temp = temp + a(j,k) * a(k,k) * a(k,j)
          end do
          a(j,j) = a(j,j) - temp
      end do
c
c
      do i = 2,n
          temp = 0.d0
          do j = 1,i-1
              temp = temp + a(i,j) * c(j)
          end do
          c(i) = c(i) - temp
      end do
c
      do i = 1,n
          c(i) = c(i) / a(i,i)
      end do
c
      do j = n,2,-1
          do i = 1,j-1
              c(i) = c(i) - a(i,j) * c(j)
          end do
      end do
c
      return
      end
c
```

1.5.3　C コーディング例

C でのコーディング例を示すと以下のようになる．gauss_ver21 の三角分解の配列を合理化した関数が gauss_ver22 である．

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o misc.o gauss_ver22.o
```

```
TARGET = gauss_ver22

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
/* gauss elimination ver. 2.2
     by WATANABE Hiroshi (2000 July) */

#include <stdio.h>
#define N 1000
#define I_CHECK 1

void init(double *a, double *b, double *c, int n);
void check_matrix(double *a, double *b, double *c, int n);
void gauss_ver22(double *a, double *c, int n);
void check_solution(double *b, int n);

main()
{
    double *a,*b,*c;
    int n;

    n = N;

    a  = (double *)malloc(sizeof(double)*n*n);
    b  = (double *)malloc(sizeof(double)*n);
    c  = (double *)malloc(sizeof(double)*n);

    init(a,b,c,n);

    if (I_CHECK >= 2) check_matrix(a,b,c,n);

    gauss_ver22(a,c,n);

    if (I_CHECK >= 1) check_solution(b,n);
}
```

misc.c gauss_ver1 と共通

gauss_ver22.c

```
void gauss_ver22(double *arg_a, double *arg_c, int n)
{
    double *a, *c;
    int i, j, k;
    double temp_u, temp_l, temp;

    a  = arg_a -1-n;
    c = arg_c-1;

    a[1+2*n] = a[1+2*n] / a[1+1*n];
    a[2+1*n] = a[2+2*n] / a[1+1*n];

    a[2+2*n] = a[2+2*n] - a[2+1*n] * a[1+1*n] * a[1+2*n];

    for (j = 3; j <= n; ++j){
        a[1+j*n] = a[1+j*n] / a[1+1*n];
        a[j+1*n] = a[j+1*n] / a[1+1*n];
        for (i = 2; i <= j-1; ++i){
            temp_u = 0.0;
            temp_l = 0.0;
            for (k = 1; k <= i-1; ++k){
                temp_u = temp_u + a[i+k*n] * a[k+k*n] * a[k+j*n];
                temp_l = temp_l + a[j+k*n] * a[k+k*n] * a[k+i*n];
            }
            a[i+j*n] = (a[i+j*n] - temp_u) / a[i+i*n];
            a[j+i*n] = (a[j+i*n] - temp_l) / a[i+i*n];
        }
        temp = 0.0;
        for (k = 1; k <= j-1; ++k){
            temp = temp + a[j+k*n] * a[k+k*n] * a[k+j*n];
        }
        a[j+j*n] = a[j+j*n] - temp;
    }

    for (i = 2; i <= n; ++i){
        temp = 0.0;
        for (j = 1; j <= i-1; ++j){
            temp = temp + a[i+j*n] * c[j];
        }
        c[i] = c[i] - temp;
    }

    for (i = 1; i <= n; ++i){
        c[i] = c[i] / a[i+i*n];
```

```
    }

    for (j = n; j >= 2; --j){
        for (i = 1; i <= j-1; ++i){
            c[i] = c[i] - a[i+j*n] * c[j];
        }
    }
}
```

1.6 Gauss の消去法のコーディング — 演算の合理化

1.6.1 三角分解の演算の特徴

バージョン 2 では配列を合理化して三角分解の加速を行ったが, 今回は三角分解で行う演算の特徴を考えることにより, 演算量の軽減を試みる. $i = 1$ から順に U_{ij} を求めるプロセスを書き下すと以下のようになる.

$$
\begin{aligned}
U_{1j} &= A_{1j}/D_{11} \\
U_{2j} &= (A_{2j} - L_{21}\underline{D_{11}U_{1j}})/D_{22} \\
U_{3j} &= (A_{3j} - L_{31}\underline{D_{11}U_{1j}} - L_{32}\underline{D_{22}U_{2j}})/D_{33} \\
&\vdots \\
U_{ij} &= (A_{ij} - L_{i1}\underline{D_{11}U_{1j}} - L_{i2}\underline{D_{22}U_{2j}} - \ldots - L_{ii-1}\underline{D_{i-1i-1}U_{i-1j}})/D_{ii} \\
U_{i+1j} &= (A_{i+1j} - L_{i+11}\underline{D_{11}U_{1j}} - L_{i+12}\underline{D_{22}U_{2j}} - \ldots - L_{i+1i}\underline{D_{ii}U_{ij}})/D_{i+1i+1}
\end{aligned}
\tag{1.84}
$$

これからわかるように, $i = 2$ 以上では, U_{ij} を求めるのに用いられる $U_{kj}(k = 1 \sim i-1)$ の値はすべて D_{kk} と積をとったもので, 単独の U_{kj} の値は必要ない. そこで以下のように, まず $(D_{kk}U_{kj})$, $(L_{jk}D_{kk})$ を計算し, その後で U_{ij}, L_{ji} を求めるようにすれば演算量を減らすことができ, 全体としてパフォーマンスアップが可能である.

$$
\begin{aligned}
&\textbf{for} \quad j = 1 \\
&\qquad D_{11} = A_{11}
\end{aligned}
$$

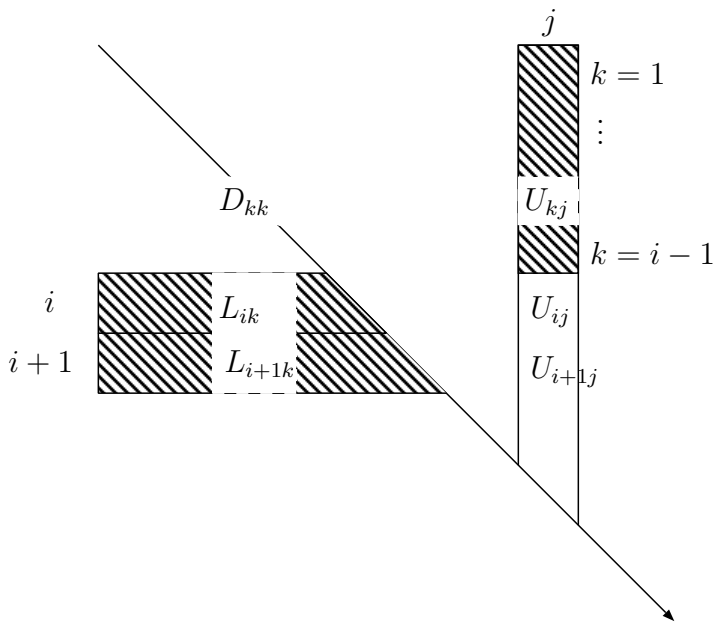


図 1.3: 三角分解の進み方

for $j = 2$

$U_{12} = A_{12}/D_{11}$

$L_{21} = A_{21}/D_{11}$

$D_{22} = A_{22} - L_{21}D_{11}U_{12}$

for $j = 3 \sim n$

for $i = 2 \sim j-1$

for $k = 1 \sim i-1$

$\texttt{temp\_}U = \texttt{temp\_}U + L_{ik}(D_{kk}U_{kj})$

$\texttt{temp\_}L = \texttt{temp\_}L + (L_{jk}D_{kk})U_{ki}$

end for

$(D_{ii}U_{ij}) = A_{ij} - \texttt{temp\_}U$

$(L_{ji}D_{ii}) = A_{ji} - \texttt{temp\_}L$

end for

for $i = 1 \sim j-1$

$U_{ij} = (D_{ii}U_{ij})/D_{ii}$

$L_{ji} = (L_{ji}D_{ii})/D_{ii}$

end for

for $k = 1 \sim j-1$

$\texttt{temp} = \texttt{temp} + L_{jk}D_{kk}U_{kj}$

end for

$D_{jj} = A_{jj} - \texttt{temp}$

end for

バージョン 3 として三角分解の演算量を軽減したコーディングをする. また 1000×1000 のマトリックスにしたときの cpu time を計測する. 用いる変数はバージョン 2.2 と同じとする. 三角分解中に用いる $(D_{kk}U_{kj})$, $(L_{jk}D_{kk})$ も U_{kj}, L_{jk} 同様, A の配列に上書きして記憶する.

用いる OS やコンパイラにもよるがバージョン 2.2 よりもかなり加速するはずである.

1.6.2 fortran コーディング例

fortran でのコーディング例を示すと以下のようになる. gauss_ver22 の三角分解の演算量を軽減したサブルーチンが gauss_ver3 である.

Makefile

```
FC = g77
F_OPT = -O2
```

```
OBJS = main.o misc.o gauss_ver3.o

TARGET = gauss_ver3

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
c gauss elimination ver. 3
c     by WATANABE Hiroshi (2000 May)
c
      implicit real*8(a-h,o-z)
      parameter(n = 1000)
      parameter(i_check = 1)
c
      dimension a(n,n),b(n),c(n)
c
      call init(a,b,c,n)
c
      if (i_check .ge. 2) call check_matrix(a,b,c,n)
c
      call gauss_ver3(a,c,n)
c
      if (i_check .ge. 1) call check_solution(c,n)
c
      stop
      end
c
```

misc.f　gauss_ver1 と共通

gauss_ver3.f

```
c######################################################################
      subroutine gauss_ver3(a,c,n)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),c(*)
c
```

```
      a(1,2) = a(1,2) / a(1,1)
      a(2,1) = a(2,1) / a(1,1)
c
      a(2,2) = a(2,2) - a(2,1) * a(1,1) * a(1,2)
c
      do j = 3,n
          do i = 2, j-1
              temp_u = 0.d0
              temp_l = 0.d0
              do k = 1, i-1
                  temp_u = temp_u + a(i,k) * a(k,j)
                  temp_l = temp_l + a(j,k) * a(k,i)
              end do
              a(i,j) = a(i,j) - temp_u
              a(j,i) = a(j,i) - temp_l
          end do
          do i = 1,j-1
              a(i,j) = a(i,j) / a(i,i)
              a(j,i) = a(j,i) / a(i,i)
          end do
          temp = 0.d0
          do k = 1,j-1
              temp = temp + a(j,k) * a(k,k) * a(k,j)
          end do
          a(j,j) = a(j,j) - temp
      end do
c
      do i = 2,n
          do j = 1,i-1
              c(i) = c(i) - a(i,j) * c(j)
          end do
      end do
c
      do i = 1,n
          c(i) = c(i) / a(i,i)
      end do
c
      do j = n,2,-1
          do i = 1,j-1
              c(i) = c(i) - a(i,j) * c(j)
          end do
      end do
c
      return
      end
```

c

1.6.3　C コーディング例

C でのコーディング例を示すと以下のようになる．gauss_ver22 の三角分解の演算量を軽減したサブルーチンが gauss_ver3 である．

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o misc.o gauss_ver3.o

TARGET = gauss_ver3

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
/* gauss elimination ver. 3
     by WATANABE Hiroshi (2000 July) */

#include <stdio.h>
#define N 1000
#define I_CHECK 1

void init(double *a, double *b, double *c, int n);
void check_matrix(double *a, double *b, double *c, int n);
void gauss_ver3(double *a, double *c, int n);
void check_solution(double *b, int n);

main()
{
    double *a,*b,*c;
    int n;
```

```
    n = N;

    a  = (double *)malloc(sizeof(double)*n*n);
    b  = (double *)malloc(sizeof(double)*n);
    c  = (double *)malloc(sizeof(double)*n);

    init(a,b,c,n);

    if (I_CHECK >= 2) check_matrix(a,b,c,n);

    gauss_ver3(a,c,n);

    if (I_CHECK >= 1) check_solution(b,n);
}
```

misc.c gauss_ver1 と共通

gauss_ver3.c

```
void gauss_ver3(double *arg_a, double *arg_c, int n)
{
    double *a, *c;
    int i, j, k;
    double temp_u, temp_l, temp;

    a  = arg_a -1-n;
    c = arg_c-1;

    a[1+2*n] = a[1+2*n] / a[1+1*n];
    a[2+1*n] = a[2+2*n] / a[1+1*n];

    a[2+2*n] = a[2+2*n] - a[2+1*n] * a[1+1*n] * a[1+2*n];

    for (j = 3; j <= n; ++j){
        for (i = 2; i <= j-1; ++i){
            temp_u = 0.0;
            temp_l = 0.0;
            for (k = 1; k <= i-1; ++k){
                temp_u = temp_u + a[i+k*n] * a[k+j*n];
                temp_l = temp_l + a[j+k*n] * a[k+i*n];
            }
            a[i+j*n] = a[i+j*n] - temp_u;
            a[j+i*n] = a[j+i*n] - temp_l;
        }
```

```
        for (i = 1; i <= j-1; ++i){
            a[i+j*n] = a[i+j*n] / a[i+i*n];
            a[j+i*n] = a[j+i*n] / a[i+i*n];
        }
        temp = 0.0;
        for (k = 1; k <= j-1; ++k){
            temp = temp + a[j+k*n] * a[k+k*n] * a[k+j*n];
        }
        a[j+j*n] = a[j+j*n] - temp;
    }

    for (i = 2; i <= n; ++i){
        temp = 0.0;
        for (j = 1; j <= i-1; ++j){
            temp = temp + a[i+j*n] * c[j];
        }
        c[i] = c[i] - temp;
    }

    for (i = 1; i <= n; ++i){
        c[i] = c[i] / a[i+i*n];
    }

    for (j = n; j >= 2; --j){
        for (i = 1; i <= j-1; ++i){
            c[i] = c[i] - a[i+j*n] * c[j];
        }
    }
}
```

1.7　Gauss の消去法のコーディング — 対称版 Gauss の消去法

1.7.1　対称マトリックスの三角分解

前述のように係数マトリックス $[A]$ が対称であれば $[A] = [L][D][L^T] = [U^T][D][U]$ の形式で三角分解できる. ここではバージョン 3 で示した三角分解の方法をもとにして, 上三角領域を使って三角分解を行う. 即ち $[U]$ を求める手続きを示す.

for $j = 1$

$D_{11} = A_{11}$

for $j = 2$

$U_{12} = A_{12}/D_{11}$

$D_{22} = A_{22} - U_{12}D_{11}U_{12}$

for $j = 3 \sim n$

for $i = 1 \sim j - 1$

for $k = 1 \sim i - 1$

$\texttt{temp} = \texttt{temp} + U_{ki}(D_{kk}U_{kj})$

end for

$D_{ii}U_{ij} = A_{ij} - \texttt{temp}$

end for

for $i = 1 \sim j - 1$

$U_{ij} = D_{ii}U_{ij}/D_{ii}$

end for

for $i = 1 \sim j - 1$

$\texttt{temp} = \texttt{temp} + U_{ij}D_{ii}U_{ij}$

end for

$D_{jj} = A_{jj} - \texttt{temp}$

end for

バージョン 4 として, 対称版 Gauss の消去法をコーディングする. $[U]$ を用いる場合と $[L]$ を用いる場合をコーディングし, それぞれ 1000×1000 のマトリックスにしたときの cpu time を計測する. 用いる変数はバージョン 3 と同じとする.

1.7.2　fortran コーディング例

fortran でのコーディング例を示すと以下のようになる．gauss_ver4l が $[L]$ を用いたもの, gauss_ver4u が $[U]$ を用いたものである．

演算量が全く同じにもかかわらず, cpu time がかなり異なるはずである．この理由はメモリーアクセススピードにある．ワークステーション, パソコンを問わず最近の計算機は普通のメモリーの他にキャッシュメモリーという非常に高速にアクセスできるメモリーをもち, CPU 内部で演算を並列に行うスーパースカラー機能をもっている．スーパースカラーを有効に活用するためには, CPU にデータを効率良く供給し続ける必要があり, そのためにキャッシュメモリーは用いられているがキャッシュメモリーの容量には上限があるので, 係数マトリックス全体を記憶することができない．そのためキャッシュメモリにロードされたデータをいかに効率良く使うかが問題になる．今回は, 速かった方がこれらの使いかたが上手であったと理解すればよい．

fortran の場合 2 次元配列は内部では 1 次元配列としてとりあつかわれる．$a(i,j)$ はメモリー上では, a の次元を n として $(j-1)\times n+i$ 番目に記憶されている．

$a(i+1,j)$ と $a(i,j+1)$ は 配列としては $a(i,j)$ の隣にあるがメモリー上では $a(i+1,j)$ は隣であるのに対して $a(i,j+1)$ は n 個はなれている．このため一般にマトリックスの計算では列を固定してループをまわした方がメモリー上で近い場所のデータしかアクセスしないので演算時間は短くなる．

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o misc.o gauss_ver4l.o gauss_ver4u.o

TARGET = gauss_ver4

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

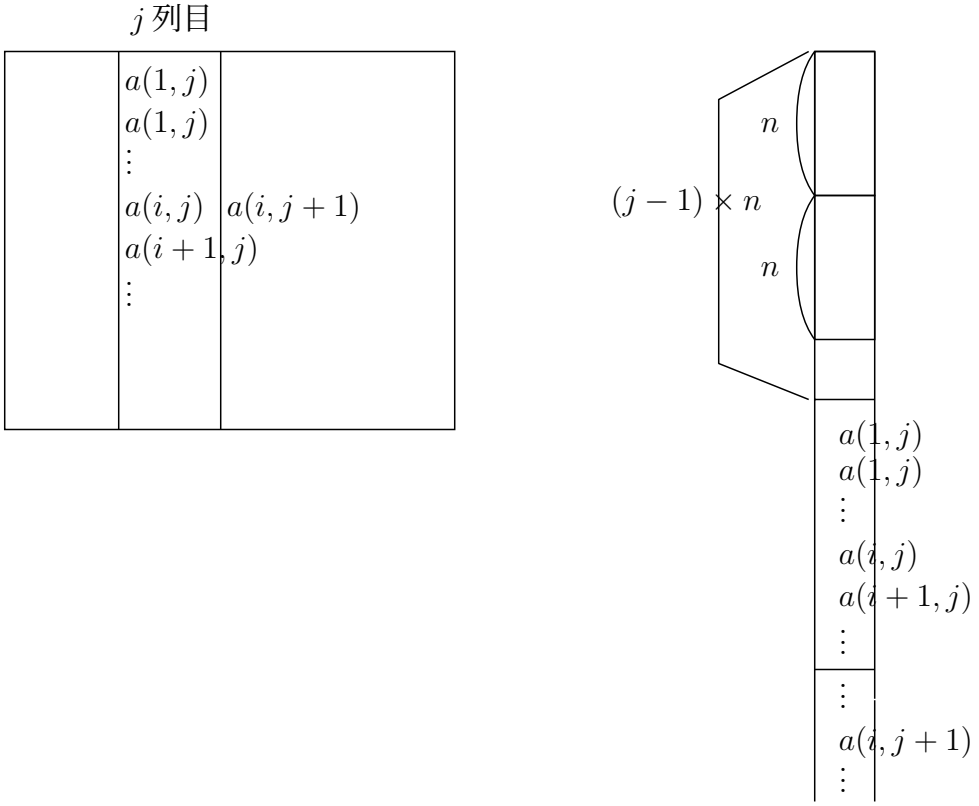


図 1.4: 配列とメモリー

main.f

```
c gauss elimination ver. 4l, 4u
c     by WATANABE Hiroshi (2000 June)
c
      implicit real*8(a-h,o-z)
      parameter(n = 1000)
      parameter(i_check = 1)
c
      dimension a(n,n),b(n),c(n)
c
      call init(a,b,c,n)
c
      if (i_check .ge. 2) call check_matrix(a,b,c,n)
c
c      call gauss_ver4l(a,c,n)
      call gauss_ver4u(a,c,n)
```

```
c
      if (i_check .ge. 1) call check_solution(c,n)
c
      stop
      end
c
```

misc.f　gauss_ver1 と共通

gauss_ver4l.f

```
c#########################################################################
      subroutine gauss_ver4l(a,c,n)
c#########################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),c(*)
c
      a(2,1) = a(2,1) / a(1,1)
c
      a(2,2) = a(2,2) - a(2,1) * a(1,1) * a(2,1)
c
      do j = 3,n
          do i = 2, j-1
              temp_l = 0.d0
              do k = 1, i-1
                  temp_l = temp_l + a(j,k) * a(i,k)
              end do
              a(j,i) = a(j,i) - temp_l
          end do
          do i = 1,j-1
              a(j,i) = a(j,i) / a(i,i)
          end do
          temp = 0.d0
          do k = 1,j-1
              temp = temp + a(j,k) * a(k,k) * a(j,k)
          end do
          a(j,j) = a(j,j) - temp
      end do
c
      do i = 2,n
          do j = 1,i-1
              c(i) = c(i) - a(i,j) * c(j)
          end do
      end do
```

```
c
      do i = 1,n
          c(i) = c(i) / a(i,i)
      end do
c
      do j = n,2,-1
          do i = 1,j-1
              c(i) = c(i) - a(j,i) * c(j)
          end do
      end do
c
      return
      end
c
```

gauss_ver4u.f

```
c#########################################################################
      subroutine gauss_ver4u(a,c,n)
c#########################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),c(*)
c
      a(1,2) = a(1,2) / a(1,1)
c
      a(2,2) = a(2,2) - a(1,2) * a(1,1) * a(1,2)
c
      do j = 3,n
          do i = 2, j-1
              temp_u = 0.d0
              do k = 1, i-1
                  temp_u = temp_u + a(k,i) * a(k,j)
              end do
              a(i,j) = a(i,j) - temp_u
          end do
          do i = 1,j-1
              a(i,j) = a(i,j) / a(i,i)
          end do
          temp = 0.d0
          do k = 1,j-1
              temp = temp + a(k,j) * a(k,k) * a(k,j)
          end do
          a(j,j) = a(j,j) - temp
      end do
c
```

```
      do i = 2,n
          do j = 1,i-1
              c(i) = c(i) - a(j,i) * c(j)
          end do
      end do
c
      do i = 1,n
          c(i) = c(i) / a(i,i)
      end do
c
      do j = n,2,-1
          do i = 1,j-1
              c(i) = c(i) - a(i,j) * c(j)
          end do
      end do
c
      return
      end
c
```

1.7.3　C コーディング例

fortran でのコーディング例を示すと以下のようになる. gauss_ver4l が $[L]$ を用いたもの, gauss_ver4u が $[U]$ を用いたものである.

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o misc.o gauss_ver4l.o gauss_ver4u.o

TARGET = gauss_ver4

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
/* gauss elimination ver. 4l, 4u
     by WATANABE Hiroshi (2000 July) */

#include <stdio.h>
#define N 1000
#define I_CHECK 1

void init(double *a, double *b, double *c, int n);
void check_matrix(double *a, double *b, double *c, int n);
void gauss_ver4l(double *a, double *c, int n);
void gauss_ver4u(double *a, double *c, int n);
void check_solution(double *b, int n);

main()
{
    double *a,*b,*c;
    int n;

    n = N;

    a  = (double *)malloc(sizeof(double)*n*n);
    b  = (double *)malloc(sizeof(double)*n);
    c  = (double *)malloc(sizeof(double)*n);

    init(a,b,c,n);

    if (I_CHECK >= 2) check_matrix(a,b,c,n);

/*    gauss_ver4l(a,c,n); */
    gauss_ver4u(a,c,n);

    if (I_CHECK >= 1) check_solution(b,n);
}
```

misc.c gauss_ver1 と共通

gauss_ver4l.c

```
void gauss_ver4l(double *arg_a, double *arg_c, int n)
{
    double *a, *c;
    int i, j, k;
    double temp_u, temp_l, temp;
```

```
    a  = arg_a -1-n;
    c = arg_c-1;

    a[2+1*n] = a[2+2*n] / a[1+1*n];

    a[2+2*n] = a[2+2*n] - a[2+1*n] * a[1+1*n] * a[2+1*n];

    for (j = 3; j <= n; ++j){
        for (i = 2; i <= j-1; ++i){
            temp_l = 0.0;
            for (k = 1; k <= i-1; ++k){
                temp_l = temp_l + a[j+k*n] * a[i+k*n];
            }
            a[j+i*n] = a[j+i*n] - temp_l;
        }
        for (i = 1; i <= j-1; ++i){
            a[j+i*n] = a[j+i*n] / a[i+i*n];
        }
        temp = 0.0;
        for (k = 1; k <= j-1; ++k){
            temp = temp + a[j+k*n] * a[k+k*n] * a[j+k*n];
        }
        a[j+j*n] = a[j+j*n] - temp;
    }

    for (i = 2; i <= n; ++i){
        temp = 0.0;
        for (j = 1; j <= i-1; ++j){
            temp = temp + a[i+j*n] * c[j];
        }
        c[i] = c[i] - temp;
    }

    for (i = 1; i <= n; ++i){
        c[i] = c[i] / a[i+i*n];
    }

    for (j = n; j >= 2; --j){
        for (i = 1; i <= j-1; ++i){
            c[i] = c[i] - a[j+i*n] * c[j];
        }
    }
}
```

gauss_ver4u.c

```
void gauss_ver4u(double *arg_a, double *arg_c, int n)
{
    double *a, *c;
    int i, j, k;
    double temp_u, temp_l, temp;

    a = arg_a-1-n;
    c = arg_c-1;

    a[1+2*n] = a[1+2*n] / a[1+1*n];

    a[2+2*n] = a[2+2*n] - a[1+2*n] * a[1+1*n] * a[1+2*n];

    for (j = 3; j <= n; ++j){
        for (i = 2; i <= j-1; ++i){
            temp_u = 0.0;
            temp_l = 0.0;
            for (k = 1; k <= i-1; ++k){
                temp_u = temp_u + a[k+i*n] * a[k+j*n];
            }
            a[i+j*n] = a[i+j*n] - temp_u;
        }
        for (i = 1; i <= j-1; ++i){
            a[i+j*n] = a[i+j*n] / a[i+i*n];
        }
        temp = 0.0;
        for (k = 1; k <= j-1; ++k){
            temp = temp + a[k+j*n] * a[k+k*n] * a[k+j*n];
        }
        a[j+j*n] = a[j+j*n] - temp;
    }

    for (i = 2; i <= n; ++i){
        temp = 0.0;
        for (j = 1; j <= i-1; ++j){
            temp = temp + a[j+i*n] * c[j];
        }
        c[i] = c[i] - temp;
    }

    for (i = 1; i <= n; ++i){
        c[i] = c[i] / a[i+i*n];
    }
```

```
    for (j = n; j >= 2; --j){
        for (i = 1; i <= j-1; ++i){
            c[i] = c[i] - a[i+j*n] * c[j];
        }
    }
}
```

1.8　境界条件処理

1.8.1　剛性方程式の特徴

連立一次方程式の例題として, 力学的に意味のあるものをとりあげる. 下図に示すように, x_1 軸に並行した $n+1$ 個の質点 (各質点の質量 $m_i(i=1\sim n+1)$) を n 個のばね (各ばねのばね定数 $k_i(i=1\sim n)$) で連結した系を考える. それぞれの質点は x_i 方向にしか運動しないように拘束されているとし, 各質点に外力 $f_i(i=1\sim n+1)$ が作用するとき, 各質点の運動方程式は以下のようになる.

$$
\begin{array}{lllll}
m_1\ddot{u}_1 & = f_1 & & & + k_1(u_2-u_1) \\
m_2\ddot{u}_2 & = f_2 & + k_1(u_1-u_2) & & + k_2(u_3-u_2) \\
m_3\ddot{u}_3 & = f_3 & + k_2(u_2-u_3) & & + k_3(u_4-u_3) \\
 & \vdots & & & \\
m_n\ddot{u}_n & = f_n & + k_{n-1}(u_{n-1}-u_n) & & + k_n(u_{n+1}-u_n) \\
m_{n+1}\ddot{u}_{n+1} & = f_{n+1} & + k_n(u_n-u_{n+1}) & &
\end{array}
$$

ただし $u_i(i=1\sim n+1)$ は質点 m_i の x_1 方向変位とする.

これをマトリックスで表すと次のようになる.

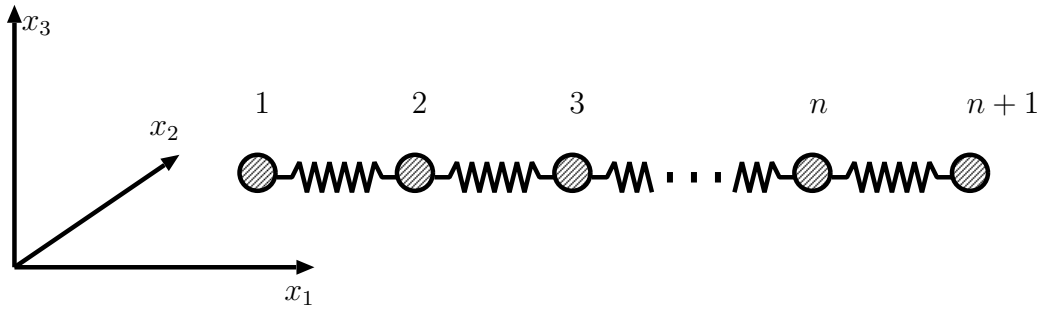


図 1.5: ばね - 質点系

$$\begin{bmatrix} k_1 & -k_1 & & & & & \\ -k_1 & k_1+k_2 & -k_2 & & & & \\ & -k_2 & k_2+k_3 & -k_3 & & & \\ & & & \ddots & & & \\ & & & -k_{n-1} & k_{n-1}+k_n & -k_n & \\ & & & & -k_n & k_{n+1} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \\ u_{n+1} \end{Bmatrix} = \begin{Bmatrix} f_1 - m_1\ddot{u}_1 \\ f_2 - m_2\ddot{u}_2 \\ f_3 - m_3\ddot{u}_3 \\ \vdots \\ f_n - m_n\ddot{u}_n \\ f_{n+1} - m_{n+1}\ddot{u}_{n+1} \end{Bmatrix} \tag{1.85}$$

ここで, $\ddot{u}_1 = \ddot{u}_2 = \cdots = \ddot{u}_n = \ddot{u}_{n+1} = 0, \dot{u}_1 = \dot{u}_2 = \cdots = \dot{u}_n = \dot{u}_{n+1} = 0$ の場合を考える. このとき, 物理的には力のつりあい状態は保たれるが, 系全体がどこでつりあうかは決まらない. また, 係数マトリックスも特異で rank は n である. (後述の 1.8.2 参照)

通常の問題設定では, 例えば $u_1 = 0$ など, 何らかの条件を追加し, 求解する. あるいは, $v_1 = u_2 - u_1, v_2 = u_3 - u_2, \cdots v_n = u_{n+1} - u_n$ と新たに変数を設定し直して求解するのが一般的である. 詳細は後述するが, 一般に静的な問題の有限要素解析で用いる剛性マトリクスは, 力のつりあい方程式を離散化して得られるもので, 系全体がどこでつりあうのかということは式には反映されない. 即ち, 剛性マトリックスは特異であり, $u_1 = 0$ などのように変位に条件を加えないと解くことができない. また, 値自体は未知であるが, ある適当値の仮想外力を作用させて, $u_j = a$ など, ある箇所の変位を任意の目標値にするような問題設定を行う場合もある. このときには u_j が既知でありかつ非零で, また f_j は未知である. 有限要素解析では, このように本来は未知量である変位の一部を既知量として取り扱うケースを一般に境界条件処理, 特に既知量が 0 でないようなケースを強制変位と呼ぶ.

以下のような連立一次方程式を考える.

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & & & \vdots \\ \vdots & & & \vdots \\ A_{n1} & \dots & \dots & A_{nn} \end{bmatrix} \begin{Bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{Bmatrix} = \begin{Bmatrix} C_1 \\ C_2 \\ \vdots \\ C_n \end{Bmatrix} \tag{1.86}$$

ここで, C_j 以外の右辺と b_j が既知であるとする. マトリックス形式を通常の連立一次方程式の形式に書き戻すと以下のようになる.

$$\begin{cases} A_{11} + A_{12}b_2 + \cdots + A_{1j}b_j + \cdots + A_{1n}b_n &= C_1 \\ A_{21} + A_{22}b_2 + \cdots + A_{2j}b_j + \cdots + A_{2n}b_n &= C_2 \\ \qquad\qquad \vdots & \vdots \\ A_{j1} + A_{j2}b_2 + \cdots + A_{jj}b_j + \cdots + A_{jn}b_n &= C_j \\ \qquad\qquad \vdots & \vdots \\ A_{n1} + A_{n2}b_2 + \cdots + A_{nj}b_j + \cdots + A_{nn}b_n &= C_n \end{cases} \tag{1.87}$$

この中で j 行目の式だけは右辺の C_j が未知量であり, C_j は通常の未知量 $b_i(i = 1 \sim j-1, j+1 \sim n)$ がすべて求められた後に,

$$C_j = A_{j1}b_1 + A_{j2}b_2 + \cdots + Ajjb_j + \cdots + A_{jn}b_n \tag{1.88}$$

により求められる. そのため他の行の式とは性質が異なるので, $b_i(i = 1 \sim j-1, j+1 \sim n)$ を求めるプロセスからは除外する. 即ち j 行目を抜いた $n-1$ 個の式を考える.

$$\begin{cases} A_{11} + A_{12}b_2 + \cdots + A_{1j}b_j + \cdots + A_{1n}b_n = C_1 \\ A_{21} + A_{22}b_2 + \cdots + A_{2j}b_j + \cdots + A_{2n}b_n = C_2 \\ \qquad\qquad\qquad\qquad \vdots \qquad\qquad\qquad\qquad \vdots \\ A_{j-1\,1} + A_{j-1\,2}b_2 + \cdots + A_{j-1\,j}b_j + \cdots + A_{j-1\,n}b_n = C_{j-1} \\ A_{j+1\,1} + A_{j+1\,2}b_2 + \cdots + A_{j+1\,j}b_j + \cdots + A_{j-1\,n}b_n = C_{j+1} \\ \qquad\qquad\qquad\qquad \vdots \qquad\qquad\qquad\qquad \vdots \\ A_{n1} + A_{n2}b_2 + \cdots + A_{nj}b_j + \cdots + A_{nn}b_n = C_n \end{cases} \tag{1.89}$$

この式中の $A_{1j}b_j, A_{2j}b_j, \cdots, A_{nj}b_j$ は既知量なので右辺に移項する.

$$\left\{\begin{array}{ccccccc}
A_{11}+A_{12}b_2 & +\cdots+ & A_{1\,j-1}b_{j-1}+A_{1\,j+1}b_{j+1} & +\cdots+ & A_{1n}b_n & = & C_1-A_{1j}b_j \\
A_{21}+A_{22}b_2 & +\cdots+ & A_{2\,j-1}b_{j-1}+A_{2\,j+1}b_{j+1} & +\cdots+ & A_{2n}b_n & = & C_2-A_{2j}b_j \\
 & & \vdots & & & & \vdots \\
A_{j-1\,1}+A_{j-1\,2}b_2 & +\cdots+ & A_{j-1\,j-1}b_{j-1}+A_{j-1\,j+1}b_{j+1} & +\cdots+ & A_{j-1\,n}b_n & = & C_{j-1}-A_{j-1\,j}b_j \\
A_{j+1\,1}+A_{j+1\,2}b_2 & +\cdots+ & A_{j+1\,j-1}b_{j-1}+A_{j+1\,j+1}b_{j+1} & +\cdots+ & A_{j+1\,n}b_n & = & C_{j+1}-A_{j+1\,j}b_j \\
 & & \vdots & & & & \vdots \\
A_{n1}+A_{n2}b_2 & +\cdots+ & A_{n\,j-1}b_{j-1}+A_{n\,j+1}b_{j+1} & +\cdots+ & A_{nn}b_n & = & C_n-A_{nj}b_j
\end{array}\right. \tag{1.90}$$

これをあらためてマトリックス表示すると, 以下のようになる. 即ち係数マトリックスは, j 行と j 列をとり除いたものに, 未知ベクトルは j 行目をとり除いたものに, 右辺は, もとの左辺からもとの係数マトリックスの j 列目の b_j 倍をひいたものの j 行目をとり除いたものに置き換えられる.

$$\begin{bmatrix}
A_{11} & \dots & A_{1\,j-1} & A_{1\,j+1} & \dots & A_{1n} \\
A_{21} & \dots & A_{2\,j-1} & A_{2\,j+1} & \dots & A_{2n} \\
\vdots & & & & & \\
A_{j-1\,1} & \dots & A_{j-1\,j-1} & A_{j+1\,j+1} & \dots & A_{j-1\,n} \\
A_{j+1\,1} & \dots & A_{j+1\,j+1} & A_{j+1\,j+1} & \dots & A_{j+1\,n} \\
\vdots & & & & & \\
A_{n1} & \dots & A_{n\,j-1} & A_{n\,j+1} & \dots & A_{nn}
\end{bmatrix}
\begin{Bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{j-1} \\ b_{j+1} \\ \vdots \\ b_n \end{Bmatrix}
=
\begin{Bmatrix} C_1 \\ C_2 \\ \vdots \\ C_{j-1} \\ C_{j+1} \\ \vdots \\ C_n \end{Bmatrix}
- b_j
\begin{Bmatrix} A_{1j} \\ A_{2j} \\ \vdots \\ A_{j-1\,j} \\ A_{j+1\,j} \\ \vdots \\ A_{nj} \end{Bmatrix} \tag{1.91}$$

b_j が 0 であれば右辺第 2 項は消滅する.

以上の操作は, 既知の b_j が複数になっても同様である. $b_{j(1)}, b_{j(2)}, \cdots, b_{j(m)}$ が既知であるとする. まず連立一次方程式から $j(1), j(2), \cdots, j(m)$ 行目の方程式を除外する. 次に左辺の $b_{j(1)}, b_{j(2)}, \cdots, b_{j(m)}$ を含んだ項を右辺に移項する. これをマトリックス表示すればよい.

1.8.2　ばね - 質点系つりあい方程式の係数マトリックスの rank

簡単のため, $n = 2$ の場合を考える. 基本行列 $[P(i, j; C)], [R(i; C)]$ を以下のように定義する.

$$[P(i,j;C)] = \begin{array}{c} \\ i \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & & & j & & \end{array} \\ \begin{bmatrix} 1 & & & & \vdots & & \\ & \ddots & & & \vdots & & \\ \cdots & \cdots & 1 & \cdots & C & & \\ & & & \ddots & \vdots & & \\ & \mathbf{0} & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} \end{array} \tag{1.92}$$

$$[R(i;C)] = \begin{array}{c} \\ i \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & & & i & & \end{array} \\ \begin{bmatrix} 1 & & & & \vdots & & \\ & \ddots & & & \vdots & \mathbf{0} & \\ & & & 1 & \vdots & & \\ \cdots & \cdots & \cdots & C & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & \mathbf{0} & & & & & 1 \end{bmatrix} \end{array} \tag{1.93}$$

(1.94)

基本行列を用いて剛性マトリックスを以下のように変形する.

$$[K] = \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \tag{1.95}$$

$$[P(2,1;1)]\underline{[K]} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \underline{\begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1+k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix}} = \begin{bmatrix} k_1 & -k_1 & 0 \\ 0 & k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \tag{1.96}$$

$$[P(3,2;1)]\underline{[P(2,1;1)][K]} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \underline{\begin{bmatrix} k_1 & -k_1 & 0 \\ 0 & k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix}} = \begin{bmatrix} k_1 & -k_1 & 0 \\ 0 & k_2 & -k_2 \\ 0 & 0 & 0 \end{bmatrix} \tag{1.97}$$

$$\underline{[P(3,2;1)][P(2,1;1)][K]}[P(1,2;1)] = \underline{\begin{bmatrix} k_1 & -k_1 & 0 \\ 0 & k_2 & -k_2 \\ 0 & 0 & 0 \end{bmatrix}} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & -k_2 \\ 0 & 0 & 0 \end{bmatrix} \tag{1.98}$$

$$\underline{[P(3,2;1)][P(2,1;1)][K][P(1,2;1)]}[P(2,3;1)] = \underline{\begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & -k_2 \\ 0 & 0 & 0 \end{bmatrix}} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{1.99}$$

$$\begin{aligned} &[R(2;\frac{1}{k_2})][R(1;\frac{1}{k_1})]\underline{[P(3,2;1)][P(2,1;1)][K][P(1,2;1)][P(2,3;1)]} \\ &\qquad = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{k_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{k_1} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{1.100}$$

したがって $[K]$ の rank は 2 である. この操作を $(n+1)\times(n+1)$ 次元の係数マトリックス $[K]$ に対して適用すると, 以下のように $[K]$ の rank が n であることがわかる.

$$[R(n;\frac{1}{k_n})]\cdots[R(1;\frac{1}{k_1})][P(n+1,n;1)]\cdots[P(i+1,i;1)]$$
$$\cdots[P(2,1;1)][K][P(1,2;1)]\cdots[P(j,j+1;1)]\cdots[P(n,n+1;1)]$$
$$= \underbrace{\left[\begin{array}{cccc} 1 & & & \\ & 1 & & 0 \\ & & \ddots & \\ & 0 & & 1 \\ & & & & 0 \end{array}\right]}_{n} \left.\begin{array}{c}\\\\\\\\\\\end{array}\right\} n \qquad (1.101)$$

1.8.3　境界条件処理のコーディング

以上で説明した方法を実際にコーディングするにあたってもっとも大きな問題は、縮退した係数マトリックスをいかにつくるかである。たとえば、二つの配列を用いて本当に縮退するとすると、二つの配列がある程度の次元に抑えられていればよいが、問題が大きい場合は計算機のメモリーからはみ出してしまう可能性がある。また後述する skyline 法では係数マトリックスを 1 次元配列に記憶するので簡単に縮退することができない。

ここでは有限要素解析コードでよく用いられている方法のうちでもっとも簡単な方法を紹介する。

未知数のうち $b_{j(1)},\cdots,b_{j(m)}$ が既知とし、それぞれの値を $\tilde{b}_{j(1)},\cdots,\tilde{b}_{j(m)}$ とする。まず最初にすべの方程式の右辺に左辺の既知項を移項する。次に連立一次方程式から除外される $j(1),\cdots,j(m)$ 行の方程式についてはそれぞれ $b_{j(1)}=\tilde{b}_{j(1)},\cdots,b_{j(m)}=\tilde{b}_{j(m)}$ に置き換える。これをマトリックス表示すると $j(1),\cdots,j(m)$ 行、$j(1),\cdots,j(m)$ 列は対角項を除き 0 対角項は 1 その他の部分は元の係数マトリックスと同じという新しい係数マトリックスが得られ、右辺については、 $j(1),\cdots,j(m)$ 列については、それぞれ $\tilde{b}_{j(1)},\cdots,\tilde{b}_{j(m)}$ に置き換えその他の部分についてはもとの右辺に左辺の既知項を移項したものになる。

例として係数マトリックスの次元を $6, m=3, j(1)=1, j(2)=3, j(3)=6$ とすると、以

下のようになる。

元の方程式：

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} & A_{16} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} & A_{26} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & A_{36} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} & A_{46} \\ A_{51} & A_{52} & A_{53} & A_{54} & A_{55} & A_{56} \\ A_{61} & A_{62} & A_{63} & A_{64} & A_{65} & A_{66} \end{bmatrix} \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{Bmatrix} = \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{Bmatrix} \tag{1.102}$$

変更後の方程式：

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & A_{22} & 0 & A_{24} & A_{25} & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & A_{42} & 0 & A_{44} & A_{45} & 0 \\ 0 & A_{52} & 0 & A_{54} & A_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{Bmatrix} = \begin{Bmatrix} \tilde{b}_1 \\ c_2 \\ \tilde{b}_3 \\ c_4 \\ c_5 \\ \tilde{b}_6 \end{Bmatrix} - \tilde{b}_1 \begin{Bmatrix} 0 \\ A_{21} \\ 0 \\ A_{41} \\ A_{51} \\ 0 \end{Bmatrix} - \tilde{b}_3 \begin{Bmatrix} 0 \\ A_{23} \\ 0 \\ A_{43} \\ A_{53} \\ 0 \end{Bmatrix} - \tilde{b}_6 \begin{Bmatrix} 0 \\ A_{26} \\ 0 \\ A_{46} \\ A_{56} \\ 0 \end{Bmatrix} \tag{1.103}$$

これを実際にコーディングするようにインデックスで書けば以下のようになる.

for $i = 1 \sim m$

　for $k = 1 \sim n$

　　$c_k = c_k - \tilde{b}_{j(i)} A_{kj(i)}$

　end for

end for

for $i = 1 \sim m$

　$c_{j(i)} = \tilde{b}_{j(i)}$

end for

for $i = 1 \sim m$

　for $k = 1 \sim n$

　　$A_{kj(i)} = 0$

　　$A_{j(i)k} = 0$

　end for

　$A_{j(i)j(i)} = 1$

　$c_{j(i)} = 0$

end for

コーディングに用いる変数は以下のようにする。

定式化	変数名
既知量の数 m	`n_bc_given`
既知量のインデックス $j(m)$	`i_bc_given(n_bc_given)`
既知量 $\tilde{b}_{j(m)}$	`v_bc_given(n_bc_given)`
係数マトリックスの次元 n	`n`
係数マトリックス A_{nn}	`a(n,n)`
未知ベクトル b_n	`b(n)`
左辺 c_n	`c(n)`

課題

前述のバネ–質点系の剛性方程式をつくり境界条件処理を行いいくつかの外力強制変位に対して解を求める。用いる Gauss の消去法のバージョンは 4 とする。簡単のため $k_1 = k_2 = \cdots = k_m = k$ とする

例 1	$n = 4$	$k = 1$	$u_1 = 0$	$f_5 = 1$
例 2	$n = 4$	$k = 1$	$u_1 = u_5 = 0$	$f_3 = 1$
例 3	$n = 4$	$k = 1$	$u_1 = u_5 = 0$	$u_3 = 1$

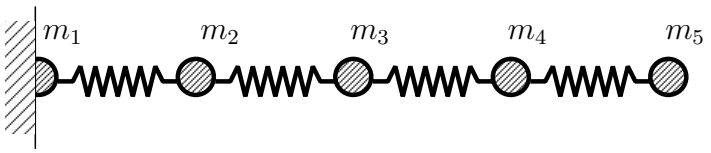


図 1.6: ばね - 質点系

1.8.4 fortran コーディング例

fortran でのコーディング例を示すと以下のようになる。

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o stiff.o ex11-13.o gauss_ver4u.o bound1.o misc2.o

TARGET = bound1

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
c bound operation ver.1 with gauss elimination ver. 4u
c     by WATANABE Hiroshi (2000 June)
c
      implicit real*8(a-h,o-z)
      parameter(n = 5)
      parameter(i_check = 3)
      parameter(i_type = 3)
c
      dimension a(n,n),b(n),c(n)
      dimension i_bc_given(n),v_bc_given(n)
c
      nk = 4
      ak = 1.D0
c
      call stiff(a,n,nk,ak)
      if (i_check .ge. 3) call check_stiff(a,n)
c
      if (i_type .eq. 1) call ex11(c,n_bc_given,i_bc_given,v_bc_given,n)
      if (i_type .eq. 2) call ex12(c,n_bc_given,i_bc_given,v_bc_given,n)
      if (i_type .eq. 3) call ex13(c,n_bc_given,i_bc_given,v_bc_given,n)
c
      call bound1(a,c,n,n_bc_given,i_bc_given,v_bc_given)
      if (i_check .ge. 2) call check_matrix2(a,c,n)
c
      call gauss_ver4u(a,c,n)
      if (i_check .ge. 1) call check_solution(c,n)
c
      stop
      end
c
```

stiff.f

```
c######################################################################
      subroutine stiff(a,n,nk,ak)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*)
c
      do i = 1,n
          do j = 1,n
              a(j,i) = 0.D0
          end do
      end do
c
      do i = 1,nk
          a(i  ,i  ) = a(i  ,i  ) + ak
          a(i  ,i+1) = a(i  ,i+1) - ak
          a(i+1,i  ) = a(i+1,i  ) - ak
          a(i+1,i+1) = a(i+1,i+1) + ak
      end do
c
      return
      end
c
```

ex11-13.f

```
c######################################################################
      subroutine ex11(c,n_bc_given,i_bc_given,v_bc_given,n)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension c(*),i_bc_given(*),v_bc_given(*)
c
      do i = 1,n
          c(i) = 0.D0
      end do
      c(n) = 1.D0
c
      n_bc_given    = 1
      i_bc_given(1) = 1
      v_bc_given(1) = 0.D0
c
      return
      end
c
```

```
c######################################################################
      subroutine ex12(c,n_bc_given,i_bc_given,v_bc_given,n)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension c(*),i_bc_given(*),v_bc_given(*)
c
      do i = 1,n
          c(i) = 0.D0
      end do
      c(3) = 1.D0
c
      n_bc_given    = 2
      i_bc_given(1) = 1
      i_bc_given(2) = 5
      v_bc_given(1) = 0.D0
      v_bc_given(2) = 0.D0
c
      return
      end
c
c######################################################################
      subroutine ex13(c,n_bc_given,i_bc_given,v_bc_given,n)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension c(*),i_bc_given(*),v_bc_given(*)
c
      do i = 1,n
          c(i) = 0.D0
      end do
c
      n_bc_given    = 3
      i_bc_given(1) = 1
      i_bc_given(2) = 3
      i_bc_given(3) = 5
      v_bc_given(1) = 0.D0
      v_bc_given(2) = 1.D0
      v_bc_given(3) = 0.D0
c
      return
      end
c
```

gauss_ver4u.f　gauss_ver4 と共通

bound1.f

```
c######################################################################
      subroutine bound1(a,c,n,n_bc_given,i_bc_given,v_bc_given)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),c(*),i_bc_given(*),v_bc_given(*)
c
      do i = 1,n_bc_given
          do k = 1,n
              c(k) = c(k) - v_bc_given(i) * a(k,i_bc_given(i))
          end do
      end do
c
      do i = 1,n_bc_given
          c(i_bc_given(i)) = v_bc_given(i)
      end do
c
      do i = 1,n_bc_given
          do k = 1,n
              a(k,i_bc_given(i)) = 0.D0
              a(i_bc_given(i),k) = 0.D0
          end do
          a(i_bc_given(i),i_bc_given(i)) = 1.D0
      end do
c
      return
      end
c
```

misc2.f

```
c######################################################################
      subroutine check_stiff(a,n)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*)
c
      write(*,*) 'matrix A'
      do i = 1,n
          write(*,1000) (a(i,j),j=1,n)
      end do
c
 1000 format(8e10.3)
```

```
      return
      end
c
c#########################################################################
      subroutine check_matrix2(a,c,n)
c#########################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),c(*)
c
      write(*,*) 'matrix A'
      do i = 1,n
          write(*,1000) (a(i,j),j=1,n)
      end do
      write(*,*) 'vector c'
      write(*,1000) (c(i),i=1,n)
c
 1000 format(8e10.3)

      return
      end
c
c#########################################################################
      subroutine check_solution(b,n)
c#########################################################################
      implicit real*8(a-h,o-z)
      dimension b(*)
c
      write(*,*) 'solution vector b'
      write(*,1000) (b(i),i=1,n)
c
 1000 format(8e10.3)
c
      return
      end
c
```

1.8.5　C コーディング例

C でのコーディング例を示すと以下のようになる。

Makefile

```
CC = gcc
```

```
C_OPT = -O2

OBJS = main.o stiff.o ex11-13.o gauss_ver4u.o bound1.o misc2.o

TARGET = bound1

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
/* bound operation ver.1 with gauss elimination ver. 4u
     by WATANABE Hiroshi (2000 July) */

#include <stdio.h>
#define N 5
#define I_CHECK 3
#define I_TYPE 3

void stiff(double *a, int n, int nk, double ak);
void check_stiff(double *a, int n);
void ex11(double *c, int *n_bc_given, int *i_bc_given,
          double *v_bc_given, int n);
void ex12(double *c, int *n_bc_given, int *i_bc_given,
          double *v_bc_given, int n);
void ex13(double *c, int *n_bc_given, int *i_bc_given,
          double *v_bc_given, int n);
void bound1(double *a, double *c, int n, int n_bc_given,
            int *i_bc_given, double *v_bc_given);
void check_matrix2(double *a, double *c, int n);
void gauss_ver4u(double *a, double *c, int n);
void check_solution(double *b, int n);

main()
{
    double *a,*b,*c,*v_bc_given;
    int *i_bc_given;
    double ak;
    int n,nk,n_bc_given;

    n = N;
```

```
    nk = 4;
    ak = 1.0;

    a  = (double *)malloc(sizeof(double)*n*n);
    b  = (double *)malloc(sizeof(double)*n);
    c  = (double *)malloc(sizeof(double)*n);
    v_bc_given = (double *)malloc(sizeof(double)*n);
    i_bc_given = (int *)malloc(sizeof(int)*n);

    stiff(a,n,nk,ak);
    if (I_CHECK >= 3) check_stiff(a,n);

    if (I_TYPE == 1) ex11(c,&n_bc_given,i_bc_given,v_bc_given,n);
    if (I_TYPE == 2) ex12(c,&n_bc_given,i_bc_given,v_bc_given,n);
    if (I_TYPE == 3) ex13(c,&n_bc_given,i_bc_given,v_bc_given,n);

    bound1(a,c,n,n_bc_given,i_bc_given,v_bc_given);
    if (I_CHECK >= 2) check_matrix2(a,c,n);

    gauss_ver4u(a,c,n);
    if (I_CHECK >= 1) check_solution(c,n);
}
```

stiff.c

```
void stiff(double *arg_a, int n, int nk, double ak)
{
    double *a;
    int i,j;

    a = arg_a -1-n;

    for (i = 1; i <= n; ++i){
        for (j = 1; j <= n; ++j){
            a[i+j*n] = 0.0;
        }
    }

    for (i = 1; i <= nk ; ++i){
        a[ i   + i   *n] = a[ i   + i   *n] + ak;
        a[ i   +(i+1)*n] = a[ i   +(i+1)*n] - ak;
        a[(i+1)+ i   *n] = a[(i+1)+ i   *n] - ak;
        a[(i+1)+(i+1)*n] = a[(i+1)+(i+1)*n] + ak;
    }
}
```

ex11-13.c

```
void ex11(double *arg_c, int *n_bc_given, int *arg_i_bc_given,
          double *arg_v_bc_given ,int n)
{
    double *c, *v_bc_given;
    int *i_bc_given;
    int i;

    c = arg_c -1;
    v_bc_given = arg_v_bc_given -1;
    i_bc_given = arg_i_bc_given -1;

    for (i = 1; i <= n; ++i){
        c[i] = 0.0;
    }
    c[n] = 1.0;

    *n_bc_given   = 1;
    i_bc_given[1] = 1;
    v_bc_given[1] = 0.0;
}

void ex12(double *arg_c, int *n_bc_given, int *arg_i_bc_given,
          double *arg_v_bc_given ,int n)
{
    double *c, *v_bc_given;
    int *i_bc_given;
    int i;

    c = arg_c -1;
    v_bc_given = arg_v_bc_given -1;
    i_bc_given = arg_i_bc_given -1;

    for (i = 1; i <= n; ++i){
        c[i] = 0.0;
    }
    c[3] = 1.0;

    *n_bc_given   = 2;
    i_bc_given[1] = 1;
    i_bc_given[2] = 5;
    v_bc_given[1] = 0.0;
    v_bc_given[2] = 0.0;
}
```

```
void ex13(double *arg_c, int *n_bc_given, int *arg_i_bc_given,
          double *arg_v_bc_given ,int n)
{
    double *c, *v_bc_given;
    int *i_bc_given;
    int i;

    c = arg_c -1;
    v_bc_given = arg_v_bc_given -1;
    i_bc_given = arg_i_bc_given -1;

    for (i = 1; i <= n; ++i){
        c[i] = 0.0;
    }

    *n_bc_given   = 3;
    i_bc_given[1] = 1;
    i_bc_given[2] = 3;
    i_bc_given[3] = 5;
    v_bc_given[1] = 0.0;
    v_bc_given[2] = 1.0;
    v_bc_given[3] = 0.0;
}
```

gauss_ver4u.c gauss_ver4 と共通

bound1.c

```
void bound1(double *arg_a, double *arg_c, int n, int n_bc_given,
            int *arg_i_bc_given, double *arg_v_bc_given)
{
    double *a,*c,*v_bc_given;
    int *i_bc_given,i,k;

    a = arg_a -1-n;
    c = arg_c -1;
    v_bc_given = arg_v_bc_given -1;
    i_bc_given = arg_i_bc_given -1;

    for (i = 1; i <= n_bc_given; ++i){
        for (k = 1; k <= n; ++k){
            c[k] = c[k] - v_bc_given[i] * a[k+i_bc_given[i]*n];
        }
    }
```

```
    for (i = 1; i <= n_bc_given; ++i){
        c[i_bc_given[i]] = v_bc_given[i];
    }

    for (i = 1; i <= n_bc_given; ++i){
        for (k = 1; k <= n; ++k){
    a[k+i_bc_given[i]*n] = 0.0;
    a[i_bc_given[i]+k*n] = 0.0;
        }
        a[i_bc_given[i]+i_bc_given[i]*n]= 1.0;
    }
}
```

misc2.c

```
void check_stiff(double *arg_a, int n)
{
    double *a;
    int i,j;

    a = arg_a -1-n;

    printf("matrix A\n");
    for (i = 1; i <= n; ++i){
        for (j = 1; j <= n; ++j) printf("% 10.3E",a[i+j*n]); printf("\n");
    }
}

void check_matrix2(double *arg_a, double *arg_c, int n)
{
    double *a,*c;
    int i,j;

    a = arg_a -1-n;
    c = arg_c -1;

    printf("matrix A\n");
    for (i = 1; i <= n; ++i){
        for (j = 1; j <= n; ++j) printf("% 10.3E",a[i+j*n]); printf("\n");
    }

    printf("vector c\n");
    for (i = 1; i <= n; ++i) printf("% 10.3E",c[i]); printf("\n");
}
```

```
void check_solution(double *arg_b, int n)
{
    double *b;
    int i;

    b = arg_b-1;

    printf("solution vector b\n");
    for (i = 1; i <= n; ++i) printf("% 10.3E",b[i]); printf("\n");
}
```

1.8.6　境界条件処理の合理化 (1) 右辺

前節で用いた例を再記する。

元の方程式：

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} & A_{16} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} & A_{26} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & A_{36} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} & A_{46} \\ A_{51} & A_{52} & A_{53} & A_{54} & A_{55} & A_{56} \\ A_{61} & A_{62} & A_{63} & A_{64} & A_{65} & A_{66} \end{bmatrix} \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{Bmatrix} = \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{Bmatrix} \tag{1.104}$$

変更後の方程式：

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & A_{22} & 0 & A_{24} & A_{25} & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & A_{42} & 0 & A_{44} & A_{45} & 0 \\ 0 & A_{52} & 0 & A_{54} & A_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{Bmatrix} = \begin{Bmatrix} \tilde{b}_1 \\ c_2 \\ \tilde{b}_3 \\ c_4 \\ c_5 \\ \tilde{b}_6 \end{Bmatrix} - \tilde{b}_1 \begin{Bmatrix} 0 \\ A_{21} \\ 0 \\ A_{41} \\ A_{51} \\ 0 \end{Bmatrix} - \tilde{b}_3 \begin{Bmatrix} 0 \\ A_{23} \\ 0 \\ A_{43} \\ A_{53} \\ 0 \end{Bmatrix} - \tilde{b}_6 \begin{Bmatrix} 0 \\ A_{26} \\ 0 \\ A_{46} \\ A_{56} \\ 0 \end{Bmatrix} \tag{1.105}$$

たとえば、$\tilde{b}_1 = \tilde{b}_6 = 0$ であったとする。このとき右辺第 2,4 項は消滅する。当然ながら不必要な計算は少ない方がプログラム全体のパフォーマンスを向上できるのでこれを

コーディングに反映させる。

定式化	変数名
非零の既知量の数 m_z	n_bc_nonzero
非零の既知量のインデックス $j_z(m)$	i_bc_nonzero(n_bc_nonzero)
非零の既知量 $\tilde{b}_{j_z(m)}$	v_bc_nonzero(n_bc_nonzero)

これを用いて先の bound1 の最初のループを変更する。

1.8.7 fortran コーディング例

fortran でのコーディング例を示すと以下のようになる。

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o stiff.o ex21-23.o gauss_ver4u.o bound2.o misc2.o

TARGET = bound2

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
c bound operation ver.2 with gauss elimination ver.4
c     by WATANABE Hiroshi (2000 June)
c
      implicit real*8(a-h,o-z)
      parameter(n = 5)
      parameter(i_check = 3)
      parameter(i_type = 3)
c
      dimension a(n,n),b(n),c(n)
      dimension i_bc_given(n),v_bc_given(n)
```

```
      dimension i_bc_nonzero(n),v_bc_nonzero(n)
c
      nk = 4
      ak = 1.D0
c
      call stiff(a,n,nk,ak)
      if (i_check .ge. 3) call check_stiff(a,n)

      if (i_type .eq. 1) call ex21(c,n_bc_given,i_bc_given,v_bc_given,n,
     1                             n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
      if (i_type .eq. 2) call ex22(c,n_bc_given,i_bc_given,v_bc_given,n,
     1                             n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
      if (i_type .eq. 3) call ex23(c,n_bc_given,i_bc_given,v_bc_given,n,
     1                             n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c
      call bound2(a,c,n,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
      if (i_check .ge. 2) call check_matrix2(a,c,n)

      call gauss_ver4u(a,c,n)
      if (i_check .ge. 1) call check_solution(c,n)
c
      stop
      end
c
```

stiff.f　bound1 と共通

ex21-23.f

```
c######################################################################
      subroutine ex21(c,n_bc_given,i_bc_given,v_bc_given,n,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension c(*),i_bc_given(*),v_bc_given(*),
     1          i_bc_nonzero(*),v_bc_nonzero(*)
c
      do i = 1,n
          c(i) = 0.D0
      end do
      c(n) = 1.D0
c
      n_bc_given    = 1
```

```
      i_bc_given(1) = 1
      v_bc_given(1) = 0.D0
c
      n_bc_nonzero    = 0
c
      return
      end
c
c######################################################################
      subroutine ex22(c,n_bc_given,i_bc_given,v_bc_given,n,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension c(*),i_bc_given(*),v_bc_given(*),
     1          i_bc_nonzero(*),v_bc_nonzero(*)
c
      do i = 1,n
          c(i) = 0.D0
      end do
      c(3) = 1.D0
c
      n_bc_given    = 2
      i_bc_given(1) = 1
      i_bc_given(2) = 5
      v_bc_given(1) = 0.D0
      v_bc_given(2) = 0.D0
c
      n_bc_nonzero    = 0
c
      return
      end
c
c######################################################################
      subroutine ex23(c,n_bc_given,i_bc_given,v_bc_given,n,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension c(*),i_bc_given(*),v_bc_given(*),
     1          i_bc_nonzero(*),v_bc_nonzero(*)
c
      do i = 1,n
          c(i) = 0.D0
      end do
c
      n_bc_given    = 3
```

```
      i_bc_given(1) = 1
      i_bc_given(2) = 3
      i_bc_given(3) = 5
      v_bc_given(1) = 0.D0
      v_bc_given(2) = 1.D0
      v_bc_given(3) = 0.D0
c
      n_bc_nonzero    = 1
      i_bc_nonzero(1) = 3
      v_bc_nonzero(1) = 1.D0
c
      return
      end
c
```

gauss_ver4u.f　gauss_ver4 と共通

bound2.f

```
c######################################################################
      subroutine bound2(a,c,n,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),c(*),i_bc_given(*),
     1          i_bc_nonzero(*),v_bc_nonzero(*)
c
      do i = 1,n_bc_nonzero
          do k = 1,n
              c(k) = c(k) - v_bc_nonzero(i) * a(k,i_bc_nonzero(i))
          end do
      end do
c
      do i = 1,n_bc_given
          c(i_bc_given(i)) = 0.D0
      end do
c
      do i = 1,n_bc_nonzero
          c(i_bc_nonzero(i)) = v_bc_nonzero(i)
      end do
c
      do i = 1,n_bc_given
          do k = 1,n
              a(k,i_bc_given(i)) = 0.D0
```

```
              a(i_bc_given(i),k) = 0.D0
           end do
           a(i_bc_given(i),i_bc_given(i)) = 1.D0
       end do
c
       return
       end
c
```

misc2.f bound1 と共通

1.8.8 C コーディング例

C でのコーディング例を示すと以下のようになる。

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o stiff.o ex21-23.o gauss_ver4u.o bound2.o misc2.o

TARGET = bound2

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
/* bound operation ver.1 with gauss elimination ver. 4u
     by WATANABE Hiroshi (2000 July) */

#include <stdio.h>
#define N 5
#define I_CHECK 3
#define I_TYPE 3
```

```
void stiff(double *a, int n, int nk, double ak);
void check_stiff(double *a, int n);
void ex21(double *c, int *n_bc_given, int *i_bc_given,
          double *v_bc_given, int n, int *n_bc_nonzero,
          int *i_bc_nonzero, double *v_bc_nonzero);
void ex22(double *c, int *n_bc_given, int *i_bc_given,
          double *v_bc_given, int n, int *n_bc_nonzero,
          int *i_bc_nonzero, double *v_bc_nonzero);
void ex23(double *c, int *n_bc_given, int *i_bc_given,
          double *v_bc_given, int n, int *n_bc_nonzero,
          int *i_bc_nonzero, double *v_bc_nonzero);
void bound2(double *a, double *c, int n, int n_bc_given,
            int *i_bc_given, int n_bc_nonzero,
            int *i_bc_nonzero, double *v_bc_nonzero);
void check_matrix2(double *a, double *c, int n);
void gauss_ver4u(double *a, double *c, int n);
void check_solution(double *b, int n);

main()
{
    double *a,*b,*c,*v_bc_given,*v_bc_nonzero;
    int *i_bc_given,*i_bc_nonzero;
    double ak;
    int n,nk,n_bc_given,n_bc_nonzero;

    n = N;
    nk = 4;
    ak = 1.0;

    a  = (double *)malloc(sizeof(double)*n*n);
    b  = (double *)malloc(sizeof(double)*n);
    c  = (double *)malloc(sizeof(double)*n);
    v_bc_given = (double *)malloc(sizeof(double)*n);
    v_bc_nonzero = (double *)malloc(sizeof(double)*n);
    i_bc_given = (int *)malloc(sizeof(int)*n);
    i_bc_nonzero = (int *)malloc(sizeof(int)*n);

    stiff(a,n,nk,ak);
    if (I_CHECK >= 3) check_stiff(a,n);

    if (I_TYPE == 1) ex21(c,&n_bc_given,i_bc_given,v_bc_given,n,
                          &n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);
    if (I_TYPE == 2) ex22(c,&n_bc_given,i_bc_given,v_bc_given,n,
                          &n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);
    if (I_TYPE == 3) ex23(c,&n_bc_given,i_bc_given,v_bc_given,n,
```

```
                        &n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);

    bound2(a,c,n,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);
    if (I_CHECK >= 2) check_matrix2(a,c,n);

    gauss_ver4u(a,c,n);
    if (I_CHECK >= 1) check_solution(c,n);
}
```

stiff.c bound1 と共通

ex11-13.c

```
void ex21(double *arg_c, int *n_bc_given, int *arg_i_bc_given,
          double *arg_v_bc_given ,int n,
          int *n_bc_nonzero, int *arg_i_bc_nonzero, double *arg_v_bc_nonzero)
{
    double *c, *v_bc_given, *v_bc_nonzero;
    int *i_bc_given, *i_bc_nonzero;
    int i;

    c = arg_c -1;
    v_bc_given   = arg_v_bc_given   -1;
    v_bc_nonzero = arg_v_bc_nonzero -1;
    i_bc_given   = arg_i_bc_given   -1;
    i_bc_nonzero = arg_i_bc_nonzero -1;

    for (i = 1; i <= n; ++i){
        c[i] = 0.0;
    }
    c[n] = 1.0;

    *n_bc_given   = 1;
    i_bc_given[1] = 1;
    v_bc_given[1] = 0.0;

    *n_bc_nonzero = 0;
}

void ex22(double *arg_c, int *n_bc_given, int *arg_i_bc_given,
          double *arg_v_bc_given ,int n,
          int *n_bc_nonzero, int *arg_i_bc_nonzero, double *arg_v_bc_nonzero)
{
```

```
    double *c, *v_bc_given, *v_bc_nonzero;
    int *i_bc_given, *i_bc_nonzero;
    int i;

    c = arg_c -1;
    v_bc_given   = arg_v_bc_given   -1;
    v_bc_nonzero = arg_v_bc_nonzero -1;
    i_bc_given   = arg_i_bc_given   -1;
    i_bc_nonzero = arg_i_bc_nonzero -1;

    for (i = 1; i <= n; ++i){
        c[i] = 0.0;
    }
    c[3] = 1.0;

    *n_bc_given   = 2;
    i_bc_given[1] = 1;
    i_bc_given[2] = 5;
    v_bc_given[1] = 0.0;
    v_bc_given[2] = 0.0;

    *n_bc_nonzero = 0;
}

void ex23(double *arg_c, int *n_bc_given, int *arg_i_bc_given,
          double *arg_v_bc_given ,int n,
          int *n_bc_nonzero, int *arg_i_bc_nonzero, double *arg_v_bc_nonzero)
{
    double *c, *v_bc_given, *v_bc_nonzero;
    int *i_bc_given, *i_bc_nonzero;
    int i;

    c = arg_c -1;
    v_bc_given   = arg_v_bc_given   -1;
    v_bc_nonzero = arg_v_bc_nonzero -1;
    i_bc_given   = arg_i_bc_given   -1;
    i_bc_nonzero = arg_i_bc_nonzero -1;

    for (i = 1; i <= n; ++i){
        c[i] = 0.0;
    }

    *n_bc_given   = 3;
    i_bc_given[1] = 1;
    i_bc_given[2] = 3;
```

```
    i_bc_given[3] = 5;
    v_bc_given[1] = 0.0;
    v_bc_given[2] = 1.0;
    v_bc_given[3] = 0.0;

    *n_bc_nonzero   = 1;
    i_bc_nonzero[1] = 3;
    v_bc_nonzero[1] = 1.0;
}
```

gauss_ver4u.c gauss_ver4 と共通

bound2.c

```
void bound2(double *arg_a, double *arg_c, int n, int n_bc_given,
            int *arg_i_bc_given,
            int n_bc_nonzero, int *arg_i_bc_nonzero, double *arg_v_bc_nonzero)
{
    double *a,*c,*v_bc_given,*v_bc_nonzero;
    int *i_bc_given,*i_bc_nonzero,i,k;

    a = arg_a -1-n;
    c = arg_c -1;
    v_bc_nonzero = arg_v_bc_nonzero -1;
    i_bc_given   = arg_i_bc_given   -1;
    i_bc_nonzero = arg_i_bc_nonzero -1;

    for (i = 1; i <= n_bc_nonzero; ++i){
        for (k = 1; k <= n; ++k){
            c[k] = c[k] - v_bc_nonzero[i] * a[k+i_bc_nonzero[i]*n];
        }
    }

    for (i = 1; i <= n_bc_given; ++i){
        c[i_bc_given[i]] = 0.0;
    }

    for (i = 1; i <= n_bc_nonzero; ++i){
        c[i_bc_nonzero[i]] = v_bc_nonzero[i];
    }

    for (i = 1; i <= n_bc_given; ++i){
        for (k = 1; k <= n; ++k){
    a[k+i_bc_given[i]*n] = 0.0;
```

```
    a[i_bc_given[i]+k*n] = 0.0;
        }
        a[i_bc_given[i]+i_bc_given[i]*n]= 1.0;
    }
}
```

misc2.c　bound1 と共通

第2章　有限要素法の定式化

有限要素法は, 構造解析の分野から発生, 発展した偏微分方程式の数値解析手法の一つである. 数学的な基礎も確立しておりコンピュータとの相性も良いことから固体・構造力学, 流体力学を始めとして現在では電磁気学や量子力学にもその応用分野を広めつつある. ここではまず簡単な微分方程式を例にとり, 有限要素法の数学的基礎について説明し, 有限要素解析コードのプロトタイプを作成する.

2.1　微分方程式の弱形式と有限要素法

以下に示すような微分方程式の境界値問題を考える.

$[B]$　以下の条件を満たす u を求めよ.

$$-\frac{d^2u}{dx^2} = f(x) \qquad (0 < x < a) \tag{2.1}$$

$$u(0) = u(a) = 0 \tag{2.2}$$

$f(x)$ が解析的に積分可能な問題であれば, 式 (2.1) を直接積分することにより解 u が得られるが, ここでは $[B]$ を近似解法の一つである有限要素法により解く.

有限要素法は, 微分方程式の弱形式をもとにした近似解法で, 近似解を求めるために必要なプロセスは以下のようになる.

- 微分方程式を弱形式にする.
- 有限要素補間関数を導入して, 連続的な関数を, 離散的な節点値に置き換える.
- それをもとに連立一次方程式を作成し近似解を求める.

ここでは数学的な厳密さや一般化ということは考えず, このプロセスのみ追ってみる.

2.1.1　微分方程式の弱形式

式 (2.1) の両辺に $v(0) = v(a) = 0$ を満たす 2 階微分可能な関数 v をかけ 0 から a まで積分する.

$$-\int_0^a \frac{d^2u}{dx^2}\, v\, dx = \int_0^a f(x)\, v\, dx \tag{2.3}$$

上式の左辺に部分積分を施せば, 次式を得る.

$$\int_0^a \frac{du}{dx}\frac{dv}{dx}\, dx - \left[\frac{du}{dx}\, v\right]_0^a = \int_0^a f(x)\, v\, dx \tag{2.4}$$

$v(0) = v(a) = 0$ から左辺第 2 項は 0 なので, 次式を得る.

$$\int_0^a \frac{du}{dx}\frac{dv}{dx}\, dx = \int_0^a f(x)\, v\, dx \tag{2.5}$$

上式は $v(0) = v(a) = 0$ の境界条件さえ満たせば任意の v について成立する. またこの証明を逆にたどることにより任意の v について上式が成立するような u は, $[B]$ の解以外には存在しないことが証明できる. すなわち $[B]$ は以下の $[V]$ と等価である.

$[V]$　以下の条件を満たす u を求めよ.

$$\int_0^a \frac{du}{dx}\frac{dv}{dx}\, dx = \int_0^a f(x)\, v\, dx \qquad \forall v \tag{2.6}$$

この $[V]$ はさらに以下の $[M]$ と等価である.

$[M]$　以下の条件を満たす u を求めよ.

$$F(u) \le F(v) \qquad \forall v \tag{2.7}$$

$$F(v) = \int_0^a \left(\frac{1}{2}\frac{dv}{dx}\frac{dv}{dx} - f(x)\, v\right) dx \tag{2.8}$$

$[V] \Leftrightarrow [M]$ の証明は以下のように行えば良い.

まず, $[M] \Rightarrow [V]$ を示す. 式 (2.7) は, 下式と等価である.

$$F(u+v) - F(u) \ge 0 \qquad \forall v \tag{2.9}$$

式 (2.9) を式 (2.8) をもとに書き下せば以下のようになる。

$$\int_0^a \left[\left\{\frac{1}{2}\left(\frac{d(u+v)}{dx}\frac{d(u+v)}{dx}\right) - f(x)\,(u+v)\right\} - \left\{\frac{1}{2}\frac{du}{dx}\frac{du}{dx} - f(x)\,u\right\}\right] dx$$
$$= \int_0^a \left[\left\{\frac{du}{dx}\frac{dv}{dx} - f(x)\,v\right\} + \frac{1}{2}\frac{dv}{dx}\frac{dv}{dx}\right] dx \geq 0 \qquad (2.10)$$

式 (2.9) は, 任意の v について成立しなければならないので, α を任意のスカラーとして v に αv を代入しても成立しなければならない. このとき式 (2.10) は, 以下のように整理される.

$$\alpha \int_0^a \left\{\frac{du}{dx}\frac{dv}{dx} - f(x)\,v\right\} dx + \frac{\alpha^2}{2}\int_0^a \frac{dv}{dx}\frac{dv}{dx}dx \geq 0 \qquad (2.11)$$

$v \equiv 0$ のとき式 (2.11) 左辺は 0 になるので, 式 (2.11) は成立する. $v \not\equiv 0$ のときは 2 次方程式 $y = ax^2 + bx + c$ が $y \geq 0$ であるための必要十分条件は, $a > 0$ かつ $b^2 - 4ac \leq 0$ (判別式) であるが, 式 (2.11) の場合は, $c = 0$ であり, $b = 0$ が必要十分条件となる. 即ち

$$\int_0^a \left\{\frac{dv}{dx}\frac{dv}{dx} - f(x)\,v\right\} dx = 0 \qquad (2.12)$$

$[V] \Rightarrow [M]$ については, 式 (2.6) を用いると, 以下のようになることから証明できる.

$$F(u+v) - F(u) = \int_0^a \left[\left\{\frac{dv}{dx}\frac{dv}{dx} - f(x)\,v\right\} + \frac{1}{2}\frac{dv}{dx}\frac{dv}{dx}\right] dx$$
$$= \int_0^a \frac{1}{2}\frac{dv}{dx}\frac{dv}{dx}dx \geq 0 \qquad (2.13)$$

2.1.2 弱形式の近似解法

以上で $[B]$, $[V]$, $[M]$ が全て等価であることが証明できた. $[B]$ を微分方程式の強形式, $[V]$ を弱形式と呼ぶ. 有限要素法は, $[V]$ を基に近似解を求める方法である. まず以下のように u の定義域 $[0, a]$ を n 個の重なりのない区間 $[x_i, x_{i+1}]\,(i = 1 \sim n)$ に分割する. この $x_i\,(i = 1 \sim n+1)$ を節点と呼ぶ. 近似解は節点 $x_i\,(i = 1 \sim n+1)$ では $u_i\,(i = 1 \sim n+1)$ の値をとり, 節点 x_i と節点 x_{i+1} の間では図 2.1 のように線形的に変化すると仮定する.

このとき式 (2.12) の積分は以下のように変更できる.

$$\sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} \frac{du}{dx}\frac{dv}{dx}\,dx = \sum_{i=1}^{n} \int_{x_i}^{x_{i+1}} f \cdot v\,dx \qquad (2.14)$$

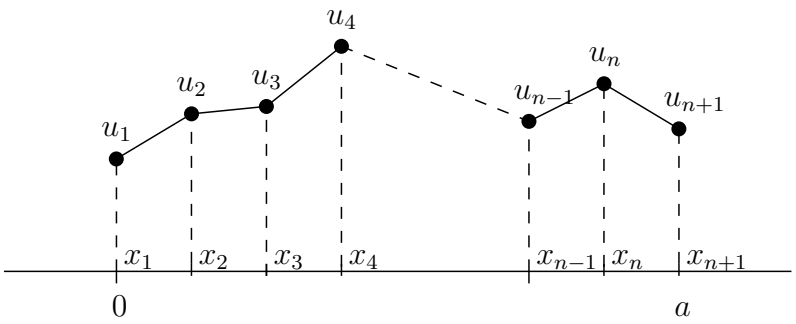


図 2.1: 線形補間

それぞれの積分区間の始点 x_i, 終点 x_{i+1} を便宜的に $x_{(i,1)}$, $x_{(i,2)}$ と表す. $x_{(i,1)}$ が -1, $x_{(i,2)}$ が 1 に対応するようにそれぞれの区間で変数変換を行なう. このとき, パラメータ $r\,(-1 \leq r \leq 1)$ を用いると $[x_{(i,1)}, x_{(i,2)}]$ に含まれる x は, 以下の形式で表される. $N^{(i)}$ を有限要素法補間関数, あるいは単に補間関数と呼ぶ.

$$x = N^{(1)} x_{(i,1)} + N^{(2)} x_{(i,2)} \tag{2.15}$$

$$N^{(1)} = \frac{1}{2}\,(1 - r) \tag{2.16}$$

$$N^{(2)} = \frac{1}{2}\,(1 + r) \tag{2.17}$$

u,v についても同様にパラメータ $r\,(-1 \leq r \leq 1)$ を用いて以下の形式で表す. ただし, $u_{(i,1)} = u_i$, $u_{(i,2)} = u_{i+1}$, $v_{(i,1)} = v_i$, $v_{(i,2)} = v_{i+1}$ である.

$$u = N^{(1)} u_{(i,1)} + N^{(2)} u_{(i,2)} \tag{2.18}$$

$$v = N^{(1)} v_{(i,1)} + N^{(2)} v_{(i,2)} \tag{2.19}$$

式 (2.14) 左辺の被積分関数に含まれている u, v の x に関する微分は, 微分の連鎖則を用いることにより以下のように求められる. u のみ示せば

$$\begin{aligned} \frac{du}{dx} &= \frac{dN^{(1)}}{dx}\, u_{(i,1)} + \frac{dN^{(2)}}{dx}\, u_{(i,2)} \\ &= \frac{dN^{(1)}}{dr}\frac{dr}{dx}\, u_{(i,1)} + \frac{dN^{(2)}}{dr}\frac{dr}{dx}\, u_{(i,2)} \end{aligned} \tag{2.20}$$

$\dfrac{dr}{dx}$ については式 (2.15) を用いて以下のようにインバースとして求められる.

$$\frac{dx}{dr} = \frac{dN^{(1)}}{dr}\, x_{(i,1)} + \frac{dN^{(2)}}{dr}\, x_{(i,2)} \tag{2.21}$$

各積分区間 $[x_{(i,1)}, x_{(i,2)}]$ に対応して, $J_{(i)} = \dfrac{dx}{dr}$ とすれば, 式 (2.14) は下式のようになる.

$$\sum_{i=1}^{n} \int_{-1}^{1} \{v_{(i,1)}\, v_{(i,2)}\} \begin{bmatrix} \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(1)}}{dx} & \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(2)}}{dx} \\ \dfrac{dN^{(2)}}{dx}\dfrac{dN^{(1)}}{dx} & \dfrac{dN^{(2)}}{dx}\dfrac{dN^{(2)}}{dx} \end{bmatrix} \begin{Bmatrix} u_{(i,1)} \\ u_{(i,2)} \end{Bmatrix} J_{(i)}\, dr$$
$$= \sum_{i=1}^{n} \int_{-1}^{1} \{v_{(i,1)}\, v_{(i,2)}\} \begin{Bmatrix} N^{(1)} \\ N^{(2)} \end{Bmatrix} f\, J_{(i)}\, dr \tag{2.22}$$

$v_{(i,1)}$, $v_{(i,2)}$, $u_{(i,1)}$, $u_{(i,2)}$ は節点での値であり r については定数であるから, 積分の外に出すことができる. 下式によって $[K^{(i)}]$, $\{F^{(i)}\}$ を定義する $[K^{(i)}]$ を要素マトリックス等と呼ぶ.

$$[K^{(i)}] = \begin{bmatrix} K_{11}^{(i)} & K_{12}^{(i)} \\ K_{21}^{(i)} & K_{22}^{(i)} \end{bmatrix} = \int_{-1}^{1} \begin{bmatrix} \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(1)}}{dx} & \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(2)}}{dx} \\ \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(2)}}{dx} & \dfrac{dN^{(2)}}{dx}\dfrac{dN^{(2)}}{dx} \end{bmatrix} J_{(i)}\, dr \tag{2.23}$$

$$\{F^{(i)}\} = \begin{Bmatrix} F_1^{(i)} \\ F_2^{(i)} \end{Bmatrix} = \int_{-1}^{1} \begin{Bmatrix} N^{(1)} \\ N^{(2)} \end{Bmatrix} f\, J_{(i)}\, dr \tag{2.24}$$

式 (2.22) の両辺を $\sum$ を用いずに書き下す. ただし式 (2.22) では, $u_{(i,1)}$, $u_{(i,2)}$, $v_{(i,1)}$, $v_{(i,2)}$ として要素毎に記述されている u_i, v_i については節点番号の順に 1 列に並べて以下の形式に表す.

$$\{v_1\, v_2 \ldots v_n\} \begin{bmatrix} K_{11} & K_{12} & \ldots & K_{1n} \\ K_{21} & & & \\ \vdots & & & \\ K_{n1} & \ldots & & K_{nn} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{Bmatrix} = \{v_1\, v_2 \ldots v_n\} \begin{Bmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{Bmatrix} \tag{2.25}$$

式 (2.25) を記述の簡略化のために $\{v\}^T[K]\{u\} = \{v\}^T\{F\}$ と表す. 式 (2.25) は任意の v に対して $\{v\}^T[[K]\{u\} - \{F\}] = 0$ が成立することを要求しているが, これは $[K]\{u\} = \{F\}$ のときのみ成立する.

式 (2.25) であらわれる, 要素マトリックスを重ね合わせた $n \times n$ 次元のマトリックス $[K]$ を, 全体マトリックス等と呼ぶ. $[K]$ を 成分で書けば以下のようになる.

$$[K] = \begin{bmatrix} K_{11}^{(1)} & K_{12}^{(1)} & & & \\ K_{21}^{(1)} & K_{22}^{(1)} + K_{11}^{(2)} & K_{12}^{(2)} & & \\ & K_{21}^{(2)} & K_{22}^{(2)} + K_{11}^{(3)} & K_{12}^{(3)} & \\ & & K_{21}^{(3)} & K_{22}^{(3)} + K_{11}^{(4)} & \\ & & & & \ddots \end{bmatrix} \tag{2.26}$$

また, $\{F\}$ を成分で書けば以下のようになる.

$$\{F\} = \begin{Bmatrix} F_1^{(1)} \\ F_2^{(1)} + F_1^{(2)} \\ F_2^{(2)} + F_1^{(3)} \\ \vdots \end{Bmatrix} \tag{2.27}$$

以上により, 微分方程式の弱形式である式 (2.12) は, $[K]\{u\} = \{F\}$ という形式の連立 1 次方程式により近似解を求められることがわかった. 実際には, $u(0) = u(a) = 0$ の境界条件を付加して解くことになる.

2.2　有限要素解析コードのプロトタイプ

$a = 1$, $f(x) = 1$ $([0, a])$ として, 式 (2.1) の近似解を有限要素法により求めよ.

まず解析的に積分を行なった解は以下のようになる.

$$u = \frac{1}{2} x (1 - x) \tag{2.28}$$

次に要素マトリックスのコーディングを行なう準備をする. 式 (2.21) に従い $\dfrac{dx}{dr}$ を計算する. 式 (2.16), (2.17) より $\dfrac{dN^{(i)}}{dr}$ は以下のようになる.

$$\frac{dN^{(1)}}{dr} = -\frac{1}{2} \tag{2.29}$$

$$\frac{dN^{(2)}}{dr} = \frac{1}{2} \tag{2.30}$$

これを用いると, $\dfrac{dx}{dr}$ は以下のようになる.

$$\begin{aligned}\frac{dx}{dr} &= -\frac{1}{2}\,x_{(i,1)} + \frac{1}{2}\,x_{(i,2)} \\ &= \frac{x_{(i,2)} - x_{(i,1)}}{2}\end{aligned} \tag{2.31}$$

これよりヤコビアン $J_{(i)}$ は以下のようになる.

$$J_{(i)} = \frac{x_{(i,2)} - x_{(i,1)}}{2} \tag{2.32}$$

要素マトリックスで用いる $\dfrac{dN}{dx}$ は,

$$\begin{aligned}\frac{dN^{(1)}}{dx} &= \frac{dN^{(1)}}{dr}\frac{dr}{dx} = -\frac{1}{2}\cdot\frac{2}{x_{(i,2)} - x_{(i,1)}} \\ &= -\frac{1}{x_{(i,2)} - x_{(i,1)}}\end{aligned} \tag{2.33}$$

$$\begin{aligned}\frac{dN^{(2)}}{dx} &= \frac{dN^{(2)}}{dr}\frac{dr}{dx} = \frac{1}{2}\cdot\frac{2}{x_{(i,2)} - x_{(i,1)}} \\ &= -\frac{1}{x_{(i,2)} - x_{(i,1)}}\end{aligned} \tag{2.34}$$

要素マトリックスの被積分関数は,

$$\begin{aligned}&\begin{bmatrix} \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(1)}}{dx} & \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(2)}}{dx} \\ \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(2)}}{dx} & \dfrac{dN^{(2)}}{dx}\dfrac{dN^{(2)}}{dx} \end{bmatrix} J_{(i)} \\ &= \begin{bmatrix} \dfrac{-1}{x_{(i,2)} - x_{(i,1)}}\cdot\dfrac{-1}{x_{(i,2)} - x_{(i,1)}} & \dfrac{-1}{x_{(i,2)} - x_{(i,1)}}\cdot\dfrac{1}{x_{(i,2)} - x_{(i,1)}} \\ \dfrac{-1}{x_{(i,2)} - x_{(i,1)}}\cdot\dfrac{1}{x_{(i,2)} - x_{(i,1)}} & \dfrac{1}{x_{(i,2)} - x_{(i,1)}}\cdot\dfrac{1}{x_{(i,2)} - x_{(i,1)}} \end{bmatrix}\cdot\frac{x_{(i,2)} - x_{(i,1)}}{2} \\ &= \frac{1}{2(x_{(i,2)} - x_{(i,1)})}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\end{aligned} \tag{2.35}$$

これより要素マトリックス $K^{(i)}$ は下式になる.

$$[K^{(i)}] = \frac{1}{x_{(i,2)} - x_{(i,1)}}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.36}$$

次に式 (2.24) は, $f(x) = 1$ なので以下のようになる.

$$\begin{aligned} F_1^{(i)} &= \int_{-1}^{1} \frac{1}{2}(1-r) \cdot \frac{x_{(i,2)} - x_{(i,1)}}{2}\, dr \\ &= \frac{1}{2}(x_{(i,2)} - x_{(i,1)}) \end{aligned} \tag{2.37}$$

$$\begin{aligned} F_2^{(i)} &= \int_{-1}^{1} \frac{1}{2}(1+r) \cdot \frac{x_{(i,2)} - x_{(i,1)}}{2}\, dr \\ &= \frac{1}{2}(x_{(i,2)} - x_{(i,1)}) \end{aligned} \tag{2.38}$$

以上を基に, 実際に有限要素法解析コードのプロトタイプを作成する. 最初から本格的なものにトライすると得てしてデバッグに時間をとられるので, 冗長ではあるがまずは非常に単純化したものから取り掛かり, 順次拡張していくことにする.

2.2.1　バージョン 1 — すべての区間が等しいと仮定

要素数を N とおくと, 式 (2.36), (2.37), (2.38) に出てくる $x_{(i,2)} - x_{(i,1)}$ はすべて $1/N$ で一定で, $[K^{(i)}]$, $\{F^{(i)}\}$ はすべての要素 i で同一である. この時, 例えば以下のようにコーディングすることにより $[K]$, $\{F\}$ が得られる.

$x = 1/N$

for　$i = 1 \sim N$

$K_{i,i} = K_{i,i} + 1/x$

$K_{i+1,i+1} = K_{i+1,i+1} + 1/x$

$K_{i+1,i} = K_{i+1,i} - 1/x$

$K_{i,i+1} = K_{i,i+1} - 1/x$

$F_i = F_i + x/2$

$F_{i+1} = F_{i+1} + x/2$

end for

得られた $[K]$, $\{F\}$ を第 1 章で作った Gauss の消去法および境界条件処理により求解すればよい.

2.2.2 fortran コーディング例

fortran コーディング例を示すと以下のようになる.

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o stiff1.o misc.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
      implicit real*8(a-h,o-z)
      parameter (N = 3)
      dimension a(N+1,N+1),b(N+1)
      dimension i_bc_given(N+1),i_bc_nonzero(N+1),v_bc_nonzero(N+1)
c
      call stiff1(a,b,N)
c
      call bc1(n_bc_given,i_bc_given,n_bc_nonzero,N)

      call bound2(a,b,N+1,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c
      call gauss_ver4u(a,b,N+1)
c
      call check_solution(b,N)
c
      stop
      end
c
```

stiff1.f

```
c######################################################################
      subroutine stiff1(a,b,N)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(N+1,*),b(*)
c
      do i = 1,N+1
          do j = 1,N+1
              a(i,j) = 0.D0
          end do
      end do
c
      do i = 1,N+1
          b(i) = 0.D0
      end do
c
      x = 1.D0 / dfloat(N)
c
      do i = 1,N
          a(i  ,i  ) = a(i  ,i  ) + 1.D0 / x
          a(i+1,i+1) = a(i+1,i+1) + 1.D0 / x
          a(i+1,i  ) = a(i+1,i  ) - 1.D0 / x
          a(i  ,i+1) = a(i  ,i+1) - 1.D0 / x
c
          b(i  ) = b(i  ) + x / 2.D0
          b(i+1) = b(i+1) + x / 2.D0
      end do
c
      return
      end
c
```

misc1.f

```
c######################################################################
      subroutine bc1(n_bc_given,i_bc_given,n_bc_nonzero,N)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension i_bc_given(*)
c
      n_bc_given    = 2
      i_bc_given(1) = 1
      i_bc_given(2) = N+1
c
      n_bc_nonzero    = 0
```

```
c
      return
      end
c
c#######################################################################
      subroutine check_solution(b,N)
c#######################################################################
      implicit real*8(a-h,o-z)
      dimension b(*)
c
      do i = 1,N+1
          x = dfloat(i-1) / dfloat(N)
          write(*,1000) b(i), 0.5D0*(x-x*x)
      end do
 1000 format(2x,e13.6,2x,e13.6)
c
      return
      end
c
```

gauss_ver4u.f 第 1 章と共通

bound2.f 第 1 章と共通

2.2.3 C コーディング例

C コーディング例を示すと以下のようになる.

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o stiff1.o misc1.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)
```

```
.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
#include <stdio.h>
#define N 3

/* file stiff1.c */
void stiff1(double *a, double *b, int n);

/* file misc1.c */
void bc1(int *n_bc_given, int *arg_i_bc_given, int *n_bc_nonzero,int n);
void check_solution(double *b,int n);

/* file gauss_ver4u.c */
void gauss_ver4u(double *a, double *c, int n);

/* file bound2.c */
void bound2(double *a, double *c, int n, int n_bc_given,
            int *i_bc_given, int n_bc_nonzero,
            int *i_bc_nonzero, double *v_bc_nonzero);

main()
{
    double *a,*b,*v_bc_nonzero;
    int n,n_bc_given,*i_bc_given,n_bc_nonzero,*i_bc_nonzero;

    n = N;

    a  = (double *)malloc(sizeof(double)*(n+1)*(n+1));
    b  = (double *)malloc(sizeof(double)*(n+1));
    v_bc_nonzero = (double *)malloc(sizeof(double)*(n+1));
    i_bc_given   = (int *)malloc(sizeof(int)*(n+1));
    i_bc_nonzero = (int *)malloc(sizeof(int)*(n+1));

    stiff1(a,b,n);

    bc1(&n_bc_given,i_bc_given,&n_bc_nonzero,n);

    bound2(a,b,n+1,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);

    gauss_ver4u(a,b,n+1);
```

```
    check_solution(b,n);
}
```

stiff1.c

```
void stiff1(double *arg_a, double *arg_b, int n)
{
    double *a, *b;
    double x;
    int i,j;

    a = arg_a -1 -(n+1);
    b = arg_b -1;

    for (i=1; i<=n+1; ++i){
        for (j=1; j<=n+1; ++j){
            a[i+j*(n+1)] = 0.0;
}
    }

    for (i=1; i<=n+1; ++i){
        b[i] = 0.0;
    }

    x = 1.0 / (double) n;

    for (i=1; i<=n; ++i){
        a[ i   + i   *(n+1)] = a[ i   + i   *(n+1)] + 1.0 / x;
        a[(i+1)+(i+1)*(n+1)] = a[(i+1)+(i+1)*(n+1)] + 1.0 / x;
        a[(i+1)+ i   *(n+1)] = a[(i+1)+ i   *(n+1)] - 1.0 / x;
        a[ i   +(i+1)*(n+1)] = a[ i   +(i+1)*(n+1)] - 1.0 / x;
        b[i  ] = b[i  ] + x / 2.0;
        b[i+1] = b[i+1] + x / 2.0;
    }
}
```

misc1.c

```
void bc1(int *n_bc_given, int *arg_i_bc_given, int *n_bc_nonzero,int n)
{
    int *i_bc_given;
    i_bc_given = arg_i_bc_given -1;
```

```
    *n_bc_given    = 2;
    i_bc_given[1] = 1;
    i_bc_given[2] = n+1;

    *n_bc_nonzero    = 0;
}

void check_solution(double *arg_b,int n)
{
    double *b;
    int i;
    double x;

    b = arg_b -1;

    for (i=1; i<= n+1; ++i){
        x = (double) (i-1) / (double) n;
        printf("  % 10.3E  % 10.3E\n",b[i],0.5*(x-x*x));
    }
}
```

gauss_ver4u.c　第 1 章と共通

bound2.c　第 1 章と共通

2.2.4　バージョン 2 — 要素の長さが可変

今回考えているような単純な問題は別にして, 一般的な有限要素法解析コードでは節点座標を入力ファイルとして与える. また, それぞれの要素がどの節点から構成されているかについての情報も併せて入力ファイルで与える. ここでは以下の変数名を用いる.

- 総節点数　`nnode`
- 節点座標　`coords`(`1..nnode`)
- 総要素数　`nelem`
- コネクティビティ　`lnods`(1..2, `1..nelem`)

fortran では入力データに合わせて動的にメモリーを確保することができないので, 以下のように配列をあらかじめ大きめに取ることにする. また, これに対応し全体マトリックスも 2 次元で宣言せず 1 次元配列にし, プログラム中で切り分けることにする.

```
      parameter (MXNODE = 100)
      parameter (MXELEM = 100)
      dimension a(MXNODE*MXNODE),b(MXNODE)
      dimension coords(MXNODE),lnods(2,MXELEM)
      dimension i_bc_given(MXNODE),i_bc_nonzero(MXNODE),
     1          v_bc_nonzero(MXNODE)
```

C では, 入力ファイルから総節点数, 総要素数を読み込み, メモリーを確保し, 改めて入力ファイルからデータを読み込むことにする.

```
double *a,*b,*v_bc_nonzero;
int n_bc_given,*i_bc_given,n_bc_nonzero,*i_bc_nonzero;
double *coords;
int nnode,nelem,*lnods,i;

datain1(&nnode,&nelem);

a  = (double *)malloc(sizeof(double)*(nnode)*(nnode));
b  = (double *)malloc(sizeof(double)*(nnode));
v_bc_nonzero = (double *)malloc(sizeof(double)*(nnode));
i_bc_given   = (int *)malloc(sizeof(int)*(nnode));
i_bc_nonzero = (int *)malloc(sizeof(int)*(nnode));
coords = (double *)malloc(sizeof(double)*(nnode));
lnods = (int *)malloc(sizeof(int)*(nelem)*2);

datain(&nnode,coords,&nelem,lnods);
```

まず適当な形式の入力データファイルを作る. ただしここでは節点座標は, 小さいものから順に並んでいるものとする. (間隔は均一である必要はない)

ファイル		プログラム中の変数
11		nnode
1	0.0	coords(1)
2	0.1	coords(2)
⋮		⋮
10	0.9	coords(10)
11	1.0	coords(11)
10		nelem
1	1 2	lnods(1,1), lnods(2,1)
2	2 3	lnods(1,2), lnods(2,2)
⋮		⋮
10	10 11	lnods(1,10), lnods(2,10)

表 2.1: データファイルの例

全体マトリックスを作る部分は以下のように変更される.

for $i = 1 \sim$ nelem

$x =$ coords(lnods$(2, i)$) − coords(lnods$(1, i)$)

$K_{i,i} = K_{i,i} + 1/x$

$K_{i+1,i+1} = K_{i+1,i+1} + 1/x$

$K_{i+1,i} = K_{i+1,i} - 1/x$

$K_{i,i+1} = K_{i,i+1} - 1/x$

$F_i = F_i + x/2$

$F_{i+1} = F_{i+1} + x/2$

end for

バージョン 1 と同様に境界条件処理を行ない Gauss の消去法により求解する.

2.2.5 入力ファイル例

入力ファイルは fortran, C とも共通である.

temp.dat

```
11
 1 0.0
 2 0.1
 3 0.2
 4 0.3
 5 0.4
 6 0.5
 7 0.6
 8 0.7
 9 0.8
10 0.9
11 1.0
10
 1  1  2
 2  2  3
 3  3  4
 4  4  5
 5  5  6
 6  6  7
 7  7  8
 8  8  9
 9  9 10
10 10 11
```

2.2.6 fortran コーディング例

fortran コーディング例を示すと以下のようになる.

Makefile

```
FC = g77
F_OPT = -O2
```

```
OBJS = main.o stiff2.o misc2.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
      implicit real*8(a-h,o-z)
      parameter (MXNODE = 100)
      parameter (MXELEM = 100)
      dimension a(MXNODE*MXNODE),b(MXNODE)
      dimension coords(MXNODE),lnods(2,MXELEM)
      dimension i_bc_given(MXNODE),i_bc_nonzero(MXNODE),
     1          v_bc_nonzero(MXNODE)
c
      call datain(nnode,coords,nelem,lnods)
c
      call stiff2(a,b,nnode,coords,nelem,lnods)
c
      call bc2(n_bc_given,i_bc_given,n_bc_nonzero,nnode)
c
      call bound2(a,b,nnode,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c
      call gauss_ver4u(a,b,nnode)
c
      call check_solution2(b,nnode,coords)
c
      stop
      end
c
```

stiff2.f

```
c######################################################################
      subroutine stiff2(a,b,nnode,coords,nelem,lnods)
c######################################################################
```

```
      implicit real*8(a-h,o-z)
      dimension a(nnode,*),b(*),coords(*),lnods(2,*)
c
      do i = 1,nnode
          do j = 1,nnode
              a(i,j) = 0.D0
          end do
      end do
c
      do i = 1,nnode
          b(i) = 0.D0
      end do
c
      do i = 1,nelem
          x = coords(lnods(2,i))-coords(lnods(1,i))
          a(i  ,i  ) = a(i  ,i  ) + 1.D0 / x
          a(i+1,i+1) = a(i+1,i+1) + 1.D0 / x
          a(i+1,i  ) = a(i+1,i  ) - 1.D0 / x
          a(i  ,i+1) = a(i  ,i+1) - 1.D0 / x
c
          b(i  ) = b(i  ) + x / 2.D0
          b(i+1) = b(i+1) + x / 2.D0
      end do
c
      return
      end
c
```

misc2.f

```
c######################################################################
      subroutine datain(nnode,coords,nelem,lnods)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(*),lnods(2,*)
c
      open(10,file='temp.dat')
c
      read(10,*) nnode
      do i = 1, nnode
          read(10,*) itemp, coords(i)
      end do
      read(10,*) nelem
      do i = 1,nelem
          read(10,*) itemp, (lnods(j,i),j=1,2)
```

```
      end do
c
      close(10)
c
      return
      end
c
c######################################################################
      subroutine bc2(n_bc_given,i_bc_given,n_bc_nonzero,nnode)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension i_bc_given(*)
c
      n_bc_given    = 2
      i_bc_given(1) = 1
      i_bc_given(2) = nnode
c
      n_bc_nonzero    = 0
c
      return
      end
c
c######################################################################
      subroutine check_solution2(b,nnode,coords)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension b(*),coords(*)
c
      do i = 1,nnode
          x = coords(i)
          write(*,1000) b(i), 0.5D0*(x-x*x)
      end do
 1000 format(2x,e13.6,2x,e13.6)
c
      return
      end
c
```

gauss_ver4u.f　第 1 章と共通

bound2.f　第 1 章と共通

2.2.7 C コーディング例

C コーディング例を示すと以下のようになる.

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o stiff2.o misc2.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
#include <stdio.h>

/* file stiff2.c */
void stiff2(double *a, double *b, int nnode, double *coords,
            int nelem, int *lnods);

/* file misc2.c */
void datasize2(int *nnode,int *nelem);
void datain2(int nnode,double *coords, int nelem, int *lnods);
void bc2(int *n_bc_given, int *i_bc_given, int *n_bc_nonzero,int nnode);
void check_solution2(double *b,int nnode, double *coords);

/* file gauss_ver4u.c */
void gauss_ver4u(double *a, double *c, int n);

/* file bound2.c */
void bound2(double *a, double *c, int n, int n_bc_given,
            int *i_bc_given, int n_bc_nonzero,
            int *i_bc_nonzero, double *v_bc_nonzero);

main()
{
```

```
    double *a,*b,*v_bc_nonzero;
    int n_bc_given,*i_bc_given,n_bc_nonzero,*i_bc_nonzero;
    double *coords;
    int nnode,nelem,*lnods,i;

    datasize2(&nnode,&nelem);

    a  = (double *)malloc(sizeof(double)*(nnode)*(nnode));
    b  = (double *)malloc(sizeof(double)*(nnode));
    v_bc_nonzero = (double *)malloc(sizeof(double)*(nnode));
    i_bc_given   = (int *)malloc(sizeof(int)*(nnode));
    i_bc_nonzero = (int *)malloc(sizeof(int)*(nnode));
    coords = (double *)malloc(sizeof(double)*(nnode));
    lnods  = (int *)malloc(sizeof(int)*(nelem)*2);

    datain2(nnode,coords,nelem,lnods);

    stiff2(a,b,nnode,coords,nelem,lnods);

    bc2(&n_bc_given,i_bc_given,&n_bc_nonzero,nnode);

    bound2(a,b,nnode,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);

    gauss_ver4u(a,b,nnode);

    check_solution2(b,nnode,coords);
}
```

stiff2.c

```
void stiff2(double *arg_a, double *arg_b, int nnode, double *arg_coords,
            int nelem, int *arg_lnods)
{
    double *a, *b;
    double x;
    int i,j;
    double *coords;
    int *lnods;

    a = arg_a -1 -nnode;
    b = arg_b -1;

    coords = arg_coords -1;
    lnods = arg_lnods -1 -2;
```

```
    for (i=1; i<=nnode; ++i){
        for (j=1; j<=nnode; ++j){
            a[i+j*nnode] = 0.0;
}
    }

    for (i=1; i<=nnode; ++i){
        b[i] = 0.0;
    }

    for (i=1; i<=nelem; ++i){
        x = coords[lnods[2+i*2]]-coords[lnods[1+i*2]];
        a[ i   + i   *nnode] = a[ i   + i   *nnode] + 1.0 / x;
        a[(i+1)+(i+1)*nnode] = a[(i+1)+(i+1)*nnode] + 1.0 / x;
        a[(i+1)+ i   *nnode] = a[(i+1)+ i   *nnode] - 1.0 / x;
        a[ i   +(i+1)*nnode] = a[ i   +(i+1)*nnode] - 1.0 / x;
        b[i  ] = b[i  ] + x / 2.0;
        b[i+1] = b[i+1] + x / 2.0;
    }
}
```

misc2.c

```
#include <stdio.h>
void datasize2(int *nnode,int *nelem)
{
    FILE *fp;
    double coords;
    int i,inode,ielem,lnods1,lnods2;

    if ((fp = fopen("temp.dat","r")) == NULL){
        printf("can't open file\n");
        exit(1);
    }

    fscanf(fp,"%d",nnode);
    for (i = 1; i<=*nnode; ++i){
        fscanf(fp,"%d %lf", &inode, &coords);
    }
    fscanf(fp,"%d",nelem);
    for (i = 1; i<=*nelem; ++i){
        fscanf(fp,"%d %d %d",&ielem,&lnods1,&lnods2);
    }
```

```
    fclose(fp);
}

void datain2(int nnode,double *arg_coords, int nelem, int *arg_lnods)
{
    FILE *fp;
    int inode,ielem,idum;
    double *coords;
    int *lnods;

    coords = arg_coords -1;
    lnods = arg_lnods -1-2;

    if ((fp = fopen("temp.dat","r")) == NULL){
        printf("can't open file\n");
        exit(1);
    }

    fscanf(fp,"%d",&idum);
    for (inode = 1; inode <= nnode; ++inode){
        fscanf(fp,"%d %lf", &idum, &coords[inode]);
    }

    fscanf(fp,"%d",&idum);
    for (ielem = 1; ielem <= nelem; ++ielem){
        fscanf(fp,"%d %d %d",&idum, &lnods[1+ielem*2],&lnods[2+ielem*2]);
    }

    fclose(fp);
}

void bc2(int *n_bc_given, int *arg_i_bc_given, int *n_bc_nonzero,int nnode)
{
    int *i_bc_given;
    i_bc_given = arg_i_bc_given -1;

    *n_bc_given    = 2;
    i_bc_given[1] = 1;
    i_bc_given[2] = nnode;

    *n_bc_nonzero    = 0;
}

void check_solution2(double *arg_b,int nnode, double *arg_coords)
{
```

```
    double *b,*coords;
    int i;
    double x;

    b = arg_b -1;
    coords = arg_coords -1;

    for (i=1; i<=nnode; ++i){
        x = coords[i];
        printf("  % 10.3E  % 10.3E\n",b[i],0.5*(x-x*x));
    }
}
```

gauss_ver4u.c　第 1 章と共通

bound2.c　第 1 章と共通

2.2.8　バージョン 3 — 境界条件処理の拡張

これまでは境界条件処理は両端 2 点で共に 0 として行なってきたが, 0 以外の値を取る場合を考える. 例として $u(0) = 1$, $u(1) = 0$ と $u(0) = 1$, $u(1) = 2$ の場合を考える. 解析的に積分を行なった解はそれぞれ以下のようになる.

$$u = -\frac{1}{2}\,x^2 - \frac{1}{2}\,x + 1 \tag{2.39}$$

$$u = -\frac{1}{2}\,x^2 + \frac{3}{2}\,x + 1 \tag{2.40}$$

バージョン 2 のプログラムを変更し, 入力ファイルから境界条件に関する情報を読み込むようにする. ここではデータのわかりやすさを考えて, 各節点ごとに境界条件を与えるか否かを指定し, 与える節点を加えて `n_bc_given` を計算する. 非零の境界条件が与えられるものについては, 以下のように入力ファイルの最後にデータを付け加える.

2.2.9　入力ファイル例

入力ファイルは fortran, C とも共通である.

ファイル	プログラム中の変数
1	n_bc_nonzero
1　1　1.0	i_bc_nonzero(1), v_bc_nonzero(1)

表 2.2: データファイルの例の追加

ファイル	プログラム中の変数
2	n_bc_nonzero
1　1　1.0	i_bc_nonzero(1), v_bc_nonzero(1)
2　11　2.0	i_bc_nonzero(2), v_bc_nonzero(2)

表 2.3: データファイルの例の追加

入力ファイル case1.dat

```
11
 1  1  0.0
 2  0  0.1
 3  0  0.2
 4  0  0.3
 5  0  0.4
 6  0  0.5
 7  0  0.6
 8  0  0.7
 9  0  0.8
10  0  0.9
11  1  1.0
10
 1  1  2
 2  2  3
 3  3  4
 4  4  5
 5  5  6
 6  6  7
 7  7  8
 8  8  9
 9  9 10
10 10 11
1
 1  1  1.0
```

入力ファイル case2.dat

```
11
 1  1  0.0
 2  0  0.1
 3  0  0.2
 4  0  0.3
 5  0  0.4
 6  0  0.5
 7  0  0.6
 8  0  0.7
 9  0  0.8
10  0  0.9
11  1  1.0
10
 1  1  2
 2  2  3
 3  3  4
 4  4  5
 5  5  6
 6  6  7
 7  7  8
 8  8  9
 9  9 10
10 10 11
2
 1  1  1.0
 2 11  2.0
```

2.2.10 fortran コーディング例

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o stiff2.o misc3.o gauss_ver4u.o bound2.o

TARGET = onedim
```

```
$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
      implicit real*8(a-h,o-z)
      parameter (MXNODE = 100)
      parameter (MXELEM = 100)
      dimension a(MXNODE*MXNODE),b(MXNODE)
      dimension coords(MXNODE),lnods(2,MXELEM)
      dimension i_bc_given(MXNODE),i_bc_nonzero(MXNODE),
     1          v_bc_nonzero(MXNODE)
c
      icase = 2
c
      call datain3(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase)
c
      call stiff2(a,b,nnode,coords,nelem,lnods)
c
      call bound2(a,b,nnode,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c
      call gauss_ver4u(a,b,nnode)
c
      call check_solution3(b,nnode,coords,icase)
c
      stop
      end
c
```

stiff2.f　バージョン 2 と共通

misc3.f

```
c######################################################################
      subroutine datain3(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase)
```

```
c######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(*),lnods(2,*),i_bc_given(*),i_bc_nonzero(*),
     1          v_bc_nonzero(*)
c
      if (icase .eq. 1) open(10,file='case1.dat')
      if (icase .eq. 2) open(10,file='case2.dat')
c
      n_bc_given = 0
      read(10,*) nnode
      do i = 1, nnode
          read(10,*) itemp, ibc,coords(i)
          if (ibc .eq. 1) then
              n_bc_given = n_bc_given + 1
              i_bc_given(n_bc_given) = i
          end if
      end do
c
      read(10,*) nelem
      do i = 1,nelem
          read(10,*) itemp, (lnods(j,i),j=1,2)
      end do
c
      read(10,*) n_bc_nonzero
      do i = 1,n_bc_nonzero
          read(10,*) itemp, i_bc_nonzero(i),v_bc_nonzero(i)
      end do
c
      close(10)
c
      return
      end
c
c######################################################################
      subroutine check_solution3(b,nnode,coords,icase)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension b(*),coords(*)
c
      do i = 1,nnode
          x = coords(i)
          if (icase .eq. 1) u = -0.5D0 * x*x - 0.5D0 * x + 1
          if (icase .eq. 2) u = -0.5D0 * x*x + 1.5D0 * x + 1
          write(*,1000) b(i), u
      end do
```

```
 1000 format(2x,e13.6,2x,e13.6)
c
      return
      end
c
```

gauss_ver4u.f　第 1 章と共通

bound2.f　第 1 章と共通

2.2.11　C コーディング例

C コーディング例を示すと以下のようになる.

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o stiff2.o misc3.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
#include <stdio.h>

/* file stiff2.c */
void stiff2(double *a, double *b, int nnode, double *coords,
            int nelem, int *lnods);

/* file misc3.c */
```

```
void datasize3(int *nnode, int *nelem, int *n_bc_given, int *n_bc_nonzero,
               int icase);
void datain3(int nnode,double *coords, int nelem, int *lnods,
             int n_bc_given, int *i_bc_given,int n_bc_nonzero,
             int *i_bc_nonzero,double *v_bc_nonzero,int icase);
void check_solution3(double *b,int nnode, double *coords,int icase);

/* file gauss_ver4u.c */
void gauss_ver4u(double *a, double *c, int n);

/* file bound2.c */
void bound2(double *a, double *c, int n, int n_bc_given,
            int *i_bc_given, int n_bc_nonzero,
            int *i_bc_nonzero, double *v_bc_nonzero);

main()
{
    double *a,*b,*v_bc_nonzero,*coords;
    int n_bc_given,*i_bc_given,n_bc_nonzero,*i_bc_nonzero,
        nnode,nelem,*lnods,icase;

    icase = 2;

    datasize3(&nnode,&nelem,&n_bc_given,&n_bc_nonzero,icase);

    a  = (double *)malloc(sizeof(double)*(nnode)*(nnode));
    b  = (double *)malloc(sizeof(double)*(nnode));
    coords = (double *)malloc(sizeof(double)*(nnode));
    lnods  = (int *)malloc(sizeof(int)*(nelem)*2);
    i_bc_given   = (int *)malloc(sizeof(int)*(n_bc_given));
    i_bc_nonzero = (int *)malloc(sizeof(int)*(n_bc_nonzero));
    v_bc_nonzero = (double *)malloc(sizeof(double)*(n_bc_nonzero));

    datain3(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase);

    stiff2(a,b,nnode,coords,nelem,lnods);

    bound2(a,b,nnode,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);

    gauss_ver4u(a,b,nnode);

    check_solution3(b,nnode,coords,icase);
}
```

stiff2.c バージョン2と共通

misc3.c

```
#include <stdio.h>
void datasize3(int *nnode, int *nelem, int *n_bc_given, int *n_bc_nonzero,
               int icase)
{
    FILE *fp;
    double coords;
    int i,inode,ielem,lnods1,lnods2,ibc;

    if (icase == 1){
        if ((fp = fopen("case1.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 2){
        if ((fp = fopen("case2.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }

    *n_bc_given = 0;
    fscanf(fp,"%d",nnode);
    for (i = 1; i <= *nnode; ++i){
        fscanf(fp,"%d %d %lf", &inode, &ibc, &coords);
        if (ibc == 1){
            *n_bc_given = *n_bc_given + 1;
        }
    }

    fscanf(fp,"%d",nelem);
    for (i = 1; i <= *nelem; ++i){
        fscanf(fp,"%d %d %d",&ielem,&lnods1,&lnods2);
    }

    fscanf(fp,"%d",n_bc_nonzero);

    fclose(fp);
}

void datain3(int nnode,double *arg_coords, int nelem, int *arg_lnods,
```

```
            int n_bc_given, int *arg_i_bc_given,int n_bc_nonzero,
            int *arg_i_bc_nonzero,double *arg_v_bc_nonzero,int icase)
{
    FILE *fp;
    int i,inode,ielem,ibc,idum,*lnods, *i_bc_given, *i_bc_nonzero;
    double *coords, *v_bc_nonzero;

    coords = arg_coords -1;
    lnods = arg_lnods -1-2;
    i_bc_given = arg_i_bc_given -1;
    i_bc_nonzero = arg_i_bc_nonzero -1;
    v_bc_nonzero = arg_v_bc_nonzero -1;

    if (icase == 1){
        if ((fp = fopen("case1.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 2){
        if ((fp = fopen("case2.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }

    i = 0;
    fscanf(fp,"%d",&idum);
    for (inode = 1; inode <= nnode; ++inode){
        fscanf(fp,"%d %d %lf", &idum, &ibc, &coords[inode]);
        if (ibc == 1){
            i = i + 1;
            i_bc_given[i] = inode;
        }
    }

    fscanf(fp,"%d",&idum);
    for (ielem = 1; ielem <= nelem; ++ielem){
        fscanf(fp,"%d %d %d",&idum,&lnods[1+ielem*2],&lnods[2+ielem*2]);
    }

    fscanf(fp,"%d",&idum);
    for (i = 1; i <= n_bc_nonzero; ++i){
        fscanf(fp,"%d %d %lf",&ibc,&i_bc_nonzero[i],&v_bc_nonzero[i]);
    }
```

```
    fclose(fp);
}

void check_solution3(double *arg_b,int nnode, double *arg_coords, int icase)
{
    double *b,*coords;
    int i;
    double x,u;

    b = arg_b -1;
    coords = arg_coords -1;

    for (i=1; i<=nnode; ++i){
        x = coords[i];
        if (icase == 1) u = -0.5 * x*x - 0.5 * x + 1.0;
        if (icase == 2) u = -0.5 * x*x + 1.5 * x + 1.0;
        printf("  % 10.3E  % 10.3E\n",b[i],u);
    }
}
```

gauss_ver4u.c　第 1 章と共通

bound2.c　第 1 章と共通

2.2.12　バージョン 4 — 要素マトリックスを作ってから全体マトリックスに加える

これまで全体マトリックスを直接作ってきたが, 一般の有限要素解析コードでは, まず要素マトリックスを作り, それを全体マトリックスに加える構造になっている. これは要素マトリックスは鉄, ゴム, 流体など解析対象により異なるが, 残りの部分はほとんど共通にすることができるからである. なお, この処理は merge (マージ) と呼ばれ, プログラム中で用いられるサブルーチン名もそれに関連した名称が用いられている。

最終的には要素マトリックスと merge は分離して別サブルーチンにするが, 分離する前

のものであれば以下のようなコーディングになる。

for $i = 1 \sim \texttt{nelem}$

$x = \texttt{coords}(\texttt{lnods}\,(2, i)) - \texttt{coords}(\texttt{lnods}(1, i))$

$A_{11} = 1/x$

$A_{12} = -1/x$

$A_{21} = -1/x$

$A_{22} = 1/x$

$ip1 = \texttt{lnods}(1, i)$

$ip2 = \texttt{lnods}(2, i)$

$K_{ip1,ip1} = K_{ip1,ip1} + A_{11}$

$K_{ip1,ip2} = K_{ip1,ip2} + A_{12}$

$K_{ip2,ip1} = K_{ip2,ip1} + A_{21}$

$K_{ip2,ip2} = K_{ip2,ip2} + A_{22}$

end for

2.2.13 入力ファイル例

入力ファイルは fortran, C とも共通である.

入力ファイル case1.dat バージョン 3 と共通

入力ファイル case2.dat バージョン 3 と共通

2.2.14 fortran コーディング例

Makefile

```
FC = g77
```

```
F_OPT = -O2

OBJS = main.o stiff4.o misc3.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
      implicit real*8(a-h,o-z)
      parameter (MXNODE = 100)
      parameter (MXELEM = 100)
      dimension a(MXNODE*MXNODE),b(MXNODE)
      dimension coords(MXNODE),lnods(2,MXELEM)
      dimension i_bc_given(MXNODE),i_bc_nonzero(MXNODE),
     1          v_bc_nonzero(MXNODE)
c
      icase = 2
c
      call datain3(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase)
c
      call stiff4(a,b,nnode,coords,nelem,lnods)
c
      call bound2(a,b,nnode,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c
      call gauss_ver4u(a,b,nnode)
c
      call check_solution3(b,nnode,coords,icase)
c
      stop
      end
c
```

stiff4.f

```
c######################################################################
```

```
      subroutine stiff4(a,b,nnode,coords,nelem,lnods)
c#####################################################################
      implicit real*8(a-h,o-z)
      dimension a(nnode,*),b(*),coords(*),lnods(2,*)
      dimension astiff(2,2),c(2)
c
      do i = 1,nnode
          do j = 1,nnode
              a(i,j) = 0.D0
          end do
      end do
c
      do i = 1,nnode
          b(i) = 0.D0
      end do
c
      do i = 1,nelem
          x = coords(lnods(2,i))-coords(lnods(1,i))
          astiff(1,1) =  1.D0 / x
          astiff(1,2) = -1.D0 / x
          astiff(2,1) = -1.D0 / x
          astiff(2,2) =  1.D0 / x
          ip1 = lnods(1,i)
          ip2 = lnods(2,i)
          c(1) = x / 2.D0
          c(2) = x / 2.D0
c
          ip1 = lnods(1,i)
          ip2 = lnods(2,i)
          a(ip1,ip1) = a(ip1,ip1) + astiff(1,1)
          a(ip2,ip1) = a(ip2,ip1) + astiff(2,1)
          a(ip1,ip2) = a(ip1,ip2) + astiff(1,2)
          a(ip2,ip2) = a(ip2,ip2) + astiff(2,2)
          b(ip1) = b(ip1) + c(2)
          b(ip2) = b(ip2) + c(2)
      end do
c
      return
      end
c
```

misc3.f バージョン 3 と共通

gauss_ver4u.f 第 1 章と共通

bound2.f　第 1 章と共通

2.2.15　C コーディング例

C コーディング例を示すと以下のようになる.

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o stiff4.o misc3.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
#include <stdio.h>

/* file stiff4.c */
void stiff4(double *a, double *b, int nnode, double *coords,
            int nelem, int *lnods);

/* file misc3.c */
void datasize3(int *nnode, int *nelem, int *n_bc_given, int *n_bc_nonzero,
               int icase);
void datain3(int nnode,double *coords, int nelem, int *lnods,
             int n_bc_given, int *i_bc_given,int n_bc_nonzero,
             int *i_bc_nonzero,double *v_bc_nonzero,int icase);
void check_solution3(double *b,int nnode, double *coords,int icase);

/* file gauss_ver4u.c */
void gauss_ver4u(double *a, double *c, int n);

/* file bound2.c */
```

```
void bound2(double *a, double *c, int n, int n_bc_given,
            int *i_bc_given, int n_bc_nonzero,
            int *i_bc_nonzero, double *v_bc_nonzero);

main()
{
    double *a,*b,*v_bc_nonzero,*coords;
    int n_bc_given,*i_bc_given,n_bc_nonzero,*i_bc_nonzero,
        nnode,nelem,*lnods,i,icase;

    icase = 2;

    datasize3(&nnode,&nelem,&n_bc_given,&n_bc_nonzero,icase);

    a  = (double *)malloc(sizeof(double)*(nnode)*(nnode));
    b  = (double *)malloc(sizeof(double)*(nnode));
    coords = (double *)malloc(sizeof(double)*(nnode));
    lnods  = (int *)malloc(sizeof(int)*(nelem)*2);
    i_bc_given   = (int *)malloc(sizeof(int)*(n_bc_given));
    i_bc_nonzero = (int *)malloc(sizeof(int)*(n_bc_nonzero));
    v_bc_nonzero = (double *)malloc(sizeof(double)*(n_bc_nonzero));

    datain3(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase);

    stiff4(a,b,nnode,coords,nelem,lnods);

    bound2(a,b,nnode,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);

    gauss_ver4u(a,b,nnode);

    check_solution3(b,nnode,coords,icase);
}
```

stiff4.c

```
void stiff4(double *arg_a, double *arg_b, int nnode, double *arg_coords,
            int nelem, int *arg_lnods)
{
    double *a, *b;
    double x;
    int i,j;
    double *coords;
    int *lnods;
```

```
    double *astiff,*c;
    int ip1,ip2;

    astiff = (double *)malloc(sizeof(double)*(2)*(2));
    c = (double *)malloc(sizeof(double)*(2));

    a = arg_a -1 -nnode;
    b = arg_b -1;

    astiff = astiff -1 -2;
    c = c -1;

    coords = arg_coords -1;
    lnods = arg_lnods -1 -2;

    for (i=1; i<=nnode; ++i){
        for (j=1; j<=nnode; ++j){
            a[i+j*nnode] = 0.0;
}
    }

    for (i=1; i<=nnode; ++i){
        b[i] = 0.0;
    }

    for (i=1; i<=nelem; ++i){
        x = coords[lnods[2+i*2]]-coords[lnods[1+i*2]];
        astiff[1+1*2] =  1.0 / x;
        astiff[1+2*2] = -1.0 / x;
        astiff[2+1*2] = -1.0 / x;
        astiff[2+2*2] =  1.0 / x;
        c[1] = x / 2.0;
        c[2] = x / 2.0;

        ip1 = lnods[1+i*2];
        ip2 = lnods[2+i*2];
        a[ip1+ip1*nnode] = a[ip1+ip1*nnode] + astiff[1+1*2];
        a[ip2+ip2*nnode] = a[ip2+ip2*nnode] + astiff[2+2*2];
        a[ip2+ip1*nnode] = a[ip2+ip1*nnode] + astiff[2+1*2];
        a[ip1+ip2*nnode] = a[ip1+ip2*nnode] + astiff[1+2*2];
        b[ip1] = b[ip1] + c[1];
        b[ip2] = b[ip2] + c[2];
    }
    astiff = astiff +1 +2;
    c = c +1;
```

```
    free(astiff);
    free(c);
}
```

misc3.c バージョン 3 と共通

gauss_ver4u.c 第 1 章と共通

bound2.c 第 1 章と共通

2.2.16 有限要素解析コードの基本型

バージョン 4 までで き上がったプログラムは, 図 2.2 のような構造になっている.

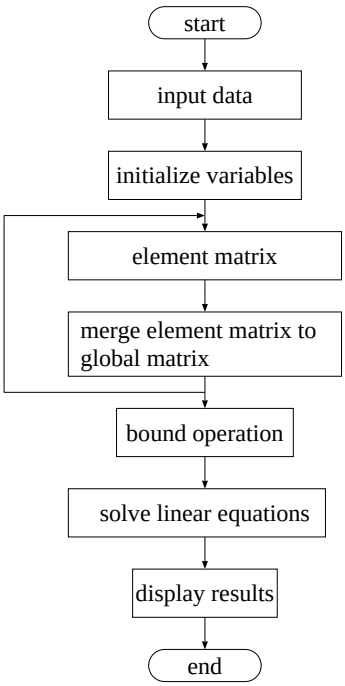


図 2.2: バージョン 4 のプログラムの構造

この構造は線形有限要素法解析コードならすべて同一であり, 解析対象により要素マト

リックスが変更され, 規模が大きくなると, 連立 1 次方程式の解法の効率化が必要なため, 別に前処理等が加わるだけである. また動解析や非線形解析のためのプログラムも, これを元に拡張されたような構造になっている.

2.3　高次有限要素補間関数と数値積分

2.1 節で示した例では, 関数 u の近似として区分線形関数 (ようするに折れ線) を用いた. ここでは近似の精度を上げるため区分 2 次関数を用いることにする.

2.3.1　高次補間

まず u の定義域 $[0, a]$ を n 個の重なりのない区間 $I_i (i = 1 \sim n)$ に分割する. 2 次関数を一意に定めるためには 3 点必要なので, 図 2.3 のように $2n + 1$ 個の節点をとる.

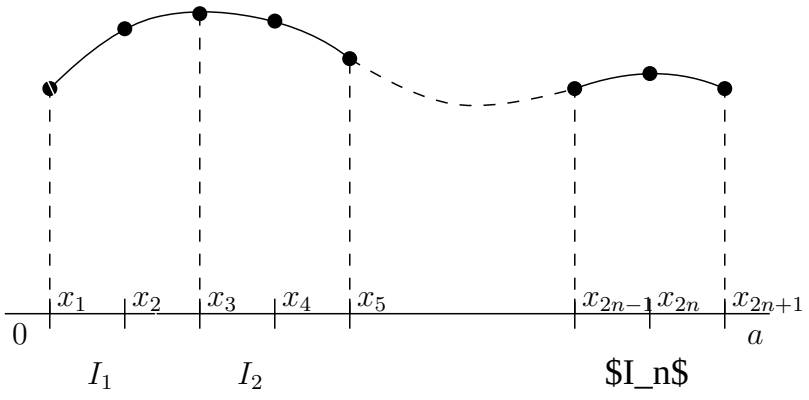


図 2.3: 2 次関数による補間

区間 I_i に含まれる節点は x_{2i-1}, x_{2i}, x_{2i+1} であるが, これをそれぞれ $x_{(i,1)}$, $x_{(i,3)}$, $x_{(i,2)}$

と表す. この時, 区間 $I_i = [x_{(i,1)}, x_{(i,2)}]$ と表すことができ, 2.1 節で示した区分線形の場合と同じ表記を用いることができる. 式 (2.14) の積分の総和表記も同様である. 再記すれば以下のようになる.

$$\sum_{i=1}^{n} \int_{x_{(i,1)}}^{x_{(i,2)}} \frac{du}{dx}\frac{dv}{dx}\, dx = \sum_{i=1}^{n} \int_{x_{(i,1)}}^{x_{(i,2)}} f \cdot v\, dx \tag{2.41}$$

それぞれの積分区間で, $x_{(i,1)}$ が -1, $x_{(i,2)}$ が 1 に対応するように座標変換を行なう. このときパラメータ $r\,(-1 \le r \le 1)$ を用いると, $x \in [x_{(i,1)}, x_{(i,2)}]$ は以下の形式で表される.

$$x = N^{(1)}\, x_{(i,1)} + N^{(2)}\, x_{(i,2)} + N^{(3)}\, x_{(i,3)} \tag{2.42}$$

ただし

$$N^{(1)} = -\frac{1}{2}\, r\,(1 - r) \tag{2.43}$$

$$N^{(2)} = \frac{1}{2}\, r\,(1 + r) \tag{2.44}$$

$$N^{(3)} = 1 - r^2 \tag{2.45}$$

u についても同様にパラメータ $r\,(-1 \le r \le 1)$ を用いて以下の形式で表す. ただし, $u_{(i,1)} = u_{2i-1}$, $u_{(i,2)} = u_{2i+1}$, $u_{(i,3)} = u_{2i}$ である.

$$u = N^{(1)} u_{(i,1)} + N^{(2)} u_{(i,2)} + N^{(3)} u_{(i,3)} \tag{2.46}$$

式 (2.41) 左辺の被積分関数には u の x に関する微分が含まれているが, 微分の連鎖則を用いることにより以下のように求められる.

$$\begin{aligned} \frac{du}{dx} &= \frac{dN^{(1)}}{dx}\, u_{(i,1)} + \frac{dN^{(2)}}{dx}\, u_{(i,2)} + \frac{dN^{(3)}}{dx}\, u_{(i,3)} \\ &= \frac{dN^{(1)}}{dr}\frac{dr}{dx}\, u_{(i,1)} + \frac{dN^{(2)}}{dr}\frac{dr}{dx}\, u_{(i,2)} + \frac{dN^{(3)}}{dr}\frac{dr}{dx}\, u_{(i,3)} \end{aligned} \tag{2.47}$$

$\dfrac{dr}{dx}$ については式 (2.42)〜(2.45) を用いて以下のようにインバースとして求められる.

$$\frac{dx}{dr} = \frac{dN^{(1)}}{dr}\, x_{(i,1)} + \frac{dN^{(2)}}{dr}\, x_{(i,2)} + \frac{dN^{(3)}}{dr}\, x_{(i,3)} \tag{2.48}$$

各積分区間 $[x_{(i,1)}, x_{(i,2)}]$ に対応して $J_{(i)} = \left|\dfrac{dx}{dr}\right|$ とすれば, 式 (2.41) は以下のようになる.

$$\sum_{i=1}^{n}\int_{-1}^{1}\{v_{(i,1)}\, v_{(i,2)}\, v_{(i,3)}\}\begin{bmatrix} \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(1)}}{dx} & \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(2)}}{dx} & \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(3)}}{dx} \\ \dfrac{dN^{(2)}}{dx}\dfrac{dN^{(1)}}{dx} & \dfrac{dN^{(2)}}{dx}\dfrac{dN^{(2)}}{dx} & \dfrac{dN^{(2)}}{dx}\dfrac{dN^{(3)}}{dx} \\ \dfrac{dN^{(3)}}{dx}\dfrac{dN^{(1)}}{dx} & \dfrac{dN^{(3)}}{dx}\dfrac{dN^{(2)}}{dx} & \dfrac{dN^{(3)}}{dx}\dfrac{dN^{(3)}}{dx} \end{bmatrix}\begin{Bmatrix} u_{(i,1)} \\ u_{(i,2)} \\ u_{(i,3)} \end{Bmatrix} J_{(i)}\, dr$$
$$= \sum_{i=1}^{n}\int_{-1}^{1}\{v_{(i,1)}\, v_{(i,2)}\, v_{(i,3)}\}\begin{Bmatrix} N^{(1)} \\ N^{(2)} \\ N^{(3)} \end{Bmatrix} f\, J_{(i)}\, dr \tag{2.49}$$

$v_{(i,1)}$, $v_{(i,2)}$, $v_{(i,3)}$, $u_{(i,1)}$, $u_{(i,2)}$, $u_{(i,3)}$, は節点での値であり r については定数であるから積分の外に出すことができる. 下式によって $[K^{(i)}]$, $\{F^{(i)}\}$ を定義する.

$$[K^{(i)}] = \begin{bmatrix} K_{11}^{(i)} & K_{12}^{(i)} & K_{13}^{(i)} \\ K_{21}^{(i)} & K_{22}^{(i)} & K_{23}^{(i)} \\ K_{31}^{(i)} & K_{32}^{(i)} & K_{33}^{(i)} \end{bmatrix} = \int_{-1}^{1}\begin{bmatrix} \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(1)}}{dx} & \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(2)}}{dx} & \dfrac{dN^{(1)}}{dx}\dfrac{dN^{(3)}}{dx} \\ \dfrac{dN^{(2)}}{dx}\dfrac{dN^{(1)}}{dx} & \dfrac{dN^{(2)}}{dx}\dfrac{dN^{(2)}}{dx} & \dfrac{dN^{(2)}}{dx}\dfrac{dN^{(3)}}{dx} \\ \dfrac{dN^{(3)}}{dx}\dfrac{dN^{(1)}}{dx} & \dfrac{dN^{(3)}}{dx}\dfrac{dN^{(2)}}{dx} & \dfrac{dN^{(3)}}{dx}\dfrac{dN^{(3)}}{dx} \end{bmatrix} J_{(i)}\, dr \tag{2.50}$$

$$\{F^{(i)}\} = \begin{Bmatrix} F_1^{(i)} \\ F_2^{(i)} \\ F_3^{(i)} \end{Bmatrix} = \int_{-1}^{1}\begin{Bmatrix} N^{(1)} \\ N^{(2)} \\ N^{(3)} \end{Bmatrix} f\, J_{(i)}\, dr \tag{2.51}$$

式 (2.49) の両辺を $\sum$ を用いずに書き下す. ただし式 (2.49) では要素毎に $u_{(i,1)}$, $u_{(i,2)}$, $u_{(i,3)}$, $v_{(i,1)}$, $v_{(i,2)}$, $v_{(i,3)}$, と記述されている u_i, v_i については節点番号の順に 1 列に並べて以下の形式に表す.

$$\{v_1\, v_2 \ldots v_n\}\begin{bmatrix} K_{11} & K_{12} & \ldots & K_{1n} \\ K_{21} & K_{22} & & K_{2n} \\ \vdots & & & \vdots \\ K_{n1} & K_{n2} & \ldots & K_{nn} \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{Bmatrix} = \{v_1\, v_2 \ldots v_n\}\begin{Bmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{Bmatrix} \tag{2.52}$$

式 (2.52) を記述の簡略化のために $\{v\}^T[K]\{u\} = \{v\}^T\{F\}$ と表す.

要素マトリックスを重ね合わせた $n \times n$ 次元のマトリックス $[K]$ を全体マトリックス等と呼ぶ. $[K]$ を成分で書けば以下のようになる.

$$[K] = \begin{bmatrix} K_{11}^{(1)} & K_{13}^{(1)} & K_{12}^{(1)} & & & \\ K_{31}^{(1)} & K_{33}^{(1)} & K_{32}^{(1)} & & & \\ K_{21}^{(1)} & K_{23}^{(1)} & K_{22}^{(1)} + K_{11}^{(2)} & K_{13}^{(2)} & K_{12}^{(2)} & \\ & & K_{31}^{(2)} & K_{33}^{(2)} & K_{32}^{(2)} & \\ & & K_{21}^{(2)} & K_{23}^{(2)} & K_{22}^{(2)} + K_{11}^{(3)} & \\ & & & & & \ddots \end{bmatrix} \tag{2.53}$$

また, $\{F\}$ を成分で書けば以下のようになる.

$$\{F\} = \begin{Bmatrix} F_1^{(1)} \\ F_3^{(1)} \\ F_2^{(1)} + F_1^{(2)} \\ F_3^{(2)} \\ F_2^{(2)} + F_1^{(3)} \\ \vdots \end{Bmatrix} \tag{2.54}$$

以上により得られた連立 1 次方程式 $[K]\{u\} = \{F\}$ を解けば, 近似解が得られる.

2.3.2 課題

$a = 1$, $f(x) = 1$ $([0, 1])$ として, 式 (2.1) の近似解を有限要素法により求めよ. ただし補間関数は 1 次, 2 次が選択できるようにすること.

まず解析的に積分を行なった解は, 以下のようになる.

$$u = \frac{1}{2}\, x(1 - x) \tag{2.55}$$

次に剛性マトリックスのコーディングを行なう準備をする. 式 (2.48) に従い $\dfrac{dx}{dr}$ を計算す

る. 式 (2.43)∼(2.45) より $\dfrac{dN^{(i)}}{dr}$ は以下のようになる.

$$\frac{dN^{(1)}}{dr} = r - \frac{1}{2} \tag{2.56}$$

$$\frac{dN^{(2)}}{dr} = r + \frac{1}{2} \tag{2.57}$$

$$\frac{dN^{(3)}}{dr} = -2\,r \tag{2.58}$$

これを用いると $\dfrac{dx}{dr}$ は以下のようになる.

$$\frac{dx}{dr} = (r - \frac{1}{2})\,x_{(i,1)} + (r + \frac{1}{2})\,x_{(i,2)} - 2\,r\,x_{(i,3)} \tag{2.59}$$

もし, $x_{(i,3)}$ が $x_{(i,1)}$, $x_{(i,2)}$ の中点であれば, 以下のように単純化される.

$$\begin{aligned}\frac{dx}{dr} &= (r - \frac{1}{2})\,x_{(i,1)} + (r + \frac{1}{2})\,x_{(i,2)} - 2\,r\frac{x_{(i,1)} + x_{(i,2)}}{2} \\ &= \frac{x_{(i,2)} - x_{(i,1)}}{2}\end{aligned} \tag{2.60}$$

これよりヤコビアン $J_{(i)}$ は以下のようになる.

$$J_{(i)} = \frac{x_{(i,2)} - x_{(i,1)}}{2} \tag{2.61}$$

次に式 (2.51) は $f(x) = 1$ なので, 以下のようになる.

$$\begin{aligned}F_1^{(i)} &= \int_{-1}^{1} -\frac{1}{2}\,r\,(1-r)\frac{x_{(i,2)} - x_{(i,1)}}{2}\,dr \\ &= \frac{1}{6}(x_{(i,2)} - x_{(i,1)})\end{aligned} \tag{2.62}$$

$$\begin{aligned}F_2^{(i)} &= \int_{-1}^{1} \frac{1}{2}\,r\,(1+r)\frac{x_{(i,2)} - x_{(i,1)}}{2}\,dr \\ &= \frac{1}{6}(x_{(i,2)} - x_{(i,1)})\end{aligned} \tag{2.63}$$

$$\begin{aligned}F_3^{(i)} &= \int_{-1}^{1} (1-r^2)\frac{x_{(i,2)} - x_{(i,1)}}{2}\,dr \\ &= \frac{2}{3}(x_{(i,2)} - x_{(i,1)})\end{aligned} \tag{2.64}$$

2.3.3 数値積分

2.1 節では線形の補間関数を用いたので被積分関数が定数になり積分を行なう必要がなかったが, ここで用いる 2 次の補間関数の場合, 何らかの方法で積分を行う必要がある. 一般の解析対象では, 被積分関数が解析的に積分できない場合がほとんどなので, 有限要素コードでは数値積分を用いるのが普通である. 詳細については後述するとして, ここでは代表的な Gauss 積分の方法のみ説明する.

1 次元の関数 $f(x)$ の Gauss 積分は, 後述するような方法により求められたサンプリング点 x_i と対応する重み w_i を用いて下式のように表される.

$$\int_{-1}^{1} f(x)\,dx = \sum_i w_i\, f(x_i) \tag{2.65}$$

サンプリング点と重みの値は表 2.4 のようになっている.

サンプリング点数	x_i	w_i
1	0	2
2	±0.57735 02691 89626	1
3	±0.77459 66692 41483	0.55555 55555 55556
	0	0.88888 88888 88889
4	±0.86113 63115 94053	0.34785 48451 37454
	±0.33998 10435 84856	0.65214 51548 62546
5	±0.90617 98459 38664	0.23692 68850 56189
	±0.53846 93101 05683	0.47862 86704 99366
	0	0.56888 88888 88889
6	±0.93246 95142 03152	0.17132 44923 79170
	±0.66120 93864 66265	0.36076 15730 48139
	±0.23861 91860 83197	0.46791 39345 72691

表 2.4: サンプリング点と重みの値

$f(x)$ が定数ならば 1 点以上, 線形なら 2 点以上, 2 次関数なら 3 点以上の Gauss 積

分を用いる. 当然ながら, サンプリング点数が十分であれば数値積分の結果は変わらない. 有限要素法の場合は被積分関数に微分やヤコビアンなどが含まれるため, 単純に次数では表せないが, 1 次の補間関数なら 2 点, 2 次の補間関数なら 3 点の Gauss 積分を用いるのが一般的である.

さて, 今回の課題では補間関数を 1 次にするか 2 次にするかを選択できなくてはいけない. 通常, 有限要素法では数値積分のループの構成が同じなら一つのサブルーチンにする. つまり, 今回のように 1 重のループしかない場合は, 一つのサブルーチンで if 文などを使って 1 要素あたりの節点数の変化に対応する.

以上を元に先の有限要素解析コードのプロトタイプを拡張する.

2.3.4　バージョン 5 — 数値積分の導入

要素マトリックスを作るプログラムの構造は, 線形・非線形の違いや, 解析対象の違いに関わらず, ほぼ数値積分のループの構成で決まる. 例えば, 今回コーディングしているような線分状の要素ならば線積分で 1 重のループ, 平面上の四角形の要素ならば 2 重のループになる. 通常はプログラムを見やすくするために, このように構成の異なるループを一つのサブルーチン内に混在させず, 別サブルーチンとしてコーディングする. 逆に同じループの構成ならば, 数値積分のサンプリング点数に関わらず一つにまとめるのが普通である. ただし, プログラムの速さを追求する際にはこの限りでない.

ここでは, 先のバージョン 4 のプログラムで要素マトリックスを作る部分を別サブルーチンにし, 式 (2.23) を数値積分するように変更する. この時, $J_{(i)}$ についても式 (2.21) をもとに計算するようにしておく. また, サブルーチンの入力変数と出力変数をわかりやすくするため, グローバル変数は使用しない. Gauss 積分のサンプリング点数は入力ファイルから読み込み, 変数名は `nint` とし, `nint` を $1 \sim 4$ まで変化させて, すべて同じ解になることを確認する. 補間関数が 1 次の場合の要素マトリックス生成サブルーチンは, 例え

ば以下のようなコーディングになる.

for $i = 1 \sim$ `nelem`

(clear A_{ij})

for $j = 1 \sim$ `nint`

$r =$ (sampling point j)

$dN/dr_1 = -0.5$

$dN/dr_2 = 0.5$

$J = \sum_k dN/dr_k \cdot$ `coords`(`lnods`(k, i))

$\det J = J$

$J^{-1} = 1/J$

for $k = 1 \sim 2$

$dN/dx_k = J^{-1} \cdot dN/dr_k$

end for

$w =$ (weight j)

for $k = 1 \sim 2$

for $l = 1 \sim 2$

$A_{kl} = A_{kl} + \det J \cdot w \cdot dN/dx_k \cdot dN/dx_l$

end for

end for

end for

end for

2.3.5　入力ファイル例

入力ファイルは fortran, C とも共通である.

case1.dat

```
11
 1  1  0.0
 2  0  0.1
 3  0  0.2
 4  0  0.3
 5  0  0.4
 6  0  0.5
 7  0  0.6
 8  0  0.7
 9  0  0.8
10  0  0.9
11  1  1.0
10
 1  1  2
 2  2  3
 3  3  4
 4  4  5
 5  5  6
 6  6  7
 7  7  8
 8  8  9
 9  9 10
10 10 11
1
 1  1  1.0
4
```

case2.dat

```
11
 1  1  0.0
 2  0  0.1
 3  0  0.2
 4  0  0.3
 5  0  0.4
 6  0  0.5
 7  0  0.6
 8  0  0.7
```

```
 9  0  0.8
10  0  0.9
11  1  1.0
10
 1  1  2
 2  2  3
 3  3  4
 4  4  5
 5  5  6
 6  6  7
 7  7  8
 8  8  9
 9  9 10
10 10 11
2
 1  1  1.0
 2 11  2.0
4
```

2.3.6 fortran コーディング例

fortran コーディング例を示すと以下のようになる.

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o stiff5.o misc5.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
      implicit real*8(a-h,o-z)
```

```
      parameter (MXNODE = 100)
      parameter (MXELEM = 100)
      dimension a(MXNODE*MXNODE),b(MXNODE)
      dimension coords(MXNODE),lnods(2,MXELEM)
      dimension i_bc_given(MXNODE),i_bc_nonzero(MXNODE),
     1          v_bc_nonzero(MXNODE)
      dimension astiff(2,2),c(2)
c
      icase = 2
c
      call datain5(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase,nint)
c
      call stiff5(a,b,nnode,coords,nelem,lnods,astiff,c,nint)
c
      call bound2(a,b,nnode,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c
      call gauss_ver4u(a,b,nnode)
c
      call check_solution3(b,nnode,coords,icase)
c
      stop
      end
c
```

stiff5.f

```
c######################################################################
      subroutine stiff5(a,b,nnode,coords,nelem,lnods,astiff,c,nint)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(nnode,*),b(*),coords(*),lnods(2,*)
      dimension astiff(2,*),c(*)
c
      do i = 1,nnode
          do j = 1,nnode
              a(i,j) = 0.D0
          end do
      end do
c
      do i = 1,nnode
          b(i) = 0.D0
      end do
c
```

```
      do i = 1,nelem
          call element(coords,lnods(1,i),astiff,nint)
c
          x = coords(lnods(2,i))-coords(lnods(1,i))
          c(1) = x / 2.D0
          c(2) = x / 2.D0
c
          ip1 = lnods(1,i)
          ip2 = lnods(2,i)
          a(ip1,ip1) = a(ip1,ip1) + astiff(1,1)
          a(ip2,ip1) = a(ip2,ip1) + astiff(2,1)
          a(ip1,ip2) = a(ip1,ip2) + astiff(1,2)
          a(ip2,ip2) = a(ip2,ip2) + astiff(2,2)
          b(ip1) = b(ip1) + c(2)
          b(ip2) = b(ip2) + c(2)
      end do
c
      return
      end
c
c######################################################################
      subroutine element(coords,lnods,astiff,nint)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(*),lnods(*),astiff(2,*)
      dimension gsp(4,4),wgh(4,4),dndx(2),dndr(2)
c
      data gsp /  0.D0           , 0.D0           , 0.D0,           0.D0,
     1           -0.577350269189626D0, 0.577350269189626D0, 0.D0,0.D0,
     1           -0.774596669241483D0, 0.D0, 0.774596669241483D0,0.D0,
     1           -0.861136311594053D0,-0.339981043584856D0,
     1            0.339981043584856D0, 0.861136311594053D0 /
      data wgh /  2.D0           , 0.D0           , 0.D0,           0.D0,
     1            1.D0           , 1.D0           , 0.D0,           0.D0,
     1            0.555555555555556D0, 0.888888888888889D0,
     1            0.555555555555556D0, 0.D0,
     1            0.347854845137454D0, 0.652145154862546D0,
     1            0.652145154862546D0, 0.347854845137454D0 /
c
      do j = 1,2
          do i = 1,2
              astiff(i,j) = 0.D0
          end do
      end do
c
```

```
      do ir = 1,nint
c
          r = gsp(ir,nint)
c
          dndr(1) = -0.5D0
          dndr(2) =  0.5D0
c
          ajacob = 0.D0
          do i = 1,2
              ajacob = ajacob + dndr(i) * coords(lnods(i))
          end do
          detjac = ajacob
          ajainv = 1.D0 / ajacob
c
          do i = 1,2
              dndx(i) = dndr(i) * ajainv
          end do
c
          detwei = detjac * wgh(ir,nint)
c
          do j = 1,2
              do i = 1,2
                  astiff(i,j) = astiff(i,j) + detwei * dndx(i) * dndx(j)
              end do
          end do
      end do
c
      return
      end
c
```

misc5.f

```
c######################################################################
      subroutine datain5(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase,nint)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(*),lnods(2,*),i_bc_given(*),i_bc_nonzero(*),
     1          v_bc_nonzero(*)
c
      if (icase .eq. 1) open(10,file='case1.dat')
      if (icase .eq. 2) open(10,file='case2.dat')
c
      n_bc_given = 0
```

```
      read(10,*) nnode
      do i = 1, nnode
          read(10,*) itemp, ibc,coords(i)
          if (ibc .eq. 1) then
              n_bc_given = n_bc_given + 1
              i_bc_given(n_bc_given) = i
          end if
      end do
c
      read(10,*) nelem
      do i = 1,nelem
          read(10,*) itemp, (lnods(j,i),j=1,2)
      end do
c
      read(10,*) n_bc_nonzero
      do i = 1,n_bc_nonzero
          read(10,*) itemp, i_bc_nonzero(i),v_bc_nonzero(i)
      end do
c
      read(10,*) nint
c
      close(10)
c
      return
      end
c
c#######################################################################
      subroutine check_solution3(b,nnode,coords,icase)
c#######################################################################
      implicit real*8(a-h,o-z)
      dimension b(*),coords(*)
c
      do i = 1,nnode
          x = coords(i)
          if (icase .eq. 1) u = -0.5D0 * x*x - 0.5D0 * x + 1
          if (icase .eq. 2) u = -0.5D0 * x*x + 1.5D0 * x + 1
          write(*,1000) b(i), u
      end do
 1000 format(2x,e13.6,2x,e13.6)
c
      return
      end
c
```

gauss_ver4u.f　第 1 章と共通

bound2.f　第 1 章と共通

2.3.7　C コーディング例

C コーディング例を示すと以下のようになる.

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o stiff5.o misc5.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
#include <stdio.h>

/* file stiff5.c */
void stiff5(double *a, double *b, int nnode, double *coords,
            int nelem, int *lnods, double *astiff, double *c, int nint);

/* file misc5.c */
void datasize3(int *nnode, int *nelem, int *n_bc_given, int *n_bc_nonzero,
               int icase);
void datain5(int nnode, double *coords, int nelem, int *lnods,
             int n_bc_given, int *i_bc_given,int n_bc_nonzero,
             int *i_bc_nonzero,double *v_bc_nonzero,int icase,int *nint);
void check_solution3(double *b,int nnode, double *coords,int icase);

/* file gauss_ver4u.c */
```

```
void gauss_ver4u(double *a, double *c, int n);

/* file bound2.c */
void bound2(double *a, double *c, int n, int n_bc_given,
            int *i_bc_given, int n_bc_nonzero,
            int *i_bc_nonzero, double *v_bc_nonzero);


main()
{
    double *a,*b,*v_bc_nonzero;
    int n_bc_given,*i_bc_given,n_bc_nonzero,*i_bc_nonzero;
    double *coords;

    int nnode,nelem,*lnods,i,icase;
    double *astiff,*c;
    int nint;

    icase = 2;

    datasize3(&nnode,&nelem,&n_bc_given,&n_bc_nonzero,icase);

    a  = (double *)malloc(sizeof(double)*(nnode)*(nnode));
    b  = (double *)malloc(sizeof(double)*(nnode));
    coords = (double *)malloc(sizeof(double)*(nnode));
    lnods  = (int *)malloc(sizeof(int)*(nelem)*2);
    i_bc_given   = (int *)malloc(sizeof(int)*(n_bc_given));
    i_bc_nonzero = (int *)malloc(sizeof(int)*(n_bc_nonzero));
    v_bc_nonzero = (double *)malloc(sizeof(double)*(n_bc_nonzero));
    astiff = (double *)malloc(sizeof(double)*(2)*(2));
    c      = (double *)malloc(sizeof(double)*(2));

    datain5(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase,&nint);

    stiff5(a,b,nnode,coords,nelem,lnods,astiff,c,nint);

    bound2(a,b,nnode,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);

    gauss_ver4u(a,b,nnode);

    check_solution3(b,nnode,coords,icase);
}
```

stiff5.c

```
void element(double *coords, int *lnods,double *astiff, int nint);

void stiff5(double *arg_a, double *arg_b, int nnode, double *arg_coords,
            int nelem, int *arg_lnods,
            double *arg_astiff, double *arg_c, int nint)
{
    double *a, *b;
    double x;
    int i,j;
    double *coords;
    int *lnods;
    int ip1,ip2;
    double *astiff, *c;

    a = arg_a -1 -nnode;
    b = arg_b -1;

    coords = arg_coords -1;
    lnods = arg_lnods -1 -2;

    astiff = arg_astiff -1 -2;
    c = arg_c -1;

    for (i=1; i<=nnode; ++i){
        for (j=1; j<=nnode; ++j){
            a[i+j*nnode] = 0.0;
}
    }

    for (i=1; i<=nnode; ++i){
        b[i] = 0.0;
    }

    for (i=1; i<=nelem; ++i){
        element(coords,&lnods[1+i*2],astiff,nint);
        x = coords[lnods[2+i*2]]-coords[lnods[1+i*2]];
        c[1] = x / 2.0;
        c[2] = x / 2.0;

        ip1 = lnods[1+i*2];
        ip2 = lnods[2+i*2];
        a[ip1+ip1*nnode] = a[ip1+ip1*nnode] + astiff[1+1*2];
        a[ip2+ip2*nnode] = a[ip2+ip2*nnode] + astiff[2+2*2];
        a[ip2+ip1*nnode] = a[ip2+ip1*nnode] + astiff[2+1*2];
```

```
        a[ip1+ip2*nnode] = a[ip1+ip2*nnode] + astiff[1+2*2];
        b[ip1] = b[ip1] + c[1];
        b[ip2] = b[ip2] + c[2];
    }

    free(arg_astiff);
    free(arg_c);
}

void element(double *coords, int *arg_lnods,double *astiff, int nint)
{
    int *lnods;
    double x,r,ajacob,detjac,ajainv,detwei,
           *gsp,*wgh,*dndx,*dndr;
    int i,j,ir;

    static double arg_gsp[16] = { 0.0, 0.0, 0.0, 0.0,
                 -0.577350269189626, 0.577350269189626, 0.0, 0.0,
                 -0.774596669241483, 0.0, 0.774596669241483, 0.0,
                 -0.861136311594053,-0.339981043584856,
                  0.339981043584856, 0.861136311594053 };

    static double arg_wgh[16] = {2.0,0.0,0.0,0.0,
                  1.0,1.0,0.0,0.0,
                  0.555555555555556,0.888888888888889,0.555555555555556,0.0,
                  0.347854845137454,0.652145154862546,
                  0.652145154862546,0.347854845137454 };

    lnods  = arg_lnods  -1;

    dndx = (double *)malloc(sizeof(double)*(2));
    dndr = (double *)malloc(sizeof(double)*(2));

    gsp  = arg_gsp  -1 -4;
    wgh  = arg_wgh  -1 -4;
    dndx = dndx -1;
    dndr = dndr -1;

    for (i = 1; i <= 2; ++i){
        for (j = 1; j <= 2; ++j){
            astiff[i+j*2] = 0.0;
        }
    }

    for (ir = 1; ir <= nint; ++ir){
```

```
        r = gsp[ir + nint*4];

        dndr[1] = -0.5;
        dndr[2] =  0.5;

        ajacob = 0.0;
        for (i = 1; i <= 2; ++i){
            ajacob = ajacob + dndr[i] * coords[lnods[i]];
        }
        detjac = ajacob;
        ajainv = 1.0 / ajacob;

        for (i = 1; i <= 2; ++i){
            dndx[i] = dndr[i] * ajainv;
        }

        detwei = detjac * wgh[ir+nint*4];

        for (i = 1; i <= 2; ++i){
            for (j = 1; j <= 2; ++j){
                astiff[i+j*2] = astiff[i+j*2] + detwei * dndx[i] * dndx[j];
            }
        }
    }
    dndx = dndx +1;
    dndr = dndr +1;
    free(dndx);
    free(dndr);
}
```

misc5.c

```
#include <stdio.h>
void datasize3(int *nnode, int *nelem, int *n_bc_given, int *n_bc_nonzero,
               int icase)
{
    FILE *fp;
    double coords;
    int i,inode,ielem,lnods1,lnods2,ibc;

    if (icase == 1){
        if ((fp = fopen("case1.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
```

```
    }
    if (icase == 2){
        if ((fp = fopen("case2.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }

    *n_bc_given = 0;
    fscanf(fp,"%d",nnode);
    for (i = 1; i <= *nnode; ++i){
        fscanf(fp,"%d %d %lf", &inode, &ibc, &coords);
        if (ibc == 1){
            *n_bc_given = *n_bc_given + 1;
        }
    }

    fscanf(fp,"%d",nelem);
    for (i = 1; i <= *nelem; ++i){
        fscanf(fp,"%d %d %d",&ielem,&lnods1,&lnods2);
    }

    fscanf(fp,"%d",n_bc_nonzero);

    fclose(fp);
}

void datain5(int nnode,double *arg_coords, int nelem, int *arg_lnods,
             int n_bc_given, int *arg_i_bc_given,int n_bc_nonzero,
             int *arg_i_bc_nonzero,double *arg_v_bc_nonzero,int icase,
             int *nint)
{
    FILE *fp;
    int i,inode,ielem,ibc,idum,*lnods, *i_bc_given, *i_bc_nonzero;
    double *coords, *v_bc_nonzero;

    coords = arg_coords -1;
    lnods = arg_lnods -1-2;
    i_bc_given = arg_i_bc_given -1;
    i_bc_nonzero = arg_i_bc_nonzero -1;
    v_bc_nonzero = arg_v_bc_nonzero -1;

    if (icase == 1){
        if ((fp = fopen("case1.dat","r")) == NULL){
            printf("can't open file\n");
```

```
            exit(1);
        }
    }
    if (icase == 2){
        if ((fp = fopen("case2.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }

    i = 0;
    fscanf(fp,"%d",&idum);
    for (inode = 1; inode <= nnode; ++inode){
        fscanf(fp,"%d %d %lf", &idum, &ibc, &coords[inode]);
        if (ibc == 1){
            i = i + 1;
            i_bc_given[i] = inode;
        }
    }

    fscanf(fp,"%d",&idum);
    for (ielem = 1; ielem <= nelem; ++ielem){
        fscanf(fp,"%d %d %d",&idum,&lnods[1+ielem*2],&lnods[2+ielem*2]);
    }

    fscanf(fp,"%d",&idum);
    for (i = 1; i <= n_bc_nonzero; ++i){
        fscanf(fp,"%d %d %lf",&ibc,&i_bc_nonzero[i],&v_bc_nonzero[i]);
    }

    fscanf(fp,"%d",nint);

    fclose(fp);
}

void check_solution3(double *arg_b,int nnode, double *arg_coords, int icase)
{
    double *b,*coords;
    int i;
    double x,u;

    b = arg_b -1;
    coords = arg_coords -1;

    for (i=1; i<=nnode; ++i){
```

```
        x = coords[i];
        if (icase == 1) u = -0.5 * x*x - 0.5 * x + 1.0;
        if (icase == 2) u = -0.5 * x*x + 1.5 * x + 1.0;
        printf("  % 10.3E  % 10.3E\n",b[i],u);
    }
}
```

gauss_ver4u.c　第 1 章と共通

bound2.c　第 1 章と共通

2.3.8 バージョン 6 — 2 次の補間関数

これまで取り扱ってきた 1 次の補間関数は 2 節点であるのに対して, 2 次の補間関数は 3 節点で構成される. そのため, まず入力データのコネクティビティの部分の形式を表 2.5 のように変更する. また, 各要素の節点数の変数名は ntnoel(nelem) とする. ただし簡単のため各要素の中間節点は両端の節点の中点にとる.

5					nelem			
1	3	1	3	2	ntnoel(1)	lnods(1,1)	lnods(2,1)	lnods(3,1)
2	3	3	5	4	ntnoel(2)	lnods(1,2)	lnods(2,2)	lnods(3,2)
3	3	5	7	6	ntnoel(3)	lnods(1,3)	lnods(2,3)	lnods(3,3)
4	3	7	9	8	ntnoel(4)	lnods(1,4)	lnods(2,4)	lnods(3,4)
5	3	9	11	10	ntnoel(5)	lnods(1,5)	lnods(2,5)	lnods(3,5)

表 2.5: 3 節点でのコネクティビティの形式

また, 1 要素あたりの節点数をプログラム中で可変にするためには, merge も変更する必要がある. 例えば以下のようなコーディングになる.

for　$i = 1 \sim$ nelem

　call element(i)　　;; 要素マトリックス生成サブルーチン

for $j = 1 \sim \texttt{ntnoel}(i)$

$ip(j) = \texttt{lnods}(j, i)$

end for

for $j = 1 \sim \texttt{ntnoel}(i)$

for $k = 1 \sim \texttt{ntnoel}(i)$

$K_{ip(j),ip(k)} = K_{ip(j),ip(k)} + A_{jk}$

end for

end for

end for

2.3.9　入力ファイル例

入力ファイルは fortran, C とも共通である.

case11.dat 1 次補間

```
11
 1  1  0.0
 2  0  0.1
 3  0  0.2
 4  0  0.3
 5  0  0.4
 6  0  0.5
 7  0  0.6
 8  0  0.7
 9  0  0.8
10  0  0.9
11  1  1.0
10
 1  2   1  2
 2  2   2  3
 3  2   3  4
 4  2   4  5
 5  2   5  6
```

```
 6  2   6  7
 7  2   7  8
 8  2   8  9
 9  2   9 10
10  2  10 11
1
 1  1  1.0
4
```

case12.dat 2 次補間

```
11
 1  1  0.0
 2  0  0.1
 3  0  0.2
 4  0  0.3
 5  0  0.4
 6  0  0.5
 7  0  0.6
 8  0  0.7
 9  0  0.8
10  0  0.9
11  1  1.0
5
 1  3   1  3  2
 2  3   3  5  4
 3  3   5  7  6
 4  3   7  9  8
 5  3   9 11 10
1
 1  1  1.0
4
```

case21.dat 1 次補間

```
11
 1  1  0.0
 2  0  0.1
 3  0  0.2
 4  0  0.3
 5  0  0.4
 6  0  0.5
 7  0  0.6
 8  0  0.7
```

```
 9  0  0.8
10  0  0.9
11  1  1.0
10
 1  2  1  2
 2  2  2  3
 3  2  3  4
 4  2  4  5
 5  2  5  6
 6  2  6  7
 7  2  7  8
 8  2  8  9
 9  2  9 10
10  2 10 11
2
 1  1  1.0
 2 11  2.0
4
```

case22.dat 2 次補間

```
11
 1  1  0.0
 2  0  0.1
 3  0  0.2
 4  0  0.3
 5  0  0.4
 6  0  0.5
 7  0  0.6
 8  0  0.7
 9  0  0.8
10  0  0.9
11  1  1.0
5
 1  3   1  3  2
 2  3   3  5  4
 3  3   5  7  6
 4  3   7  9  8
 5  3   9 11 10
2
 1  1  1.0
 2 11  2.0
4
```

2.3.10 fortran コーディング例

fortran コーディング例を示すと以下のようになる.

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o stiff6.o misc6.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
      implicit real*8(a-h,o-z)
      parameter (MXNODE = 100)
      parameter (MXELEM = 100)
      dimension a(MXNODE*MXNODE),b(MXNODE)
      dimension coords(MXNODE),lnods(3,MXELEM)
      dimension i_bc_given(MXNODE),i_bc_nonzero(MXNODE),
     1          v_bc_nonzero(MXNODE)
      dimension astiff(3,3),c(3)
      dimension ntnoel(MXELEM)
c
      icase = 22
c
      call datain6(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase,nint,
     1            ntnoel)
c
      call stiff6(a,b,nnode,coords,nelem,lnods,astiff,c,nint,ntnoel)
c
      call bound2(a,b,nnode,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c
      call gauss_ver4u(a,b,nnode)
```

```
c
      call check_solution6(b,nnode,coords,icase)
c
      stop
      end
c
```

stiff5.f

```
c######################################################################
      subroutine stiff6(a,b,nnode,coords,nelem,lnods,astiff,c,nint,
     1                  ntnoel)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(nnode,*),b(*),coords(*),lnods(3,*)
      dimension astiff(*),c(*),ntnoel(*),ip(100)
c
      do i = 1,nnode
          do j = 1,nnode
              a(i,j) = 0.D0
          end do
      end do
c
      do i = 1,nnode
          b(i) = 0.D0
      end do
c
      do ielem = 1,nelem
          call element(coords,lnods(1,ielem),astiff,nint,ntnoel(ielem))
c
          call amerge(a,lnods(1,ielem),astiff,ntnoel(ielem),nnode)
c
          do i = 1,ntnoel(ielem)
             ip(i) = lnods(i,ielem)
          end do
c
          x = coords(lnods(2,ielem)) - coords(lnods(1,ielem))
          if (ntnoel(ielem) .eq. 2) then
              c(1) = x / 2.D0
              c(2) = x / 2.D0
          end if
          if (ntnoel(ielem) .eq. 3) then
              c(1) = x / 6.D0
              c(2) = x / 6.D0
              c(3) = x * 2.D0/ 3.D0
```

```
          end if
c
          do i = 1,ntnoel(ielem)
              b(ip(i)) = b(ip(i)) + c(i)
          end do
      end do
c
      return
      end
c
c######################################################################
      subroutine element(coords,lnods,astiff,nint,ntnoel)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(*),lnods(*),astiff(ntnoel,*)
      dimension gsp(4,4),wgh(4,4),dndx(3),dndr(3)
c
      data gsp /  0.D0          , 0.D0          , 0.D0,          0.D0,
     1           -0.577350269189626D0, 0.577350269189626D0, 0.D0,0.D0,
     1           -0.774596669241483D0, 0.D0, 0.774596669241483D0,0.D0,
     1           -0.861136311594053D0,-0.339981043584856D0,
     1            0.339981043584856D0, 0.861136311594053D0 /
      data wgh /  2.D0          , 0.D0          , 0.D0,          0.D0,
     1            1.D0          , 1.D0          , 0.D0,          0.D0,
     1            0.555555555555556D0, 0.888888888888889D0,
     1            0.555555555555556D0, 0.D0,
     1            0.347854845137454D0, 0.652145154862546D0,
     1            0.652145154862546D0, 0.347854845137454D0 /
c
      do j = 1,ntnoel
          do i = 1,ntnoel
              astiff(i,j) = 0.D0
          end do
      end do
c
      do ir = 1,nint
c
          r = gsp(ir,nint)
c
          if (ntnoel .eq. 2) then
              dndr(1) = -0.5D0
              dndr(2) =  0.5D0
          end if
          if (ntnoel .eq. 3) then
              dndr(1) = r - 0.5
```

```
            dndr(2) = r + 0.5
            dndr(3) = -2.D0 * r
          end if
c
          ajacob = 0.D0
          do i = 1,ntnoel
            ajacob = ajacob + dndr(i) * coords(lnods(i))
          end do
          detjac = ajacob
          ajainv = 1.D0 / ajacob
c
          do i = 1,ntnoel
            dndx(i) = dndr(i) * ajainv
          end do
c
          detwei = detjac * wgh(ir,nint)
c
          do j = 1,ntnoel
            do i = 1,ntnoel
              astiff(i,j) = astiff(i,j) + detwei * dndx(i) * dndx(j)
            end do
          end do
       end do
c
       return
       end
c
c######################################################################
       subroutine amerge(a,lnods,astiff,ntnoel,nnode)
c######################################################################
       implicit real*8(a-h,o-z)
       dimension a(*),lnods(*),astiff(ntnoel,*)
       dimension ip(100)
c
       do i = 1,ntnoel
          ip(i) = lnods(i)
       end do
c
       do j = 1,ntnoel
          do i = 1,ntnoel
            a((ip(j)-1)*nnode+ip(i)) =  a((ip(j)-1)*nnode+ip(i))
     1                 + astiff(i,j)
          end do
       end do
c
```

```
      return
      end
c
```

misc5.f

```
c######################################################################
      subroutine datain6(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase,nint,
     1            ntnoel)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(*),lnods(3,*),i_bc_given(*),i_bc_nonzero(*),
     1          v_bc_nonzero(*),ntnoel(*)
c
      if (icase .eq. 11) open(10,file='case11.dat')
      if (icase .eq. 21) open(10,file='case21.dat')
      if (icase .eq. 12) open(10,file='case12.dat')
      if (icase .eq. 22) open(10,file='case22.dat')
c
      n_bc_given = 0
      read(10,*) nnode
      do i = 1, nnode
          read(10,*) itemp, ibc,coords(i)
          if (ibc .eq. 1) then
              n_bc_given = n_bc_given + 1
              i_bc_given(n_bc_given) = i
          end if
      end do
c
      read(10,*) nelem
      do i = 1,nelem
          read(10,*) itemp, ntnoel(i),(lnods(j,i),j=1,ntnoel(i))
      end do
c
      read(10,*) n_bc_nonzero
      do i = 1,n_bc_nonzero
          read(10,*) itemp, i_bc_nonzero(i),v_bc_nonzero(i)
      end do
c
      read(10,*) nint
c
      close(10)
c
      return
```

```
      end
c
c######################################################################
      subroutine check_solution6(b,nnode,coords,icase)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension b(*),coords(*)
c
      do i = 1,nnode
          x = coords(i)
          if (icase .lt. 20) u = -0.5D0 * x*x - 0.5D0 * x + 1
          if (icase .gt. 20) u = -0.5D0 * x*x + 1.5D0 * x + 1
          write(*,1000) b(i), u
      end do
 1000 format(2x,e13.6,2x,e13.6)
c
      return
      end
c
```

gauss_ver4u.f　第 1 章と共通

bound2.f　第 1 章と共通

2.3.11　C コーディング例

C コーディング例を示すと以下のようになる.

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o stiff6.o misc6.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(CC) $(C_OPT) -o $(TARGET) $(OBJS)
```

```
.c.o:
        $(CC) -c $(C_OPT) $<
```

main.c

```
#include <stdio.h>

/* file stiff6.c */
void stiff6(double *a, double *b, int nnode, double *coords,int nelem,
            int *lnods,double *astiff, double *c, int nint,
            int *ntnoel, int MXNOEL);

/* file misc6.c */
void datasize6(int *nnode,int *nelem, int *n_bc_given, int *n_bc_nonzero,
               int icase, int *MXNOEL);
void datain6(int nnode,double *coords,int nelem,int *lnods,
             int n_bc_given, int *i_bc_given,int n_bc_nonzero,
             int *i_bc_nonzero,double *v_bc_nonzero,int icase,
             int *nint, int *ntnoel, int MXNOEL);
void check_solution6(double *b,int nnode, double *coords, int icase);

/* file gauss_ver4u.c */
void gauss_ver4u(double *a, double *c, int n);

/* file bound2.c */
void bound2(double *a, double *c, int n, int n_bc_given,
            int *i_bc_given, int n_bc_nonzero,
            int *i_bc_nonzero, double *v_bc_nonzero);

main()
{
    double *a,*b,*v_bc_nonzero;
    int n_bc_given,*i_bc_given,n_bc_nonzero,*i_bc_nonzero;
    double *coords;

    int nnode,nelem,*lnods,i,j,icase;
    double *astiff,*c;
    int nint,*ntnoel,MXNOEL;

    icase = 22;

    datasize6(&nnode,&nelem,&n_bc_given,&n_bc_nonzero,icase,&MXNOEL);

    a  = (double *)malloc(sizeof(double)*(nnode)*(nnode));
```

```
    b  = (double *)malloc(sizeof(double)*(nnode));
    coords = (double *)malloc(sizeof(double)*(nnode));
    lnods = (int *)malloc(sizeof(int)*(nelem)*MXNOEL);
    i_bc_given   = (int *)malloc(sizeof(int)*(n_bc_given));
    i_bc_nonzero = (int *)malloc(sizeof(int)*(n_bc_nonzero));
    v_bc_nonzero = (double *)malloc(sizeof(double)*(n_bc_nonzero));
    astiff = (double *)malloc(sizeof(double)*(MXNOEL)*(MXNOEL));
    c      = (double *)malloc(sizeof(double)*(MXNOEL));
    ntnoel = (int *)malloc(sizeof(int)*(nelem));

    datain6(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase,&nint,
           ntnoel,MXNOEL);

    stiff6(a,b,nnode,coords,nelem,lnods,astiff,c,nint,ntnoel,MXNOEL);

    bound2(a,b,nnode,n_bc_given,i_bc_given,
           n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);

    gauss_ver4u(a,b,nnode);

    check_solution6(b,nnode,coords,icase);
}
```

stiff5.c

```
void element(double *coords, int *lnods,double *astiff, int nint, int ntnoel);
void merge(double *a,int *lnods, double *astiff,int ntnoel,int nnode);

void stiff6(double *arg_a, double *arg_b, int nnode, double *arg_coords,
            int nelem, int *arg_lnods,
            double *arg_astiff, double *arg_c, int nint,
            int *arg_ntnoel, int MXNOEL)
{
    double *a, *b, *coords, *astiff, *c,x;
    int i,j,ielem,*lnods,*ntnoel,*ip;

    a = arg_a -1 -nnode;
    b = arg_b -1;

    coords = arg_coords -1;
    lnods = arg_lnods -1 -MXNOEL;

    astiff = arg_astiff -1 -MXNOEL;
    c = arg_c -1;
```

```
    ntnoel = arg_ntnoel -1;

    ip = (int *)malloc(sizeof(int)*(MXNOEL));
    ip = ip -1;

    for (i=1; i<=nnode; ++i){
        for (j=1; j<=nnode; ++j){
            a[i+j*nnode] = 0.0;
}
    }

    for (i=1; i<=nnode; ++i){
        b[i] = 0.0;
    }

    for (ielem = 1; ielem <= nelem; ++ielem){
        element(coords,&lnods[1+ielem*MXNOEL],astiff,nint,ntnoel[ielem]);
        merge(a,&lnods[1+ielem*MXNOEL],astiff,ntnoel[ielem],nnode);

        for(i = 1; i <= ntnoel[ielem]; ++i){
            ip[i] = lnods[i+ielem*MXNOEL];
        }

        x = coords[lnods[2+ielem*MXNOEL]]-coords[lnods[1+ielem*MXNOEL]];
        if (ntnoel[ielem] == 2){
            c[1] = x / 2.0;
            c[2] = x / 2.0;
        }
        if (ntnoel[ielem] == 3){
            c[1] = x / 6.0;
            c[2] = x / 6.0;
            c[3] = x * 2.0 / 3.0;
        }

        for (i = 1; i <= ntnoel[ielem]; ++i){
            b[ip[i]] = b[ip[i]] + c[i];
        }
    }

    free(arg_astiff);
    free(arg_c);
}

void element(double *coords, int *arg_lnods,double *astiff, int nint,
```

```
            int ntnoel)
{
    int *lnods;
    double x,r,ajacob,detjac,ajainv,detwei,
           *gsp,*wgh,*dndx,*dndr;
    int i,j,ir;

    static double arg_gsp[16] = { 0.0, 0.0, 0.0, 0.0,
                 -0.577350269189626, 0.577350269189626, 0.0, 0.0,
                 -0.774596669241483, 0.0, 0.774596669241483, 0.0,
                 -0.861136311594053,-0.339981043584856,
                  0.339981043584856, 0.861136311594053 };

    static double arg_wgh[16] = {2.0,0.0,0.0,0.0,
                  1.0,1.0,0.0,0.0,
                  0.555555555555556,0.888888888888889,0.555555555555556,0.0,
                  0.347854845137454,0.652145154862546,
                  0.652145154862546,0.347854845137454 };

    lnods  = arg_lnods  -1;

    dndx = (double *)malloc(sizeof(double)*(ntnoel));
    dndr = (double *)malloc(sizeof(double)*(ntnoel));

    gsp  = arg_gsp  -1 -4;
    wgh  = arg_wgh  -1 -4;
    dndx = dndx -1;
    dndr = dndr -1;

    for (i = 1; i <= ntnoel; ++i){
        for (j = 1; j <= ntnoel; ++j){
            astiff[i+j*ntnoel] = 0.0;
        }
    }

    for (ir = 1; ir <= nint; ++ir){
        r = gsp[ir + nint*4];

        if (ntnoel == 2){
            dndr[1] = -0.5;
            dndr[2] =  0.5;
        }
        if (ntnoel == 3){
            dndr[1] = r - 0.5;
            dndr[2] = r + 0.5;
```

```
            dndr[3] = -2.0 * r;
        }

        ajacob = 0.0;
        for (i = 1; i <= ntnoel; ++i){
            ajacob = ajacob + dndr[i] * coords[lnods[i]];
        }
        detjac = ajacob;
        ajainv = 1.0 / ajacob;

        for (i = 1; i <= ntnoel; ++i){
            dndx[i] = dndr[i] * ajainv;
        }

        detwei = detjac * wgh[ir+nint*4];

        for (i = 1; i <= ntnoel; ++i){
            for (j = 1; j <= ntnoel; ++j){
                astiff[i+j*ntnoel]=astiff[i+j*ntnoel]+detwei*dndx[i]*dndx[j];
            }
        }
    }
    dndx = dndx +1;
    dndr = dndr +1;
    free(dndx);
    free(dndr);
}

void merge(double *a,int *arg_lnods, double *astiff,int ntnoel,int nnode)
{
    int *lnods;
    int *ip,i,j;

    lnods = arg_lnods -1;

    ip = (int *)malloc(sizeof(int)*(ntnoel));
    ip = ip -1;

    for(i = 1; i <= ntnoel; ++i){
        ip[i] = lnods[i];
    }

    for(j = 1; j <= ntnoel; ++j){
        for(i = 1; i <= ntnoel; ++i){
            a[ip[i]+ ip[j]*nnode] = a[ip[i]+ ip[j]*nnode] + astiff[i+j*ntnoel];
```

```
        }
    }
}
```

misc5.c

```
#include <stdio.h>
void datasize6(int *nnode,int *nelem, int *n_bc_given, int *n_bc_nonzero,
               int icase, int *MXNOEL)
{
    FILE *fp;
    double coords;
    int i,j,inode,ielem,lnods,ibc,ntnoel;

    if (icase == 11){
        if ((fp = fopen("case11.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 12){
        if ((fp = fopen("case12.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 21){
        if ((fp = fopen("case21.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }    if (icase == 22){
        if ((fp = fopen("case22.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }

    *n_bc_given = 0;
    fscanf(fp,"%d",nnode);
    for (i = 1; i <= *nnode; ++i){
        fscanf(fp,"%d %d %lf", &inode, &ibc, &coords);
        if (ibc == 1){
            *n_bc_given = *n_bc_given + 1;
        }
```

```
    }

    *MXNOEL = 1;
    fscanf(fp,"%d",nelem);
    for (i = 1; i<= *nelem; ++i){
        fscanf(fp,"%d %d",&ielem,&ntnoel);
for (j = 1; j <= ntnoel; ++j){
            fscanf(fp,"%d",&lnods);
        }
        if (ntnoel > *MXNOEL) *MXNOEL = ntnoel;
    }

    fscanf(fp,"%d",n_bc_nonzero);

    fclose(fp);
}

void datain6(int nnode,double *arg_coords, int nelem, int *arg_lnods,
             int n_bc_given, int *arg_i_bc_given,int n_bc_nonzero,
             int *arg_i_bc_nonzero,double *arg_v_bc_nonzero,int icase,
             int *nint, int *arg_ntnoel, int MXNOEL)
{
    FILE *fp;
    int i,inode,ielem,ibc,idum,*lnods, *i_bc_given, *i_bc_nonzero, *ntnoel;
    double *coords, *v_bc_nonzero;

    coords       = arg_coords       -1;
    lnods        = arg_lnods        -1 -MXNOEL;
    i_bc_given   = arg_i_bc_given   -1;
    i_bc_nonzero = arg_i_bc_nonzero -1;
    v_bc_nonzero = arg_v_bc_nonzero -1;
    ntnoel       = arg_ntnoel       -1;

    if (icase == 11){
        if ((fp = fopen("case11.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 12){
        if ((fp = fopen("case12.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
```

```
    if (icase == 21){
        if ((fp = fopen("case21.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }    if (icase == 22){
        if ((fp = fopen("case22.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }

    i = 0;
    fscanf(fp,"%d",&idum);
    for (inode = 1; inode <= nnode; ++inode){
        fscanf(fp,"%d %d %lf", &idum, &ibc, &coords[inode]);
        if (ibc == 1){
            i = i + 1;
            i_bc_given[i] = inode;
        }
    }

    fscanf(fp,"%d",&idum);
    for (ielem = 1; ielem <= nelem; ++ielem){
        fscanf(fp,"%d %d",&idum,&ntnoel[ielem]);
for (i = 1; i <= ntnoel[ielem]; ++i){
            fscanf(fp,"%d",&lnods[i+ielem*MXNOEL]);
        }
    }

    fscanf(fp,"%d",&idum);
    for (i = 1; i <= n_bc_nonzero; ++i){
        fscanf(fp,"%d %d %lf",&ibc,&i_bc_nonzero[i],&v_bc_nonzero[i]);
    }

    fscanf(fp,"%d",nint);

    fclose(fp);
}

void check_solution6(double *arg_b,int nnode, double *arg_coords, int icase)
{
    double *b,*coords;
    int i;
    double x,u;
```

```
    b = arg_b -1;
    coords = arg_coords -1;

    for (i=1; i<=nnode; ++i){
        x = coords[i];
        if (icase < 20) u = -0.5 * x*x - 0.5 * x + 1.0;
        if (icase > 20) u = -0.5 * x*x + 1.5 * x + 1.0;
        printf("  % 10.3E  % 10.3E\n",b[i],u);
    }
}
```

gauss_ver4u.c　第 1 章と共通

bound2.c　第 1 章と共通

2.3.12　バージョン 7 — 3 次の補間関数

バージョン 6 までで き上がっていれば, 3 次の補間関数を導入することも簡単である. 3 次の補間関数は 4 節点から構成され, 図 2.4 のような節点の配置であれば補間関数は以下のようになる.

$$N^{(1)} = -\frac{1}{16}(1-r)(1-9r^2) \tag{2.66}$$

$$N^{(2)} = -\frac{1}{16}(1+r)(1-9r^2) \tag{2.67}$$

$$N^{(3)} = \frac{9}{16}(1-3r)(1-r^2) \tag{2.68}$$

$$N^{(4)} = \frac{9}{16}(1+3r)(1-r^2) \tag{2.69}$$

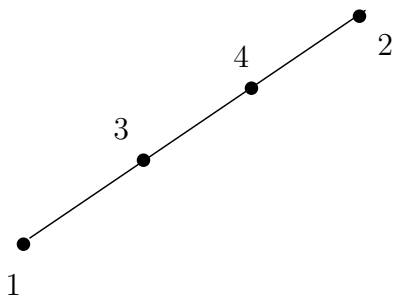


図 2.4: 4 節点の場合での節点の配置

$\dfrac{dN^{(i)}}{dr}$ については以下のようになる.

$$\frac{dN^{(1)}}{dr} = -\frac{1}{16}\left(-1 - 18r + 27r^2\right) \tag{2.70}$$

$$\frac{dN^{(2)}}{dr} = -\frac{1}{16}\left(1 - 18r - 27r^2\right) \tag{2.71}$$

$$\frac{dN^{(3)}}{dr} = \frac{9}{16}\left(-3 - 2r + 9r^2\right) \tag{2.72}$$

$$\frac{dN^{(4)}}{dr} = \frac{9}{16}\left(3 - 2r - 9r^2\right) \tag{2.73}$$

これを用いると $\dfrac{dx}{dr}$ は以下のようになる.

$$\begin{aligned}\frac{dx}{dr} = &-\frac{1}{16}\left(-1 - 18r + 27r^2\right) x_{(i,1)} - \frac{1}{16}\left(1 - 18r - 27r^2\right) x_{(i,2)} \\ &+ \frac{9}{16}\left(-3 - 2r + 9r^2\right) x_{(i,3)} + \frac{9}{16}\left(3 - 2r - 9r^2\right) x_{(i,4)}\end{aligned} \tag{2.74}$$

もし $x_{(i,3)}$ が $x_{(i,1)}$ と $x_{(i,2)}$ の $x_{(i,1)}$ に近い 3 等分点で, $x_{(i,4)}$ が $x_{(i,1)}$ と $x_{(i,2)}$ の $x_{(i,4)}$ に近い 3 等分点, 即ち

$$x_{(i,3)} = \frac{2}{3} x_{(i,1)} + \frac{1}{3} x_{(i,2)} \tag{2.75}$$

$$x_{(i,4)} = \frac{1}{3} x_{(i,1)} + \frac{2}{3} x_{(i,2)} \tag{2.76}$$

であれば, $\dfrac{dx}{dr}$ は以下のように単純化される.

$$\begin{aligned}\frac{dx}{dr} &= -\frac{1}{16}\left(-1-18r+27r^2\right)x_{(i,1)} - \frac{1}{16}\left(1-18r-27r^2\right)x_{(i,2)} \\ &\quad + \frac{9}{16}\left(-3-2r+9r^2\right)\left(\frac{2}{3}x_{(i,1)} + \frac{1}{3}x_{(i,2)}\right) + \frac{9}{16}\left(3-2r-9r^2\right)\left(\frac{1}{3}x_{(i,1)} + \frac{2}{3}x_{(i,2)}\right) \\ &= \frac{x_{(i,2)} - x_{(i,1)}}{2}\end{aligned}$$

これよりヤコビアン $J_{(i)}$ は以下のようになる.

$$J_{(i)} = \frac{x_{(i,2)} - x_{(i,1)}}{2} \tag{2.77}$$

右辺については式 (2.51) と同様に以下のようになる.

$$\{F^{(i)}\} = \begin{Bmatrix} F_1^{(i)} \\ F_2^{(i)} \\ F_3^{(i)} \\ F_4^{(i)} \end{Bmatrix} = \int_{-1}^{1} \begin{Bmatrix} N^{(i)} \\ N^{(i)} \\ N^{(i)} \\ N^{(i)} \end{Bmatrix} f J_{(i)} dr \tag{2.78}$$

$\{F^{(i)}\}$ の各成分を計算すると以下のようになる.

$$\begin{aligned} F_1^{(i)} &= \int_{-1}^{1} -\frac{1}{16}\left(1-r\right)\left(1-9r^2\right)\frac{x_{(i,2)} - x_{(i,1)}}{2}\, dr \\ &= \frac{1}{8}\left(x_{(i,2)} - x_{(i,1)}\right) \qquad (2.79) \\ F_2^{(i)} &= \int_{-1}^{1} -\frac{1}{16}\left(1+r\right)\left(1-9r^2\right)\frac{x_{(i,2)} - x_{(i,1)}}{2}\, dr \\ &= \frac{1}{8}\left(x_{(i,2)} - x_{(i,1)}\right) \qquad (2.80) \\ F_3^{(i)} &= \int_{-1}^{1} \frac{9}{16}\left(1-3r\right)\left(1-r^2\right)\frac{x_{(i,2)} - x_{(i,1)}}{2}\, dr \\ &= \frac{3}{8}\left(x_{(i,2)} - x_{(i,1)}\right) \qquad (2.81) \\ F_4^{(i)} &= \int_{-1}^{1} \frac{9}{16}\left(1+3r\right)\left(1-r^2\right)\frac{x_{(i,2)} - x_{(i,1)}}{2}\, dr \\ &= \frac{3}{8}\left(x_{(i,2)} - x_{(i,1)}\right) \qquad (2.82) \end{aligned}$$

2.3.13　入力ファイル例

入力ファイルは fortran, C とも共通である.

case11.dat 1 次補間　バージョン 6 と共通

case12.dat 2 次補間　バージョン 6 と共通

case13.dat 3 次補間

```
7
 1 1  0.0
 2 0  0.1666666666
 3 0  0.3333333333
 4 0  0.5
 5 0  0.6666666666
 6 0  0.8333333333
 7 1  1.0
2
 1  4  1  4  2  3
 2  4  4  7  5  6
1
 1  1  1.0
4
```

case21.dat 1 次補間　バージョン 6 と共通

case22.dat 2 次補間　バージョン 6 と共通

case23.dat 3 次補間

```
7
 1 1  0.0
 2 0  0.1666666666
 3 0  0.3333333333
 4 0  0.5
 5 0  0.6666666666
 6 0  0.8333333333
 7 1  1.0
2
```

```
 1  4  1  4  2  3
 2  4  4  7  5  6
2
 1  1  1.0
 2  7  2.0
4
```

2.3.14 fortran コーディング例

fortran コーディング例を示すと以下のようになる.

Makefile

```
FC = g77
F_OPT = -O2

OBJS = main.o stiff7.o misc7.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
        $(FC) $(F_OPT) -o $(TARGET) $(OBJS)

.f.o:
        $(FC) -c $(F_OPT) $<
```

main.f

```
      implicit real*8(a-h,o-z)
      parameter (MXNODE = 100)
      parameter (MXELEM = 100)
      dimension a(MXNODE*MXNODE),b(MXNODE)
      dimension coords(MXNODE),lnods(4,MXELEM)
      dimension i_bc_given(MXNODE),i_bc_nonzero(MXNODE),
     1          v_bc_nonzero(MXNODE)
      dimension astiff(4,4),c(4)
      dimension ntnoel(MXELEM)
c
      icase = 23
c
      call datain7(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
```

```
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase,nint,
     1            ntnoel)
c
      call stiff7(a,b,nnode,coords,nelem,lnods,astiff,c,nint,ntnoel)
c
      call bound2(a,b,nnode,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero)
c
      call gauss_ver4u(a,b,nnode)
c
      call check_solution6(b,nnode,coords,icase)
c
      stop
      end
c
```

stiff7.f

```
c######################################################################
      subroutine stiff7(a,b,nnode,coords,nelem,lnods,astiff,c,nint,
     1                  ntnoel)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(nnode,*),b(*),coords(*),lnods(4,*)
      dimension astiff(*),c(*),ntnoel(*),ip(100)
c
      do i = 1,nnode
          do j = 1,nnode
              a(i,j) = 0.D0
          end do
      end do
c
      do i = 1,nnode
          b(i) = 0.D0
      end do
c
      do ielem = 1,nelem
          call element(coords,lnods(1,ielem),astiff,nint,ntnoel(ielem))
c
          call amerge(a,lnods(1,ielem),astiff,ntnoel(ielem),nnode)
c
          do i = 1,ntnoel(ielem)
             ip(i) = lnods(i,ielem)
          end do
c
```

```
          x = coords(lnods(2,ielem)) - coords(lnods(1,ielem))
          if (ntnoel(ielem) .eq. 2) then
              c(1) = x / 2.D0
              c(2) = x / 2.D0
          end if
          if (ntnoel(ielem) .eq. 3) then
              c(1) = x / 6.D0
              c(2) = x / 6.D0
              c(3) = x * 2.D0/ 3.D0
          end if
          if (ntnoel(ielem) .eq. 4) then
              c(1) = x / 8.D0
              c(2) = x / 8.D0
              c(3) = x * 3.D0/ 8.D0
              c(4) = x * 3.D0/ 8.D0
          end if
c
          do i = 1,ntnoel(ielem)
              b(ip(i)) = b(ip(i)) + c(i)
          end do
      end do
c
      return
      end
c
c######################################################################
      subroutine element(coords,lnods,astiff,nint,ntnoel)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(*),lnods(*),astiff(ntnoel,*)
      dimension gsp(4,4),wgh(4,4),dndx(4),dndr(4)
c
      data gsp /  0.D0          , 0.D0          , 0.D0,          0.D0,
     1           -0.577350269189626D0, 0.577350269189626D0, 0.D0,0.D0,
     1           -0.774596669241483D0, 0.D0, 0.774596669241483D0,0.D0,
     1           -0.861136311594053D0,-0.339981043584856D0,
     1            0.339981043584856D0, 0.861136311594053D0 /
      data wgh /  2.D0          , 0.D0          , 0.D0,          0.D0,
     1            1.D0          , 1.D0          , 0.D0,          0.D0,
     1            0.555555555555556D0, 0.888888888888889D0,
     1            0.555555555555556D0, 0.D0,
     1            0.347854845137454D0, 0.652145154862546D0,
     1            0.652145154862546D0, 0.347854845137454D0 /
c
      do j = 1,ntnoel
```

```
          do i = 1,ntnoel
              astiff(i,j) = 0.D0
          end do
      end do
c
      do ir = 1,nint
c
          r = gsp(ir,nint)
c
          if (ntnoel .eq. 2) then
              dndr(1) = -0.5D0
              dndr(2) =  0.5D0
          end if
          if (ntnoel .eq. 3) then
              dndr(1) = r - 0.5
              dndr(2) = r + 0.5
              dndr(3) = -2.D0 * r
          end if
          if (ntnoel .eq. 4) then
              dndr(1) = -(-1.D0 - 18.D0*r + 27.D0*r*r) / 16.D0
              dndr(2) = -( 1.D0 - 18.D0*r - 27.D0*r*r) / 16.D0
              dndr(3) = 9.D0*(-3.D0 - 2.D0*r + 9.D0*r*r) / 16.D0
              dndr(4) = 9.D0*( 3.D0 - 2.D0*r - 9.D0*r*r) / 16.D0
          end if
c
          ajacob = 0.D0
          do i = 1,ntnoel
              ajacob = ajacob + dndr(i) * coords(lnods(i))
          end do
          detjac = ajacob
          ajainv = 1.D0 / ajacob
c
          do i = 1,ntnoel
              dndx(i) = dndr(i) * ajainv
          end do
c
          detwei = detjac * wgh(ir,nint)
c
          do j = 1,ntnoel
              do i = 1,ntnoel
                  astiff(i,j) = astiff(i,j) + detwei * dndx(i) * dndx(j)
              end do
          end do
      end do
c
```

```
      return
      end
c
c######################################################################
      subroutine amerge(a,lnods,astiff,ntnoel,nnode)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(*),lnods(*),astiff(ntnoel,*)
      dimension ip(100)
c
      do i = 1,ntnoel
          ip(i) = lnods(i)
      end do
c
      do j = 1,ntnoel
          do i = 1,ntnoel
              a((ip(j)-1)*nnode+ip(i)) =  a((ip(j)-1)*nnode+ip(i))
     1                 + astiff(i,j)
          end do
      end do
c
      return
      end
c
```

misc7.f

```
c######################################################################
      subroutine datain7(nnode,coords,nelem,lnods,n_bc_given,i_bc_given,
     1            n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase,nint,
     1            ntnoel)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(*),lnods(4,*),i_bc_given(*),i_bc_nonzero(*),
     1          v_bc_nonzero(*),ntnoel(*)
c
      if (icase .eq. 11) open(10,file='case11.dat')
      if (icase .eq. 12) open(10,file='case12.dat')
      if (icase .eq. 13) open(10,file='case13.dat')
      if (icase .eq. 21) open(10,file='case21.dat')
      if (icase .eq. 22) open(10,file='case22.dat')
      if (icase .eq. 23) open(10,file='case23.dat')
c
      n_bc_given = 0
      read(10,*) nnode
```

```
      do i = 1, nnode
          read(10,*) itemp, ibc,coords(i)
          if (ibc .eq. 1) then
              n_bc_given = n_bc_given + 1
              i_bc_given(n_bc_given) = i
          end if
      end do
c
      read(10,*) nelem
      do i = 1,nelem
          read(10,*) itemp, ntnoel(i),(lnods(j,i),j=1,ntnoel(i))
      end do
c
      read(10,*) n_bc_nonzero
      do i = 1,n_bc_nonzero
          read(10,*) itemp, i_bc_nonzero(i),v_bc_nonzero(i)
      end do
c
      read(10,*) nint
c
      close(10)
c
      return
      end
c
c######################################################################
      subroutine check_solution6(b,nnode,coords,icase)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension b(*),coords(*)
c
      do i = 1,nnode
          x = coords(i)
          if (icase .lt. 20) u = -0.5D0 * x*x - 0.5D0 * x + 1
          if (icase .gt. 20) u = -0.5D0 * x*x + 1.5D0 * x + 1
          write(*,1000) b(i), u
      end do
 1000 format(2x,e13.6,2x,e13.6)
c
      return
      end
c
```

gauss_ver4u.f 第 1 章と共通

bound2.f 第 1 章と共通

2.3.15 C コーディング例

C コーディング例を示すと以下のようになる.

Makefile

```
CC = gcc
C_OPT = -O2

OBJS = main.o stiff7.o misc7.o gauss_ver4u.o bound2.o

TARGET = onedim

$(TARGET):$(OBJS)
$(CC) $(C_OPT) -o $(TARGET) $(OBJS)

.c.o:
$(CC) -c $(C_OPT) $<
```

main.c

```
#include <stdio.h>

void datain31(int *nnode,int *nelem,int icase);
void datain5(int *nnode, double *coords, int *nelem, int *lnods,
             int *n_bc_given, int *i_bc_given,int *n_bc_nonzero,
             int *i_bc_nonzero,double *v_bc_nonzero,int icase,int *nint);
void stiff5(double *a, double *b, int nnode, double *coords,
            int nelem, int *lnods, double *astiff, double *c, int nint);
void bound2(double *a, double *c, int n, int *n_bc_given,
            int *i_bc_given, int *n_bc_nonzero,
            int *i_bc_nonzero, double *v_bc_nonzero);
void gauss_ver4u(double *a, double *c, int n);
void check_solution3(double *b,int nnode, double *coords,int icase);

main()
```

```
{
    double *a,*b,*v_bc_nonzero;
    int n_bc_given,*i_bc_given,n_bc_nonzero,*i_bc_nonzero;
    double *coords;

    int nnode,nelem,*lnods,i,j,icase;
    double *astiff,*c;
    int nint,*ntnoel,MXNOEL;

    icase = 23;

    datain71(&nnode,&nelem,icase,&MXNOEL);

    a  = (double *)malloc(sizeof(double)*(nnode)*(nnode));
    b  = (double *)malloc(sizeof(double)*(nnode));
    v_bc_nonzero = (double *)malloc(sizeof(double)*(nnode));
    i_bc_given   = (int *)malloc(sizeof(int)*(nnode));
    i_bc_nonzero = (int *)malloc(sizeof(int)*(nnode));
    coords = (double *)malloc(sizeof(double)*(nnode));
    lnods = (int *)malloc(sizeof(int)*(nelem)*MXNOEL);
    astiff = (double *)malloc(sizeof(double)*(MXNOEL)*(MXNOEL));
    c = (double *)malloc(sizeof(double)*(MXNOEL));
    ntnoel = (int *)malloc(sizeof(int)*(nelem));

    datain7(&nnode,coords,&nelem,lnods,&n_bc_given,i_bc_given,
           &n_bc_nonzero,i_bc_nonzero,v_bc_nonzero,icase,&nint,
           ntnoel,MXNOEL);

    stiff7(a,b,nnode,coords,nelem,lnods,astiff,c,nint,ntnoel,MXNOEL);

    bound2(a,b,nnode,&n_bc_given,i_bc_given,
           &n_bc_nonzero,i_bc_nonzero,v_bc_nonzero);

    gauss_ver4u(a,b,nnode);

    check_solution6(b,nnode,coords,icase);
}
```

stiff7.c

```
void element(double *coords, int *lnods,double *astiff, int nint, int ntnoel);

void stiff7(double *arg_a, double *arg_b, int nnode, double *arg_coords,
            int nelem, int *arg_lnods,
            double *arg_astiff, double *arg_c, int nint,
```

```
            int *arg_ntnoel, int MXNOEL)
{
    double *a, *b;
    double x;
    int i,j,ielem;
    double *coords;
    int *lnods;
    double *astiff, *c;
    int *ntnoel,*ip;

    a = arg_a -1 -nnode;
    b = arg_b -1;

    coords = arg_coords -1;
    lnods = arg_lnods -1 -MXNOEL;

    astiff = arg_astiff -1 -MXNOEL;
    c = arg_c -1;

    ntnoel = arg_ntnoel -1;

    ip = (int *)malloc(sizeof(int)*(MXNOEL));
    ip = ip -1;

    for (i=1; i<=nnode; ++i){
        for (j=1; j<=nnode; ++j){
            a[i+j*nnode] = 0.0;
}
    }

    for (i=1; i<=nnode; ++i){
        b[i] = 0.0;
    }

    for (ielem = 1; ielem <= nelem; ++ielem){
        element(coords,&lnods[1+ielem*MXNOEL],astiff,nint,ntnoel[ielem]);
        amerge(a,&lnods[1+ielem*MXNOEL],astiff,ntnoel[ielem],nnode);

        for(i = 1; i <= ntnoel[ielem]; ++i){
            ip[i] = lnods[i+ielem*MXNOEL];
        }

        x = coords[lnods[2+ielem*MXNOEL]]-coords[lnods[1+ielem*MXNOEL]];
        if (ntnoel[ielem] == 2){
            c[1] = x / 2.0;
```

```
            c[2] = x / 2.0;
        }
        if (ntnoel[ielem] == 3){
            c[1] = x / 6.0;
            c[2] = x / 6.0;
            c[3] = x * 2.0 / 3.0;
        }
        if (ntnoel[ielem] == 4){
            c[1] = x / 8.0;
            c[2] = x / 8.0;
            c[3] = x * 3.0 / 8.0;
            c[4] = x * 3.0 / 8.0;
        }

        for (i = 1; i <= ntnoel[ielem]; ++i){
            b[ip[i]] = b[ip[i]] + c[i];
        }
    }

    free(arg_astiff);
    free(arg_c);
}

void element(double *coords, int *arg_lnods,double *astiff, int nint,
             int ntnoel)
{
    int *lnods;
    double x,r,ajacob,detjac,ajainv,detwei,
           *gsp,*wgh,*dndx,*dndr;
    int i,j,ir;

    static double arg_gsp[16] = { 0.0, 0.0, 0.0, 0.0,
                 -0.577350269189626, 0.577350269189626, 0.0, 0.0,
                 -0.774596669241483, 0.0, 0.774596669241483, 0.0,
                 -0.861136311594053,-0.339981043584856,
                  0.339981043584856, 0.861136311594053 };

    static double arg_wgh[16] = {2.0,0.0,0.0,0.0,
                  1.0,1.0,0.0,0.0,
                  0.555555555555556,0.888888888888889,0.555555555555556,0.0,
                  0.347854845137454,0.652145154862546,
                  0.652145154862546,0.347854845137454 };

    lnods  = arg_lnods  -1;
```

```
    dndx = (double *)malloc(sizeof(double)*(ntnoel));
    dndr = (double *)malloc(sizeof(double)*(ntnoel));

    gsp  = arg_gsp  -1 -4;
    wgh  = arg_wgh  -1 -4;
    dndx = dndx -1;
    dndr = dndr -1;

    for (i = 1; i <= ntnoel; ++i){
        for (j = 1; j <= ntnoel; ++j){
            astiff[i+j*ntnoel] = 0.0;
        }
    }

    for (ir = 1; ir <= nint; ++ir){
        r = gsp[ir + nint*4];

        if (ntnoel == 2){
            dndr[1] = -0.5;
            dndr[2] =  0.5;
        }
        if (ntnoel == 3){
            dndr[1] = r - 0.5;
            dndr[2] = r + 0.5;
            dndr[3] = -2.0 * r;
        }
        if (ntnoel == 4){
            dndr[1] = -(-1.0 - 18.0*r + 27.0*r*r) / 16.0;
            dndr[2] = -( 1.0 - 18.0*r - 27.0*r*r) / 16.0;
            dndr[3] = 9.0*(-3.0 - 2.0*r + 9.0*r*r) / 16.0;
            dndr[4] = 9.0*( 3.0 - 2.0*r - 9.0*r*r) / 16.0;
        }

        ajacob = 0.0;
        for (i = 1; i <= ntnoel; ++i){
            ajacob = ajacob + dndr[i] * coords[lnods[i]];
        }
        detjac = ajacob;
        ajainv = 1.0 / ajacob;

        for (i = 1; i <= ntnoel; ++i){
            dndx[i] = dndr[i] * ajainv;
        }

        detwei = detjac * wgh[ir+nint*4];
```

```
        for (i = 1; i <= ntnoel; ++i){
            for (j = 1; j <= ntnoel; ++j){
                astiff[i+j*ntnoel]=astiff[i+j*ntnoel]+detwei*dndx[i]*dndx[j];
            }
        }
    }
    dndx = dndx +1;
    dndr = dndr +1;
    free(dndx);
    free(dndr);
}
void amerge(double *a,int *arg_lnods, double *astiff,int ntnoel,int nnode)
{
    int *lnods;
    int *ip,i,j;

    lnods = arg_lnods -1;

    ip = (int *)malloc(sizeof(int)*(ntnoel));
    ip = ip -1;

    for(i = 1; i <= ntnoel; ++i){
        ip[i] = lnods[i];
    }

    for(j = 1; j <= ntnoel; ++j){
        for(i = 1; i <= ntnoel; ++i){
            a[ip[i]+ ip[j]*nnode] = a[ip[i]+ ip[j]*nnode] + astiff[i+j*ntnoel];
        }
    }
}
```

misc7.c

```
#include <stdio.h>
void datain71(int *nnode,int *nelem, int icase, int *MXNOEL)
{
    FILE *fp;
    double coords;
    int i,j,inode,ielem,lnods,ibc,ntnoel;

    if (icase == 11){
        if ((fp = fopen("case11.dat","r")) == NULL){
            printf("can't open file\n");
```

```
            exit(1);
        }
    }
    if (icase == 12){
        if ((fp = fopen("case12.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 13){
        if ((fp = fopen("case13.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 21){
        if ((fp = fopen("case21.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 22){
        if ((fp = fopen("case22.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }

    if (icase == 23){
        if ((fp = fopen("case23.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }

    fscanf(fp,"%d",nnode);
    for (i = 1; i<=*nnode; ++i){
        fscanf(fp,"%d %d %lf", &inode, &ibc, &coords);
    }

    *MXNOEL = 1;
    fscanf(fp,"%d",nelem);
    for (i = 1; i<=*nelem; ++i){
        fscanf(fp,"%d %d",&ielem,&ntnoel);
for (j = 1; j <= ntnoel; ++j){
```

```
            fscanf(fp,"%d",&lnods);
        }
        if (ntnoel > *MXNOEL) *MXNOEL = ntnoel;
    }

    fclose(fp);
}

void datain7(int *nnode,double *arg_coords, int *nelem, int *arg_lnods,
             int *n_bc_given, int *arg_i_bc_given,int *n_bc_nonzero,
             int *arg_i_bc_nonzero,double *arg_v_bc_nonzero,int icase,
             int *nint, int *arg_ntnoel, int MXNOEL)
{
    FILE *fp;
    int i,j,inode,ielem,ibc;
    double *coords, *v_bc_nonzero;
    int *lnods, *i_bc_given, *i_bc_nonzero, *ntnoel;

    coords = arg_coords -1;
    lnods = arg_lnods -1-MXNOEL;
    i_bc_given = arg_i_bc_given -1;
    i_bc_nonzero = arg_i_bc_nonzero -1;
    v_bc_nonzero = arg_v_bc_nonzero -1;
    ntnoel = arg_ntnoel -1;

    if (icase == 11){
        if ((fp = fopen("case11.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 12){
        if ((fp = fopen("case12.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 13){
        if ((fp = fopen("case13.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 21){
        if ((fp = fopen("case21.dat","r")) == NULL){
```

```
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 22){
        if ((fp = fopen("case22.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }
    if (icase == 23){
        if ((fp = fopen("case23.dat","r")) == NULL){
            printf("can't open file\n");
            exit(1);
        }
    }

    *n_bc_given = 0;
    fscanf(fp,"%d",nnode);
    for (i = 1; i<=*nnode; ++i){
        fscanf(fp,"%d %d %lf", &inode, &ibc, &coords[i]);
        if (ibc == 1){
            *n_bc_given = *n_bc_given + 1;
            i_bc_given[*n_bc_given] = i;
        }
    }

    fscanf(fp,"%d",nelem);
    for (i = 1; i<=*nelem; ++i){
        fscanf(fp,"%d %d",&ielem,&ntnoel[i]);
for (j = 1; j <= ntnoel[i]; ++j){
            fscanf(fp,"%d",&lnods[j+i*MXNOEL]);
        }
    }

    fscanf(fp,"%d",n_bc_nonzero);
    for (i = 1; i<=*n_bc_nonzero; ++i){
        fscanf(fp,"%d %d %lf",&ibc,&i_bc_nonzero[i],&v_bc_nonzero[i]);
    }

    fscanf(fp,"%d",nint);

    fclose(fp);
}
```

```
void check_solution6(double *arg_b,int nnode, double *arg_coords, int icase)
{
    double *b,*coords;
    int i;
    double x,u;

    b = arg_b -1;
    coords = arg_coords -1;

    for (i=1; i<=nnode; ++i){
        x = coords[i];
        if (icase < 20) u = -0.5 * x*x - 0.5 * x + 1.0;
        if (icase > 20) u = -0.5 * x*x + 1.5 * x + 1.0;
        printf("  % 10.3E  % 10.3E\n",b[i],u);
    }
}
```

gauss_ver4u.c　第 1 章と共通

bound2.c　第 1 章と共通

第3章　線形弾性体の有限要素定式化

3.1　線形弾性体の境界値問題

3.1.1　強形式

図 3.1 に示すような, 線形弾性体 A についての境界値問題 $[B]$ を考える.

$[B]$　物体 A が占める領域を $\Omega \subset \boldsymbol{R}^N$, Ω の境界を $\partial\Omega$ とし, $\partial\Omega$ の部分集合 $\partial\Omega_D$ 上では変位境界条件が与えられているものとする. このような系に表面力 $\boldsymbol{t}$, 体積力 $\rho\boldsymbol{g}$ が作用するとき, つり合い条件を満たす変位 $\boldsymbol{u} \in \boldsymbol{V}$ を求めよ. ただし, ρ は密度, $\boldsymbol{g}$ は重力加速度, $\boldsymbol{V}$ は変位の許容関数すなわち境界条件を満たす解の候補全体の集合とする. ここでは $\boldsymbol{V}$ として,

$$V = \left\{ \boldsymbol{v} \,\middle|\, v_i \in L^2(\Omega), \frac{\partial v_i}{\partial x_j} \in L^2(\Omega) \quad i,j = 1,\ldots,N, \quad \boldsymbol{v} = 0 \text{ on } \partial\Omega_D \right\} \tag{3.1}$$

を用い, 簡略化のため $\boldsymbol{t}$, $\boldsymbol{g}$ は dead load すなわち変形に伴って大きさや方向が変わらないような荷重であるとする. また N は空間の次元数であり 2 次元, 3 次元解析に対応してそれぞれ $N = 2$, $N = 3$ となる. 式 (3.1) における $L^2(\Omega)$ は Ω 上で定義された自乗可積分[1]の関数の集合で Hilbert 空間[2]である.

この問題は以下のように定式化できる.

---

[1]ある関数 $\boldsymbol{v}$ に対して, $\boldsymbol{v}^2$ の領域積分 $\int_\Omega \boldsymbol{v}^2 \mathrm{d}\Omega$ が有限確定値を持つこと. 詳細は関数解析の教科書を見て下さい.

[2]内積が定義されていて, 内積により定められたノルムにより Cauchy 列が収束するような空間. 詳細は関数解析の教科書を見て下さい.

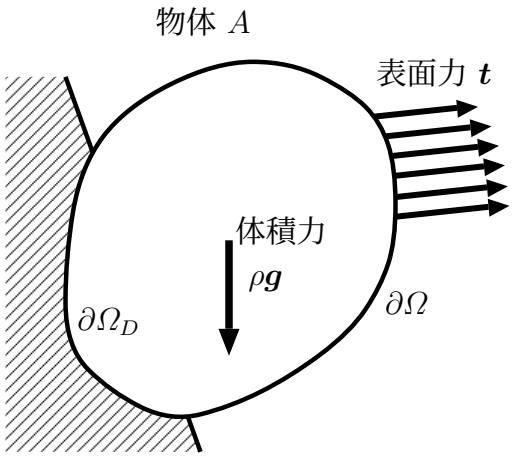


図 3.1: 境界値問題概念図

$[B]$　与えられた $\boldsymbol{t}, \boldsymbol{g}$ に対し, 以下を満たすような $\boldsymbol{u} \in \boldsymbol{V}$ を求めよ.

$$\nabla_x \cdot \boldsymbol{T} + \rho\boldsymbol{g} = 0 \tag{3.2}$$

$$\boldsymbol{T}^T \cdot \boldsymbol{n} = \boldsymbol{t} \tag{3.3}$$

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{3.4}$$

$$T_{ij} = \kappa(\operatorname{div}\boldsymbol{u})\delta_{ij} + 2G\varepsilon^D_{ij}(\boldsymbol{u}) \tag{3.5}$$

ただし, $\boldsymbol{V}$ は先に定義した変位の許容関数全体の集合, $\boldsymbol{T}$ は Cauchy 応力, κ は体積弾性係数, G はせん断弾性係数である. また δ_{ij} は Kronecker のデルタ記号, ε_{ij} は線形ひずみ, ε^D_{ij} は偏差ひずみでそれぞれ以下のように定義されている[3].

$$\delta_{ij} = \begin{cases} 1 & (i = j) \\ 0 & (i \neq j) \end{cases} \tag{3.6}$$

$$\varepsilon_{ij}(\boldsymbol{u}) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{3.7}$$

$$\varepsilon^D_{ij}(\boldsymbol{u}) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{1}{3}(\operatorname{div}\boldsymbol{u})\delta_{ij} \tag{3.8}$$

[3]これらの物理的な意味は, 連続体力学の教科書に詳しく載っています. ここでは単に, このような式が与えられた場合にどのように解くか, について考えることにします.

2.1 節で述べているように $[B]$ を偏微分方程式の強形式という. この $[B]$ が解析的に解けるのは非常に限られたケースのみであり, 通常は近似解法を用いる. 近似解法の代表が有限要素法である.

3.1.2 弱形式と停留ポテンシャルエネルギーの原理

前述のように有限要素法は, 偏微分方程式の弱形式をもとにした近似解法である. ここで微分方程式の弱形式について簡単に述べる. [4]

許容関数全体の集合を $\boldsymbol{V}$, 解くべき方程式を

$$\boldsymbol{Q}(\boldsymbol{u}) = \boldsymbol{F} \tag{3.9}$$

としたとき

$$(\boldsymbol{v}, \boldsymbol{Q}(\boldsymbol{u}) - \boldsymbol{F}) = 0 \qquad \forall \boldsymbol{v} \in \boldsymbol{V} \tag{3.10}$$

を境界値問題の弱形式と呼び, 式 (3.10) を満たす解を広義解あるいは弱の解と呼ぶ. ただし $(\quad , \quad)$ は $\boldsymbol{V}$ 上で定義された内積を表す. 解析学の教科書などで, 点列 $\boldsymbol{x}_n \in \boldsymbol{V}$ が

$$\lim_{n\to\infty} \boldsymbol{x}_n \to \boldsymbol{x}_0 \tag{3.11}$$

のとき強収束,

$$\lim_{n\to\infty} (\boldsymbol{x}_n, \boldsymbol{x}) \to (\boldsymbol{x}_0, \boldsymbol{x}) \tag{3.12}$$

のとき弱収束と書いてあるのを見たことがあるかもしれないが, 強形式, 弱形式というときの強, 弱はこの意味である. また証明は省略するが, Lax-Milgram の定理によって, 弱の解が唯一に存在し, かつ強の解と一致することが保証されている.

次に実際に $[B]$ の弱形式を導く. 2.1 節では強形式の方程式の両辺に $\boldsymbol{v} \in \boldsymbol{V}$ をかけ領域積分を行ったが, 線形弾性体の境界値問題を取り扱う際にはポテンシャルエネルギーの最小化問題を基礎にして定式化が行われることが多い. [5]

[4] 偏微分方程式の弱形式などの議論は関数解析の教科書に詳しく載っています.

[5] この理由としては

(i) ポテンシャルが存在するときは剛性マトリックスは対称になる.

(ii) 付帯条件がある場合に Lagrange 未定乗数法やペナルティ法を導入することにより対処可能である.

などが挙げられる.

$[M]$　与えられた $\boldsymbol{t}$, $\boldsymbol{g}$ に対し, 以下を満たすような $\boldsymbol{u} \in \boldsymbol{V}$ を求めよ.

$$\boldsymbol{\Pi}(\boldsymbol{u}) \leq \boldsymbol{\Pi}(\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V} \tag{3.13}$$

ただし,

$$\begin{aligned}\boldsymbol{\Pi}(\boldsymbol{v}) =& \int_\Omega \frac{1}{2}\, T_{ij}(\boldsymbol{v})\, \varepsilon_{ij}(\boldsymbol{v})\, \mathrm{d}\Omega - \int_{\partial\Omega} \boldsymbol{v}\cdot\boldsymbol{t}\, \mathrm{d}S - \int_\Omega \boldsymbol{v}\cdot\boldsymbol{g}\, \mathrm{d}\Omega \\ =& \frac{\kappa}{2}\int_\Omega (\operatorname{div}\boldsymbol{v})^2\, \mathrm{d}\Omega + G\int_\Omega \varepsilon^D_{ij}(\boldsymbol{v})\, \varepsilon^D_{ij}(\boldsymbol{v})\, \mathrm{d}\Omega - \int_{\partial\Omega} \boldsymbol{v}\cdot\boldsymbol{t}\, \mathrm{d}S - \int_\Omega \rho\boldsymbol{v}\cdot\boldsymbol{g}\, \mathrm{d}\Omega \end{aligned} \tag{3.14}$$

式 (3.14) の中の $T_{ij}(\boldsymbol{v})\, \varepsilon_{ij}(\boldsymbol{v})$ あるいは $\varepsilon^D_{ij}(\boldsymbol{v})\, \varepsilon^D_{ij}(\boldsymbol{v})$ は, 総和規約による. すなわち, 同じ添字が 1 つの積の中で 2 回以上現れている場合, その添字をダミーインデックスと呼び, $\sum$ 記号が明示されていなくても, 3 次元の場合なら 1 から 3 まで変化させて和をとるものとする. $\sum$ を用いて書けば以下のようになる.

$$\begin{aligned} T_{ij}(\boldsymbol{v})\, \varepsilon_{ij}(\boldsymbol{v}) =& \sum_{i=1}^{3}\sum_{i=1}^{3} T_{ij}(\boldsymbol{v})\, \varepsilon_{ij}(\boldsymbol{v}) \\ =& \quad T_{11}(\boldsymbol{v})\, \varepsilon_{11}(\boldsymbol{v}) + T_{12}(\boldsymbol{v})\, \varepsilon_{12}(\boldsymbol{v}) + T_{13}(\boldsymbol{v})\, \varepsilon_{13}(\boldsymbol{v}) \\ & + T_{21}(\boldsymbol{v})\, \varepsilon_{21}(\boldsymbol{v}) + T_{22}(\boldsymbol{v})\, \varepsilon_{22}(\boldsymbol{v}) + T_{23}(\boldsymbol{v})\, \varepsilon_{23}(\boldsymbol{v}) \\ & + T_{31}(\boldsymbol{v})\, \varepsilon_{31}(\boldsymbol{v}) + T_{32}(\boldsymbol{v})\, \varepsilon_{32}(\boldsymbol{v}) + T_{33}(\boldsymbol{v})\, \varepsilon_{33}(\boldsymbol{v}) \end{aligned} \tag{3.15}$$

なお, ダミーインデックス以外の添字, 例えば $A_{ij}\, B_{jk} = A_{i1}\, B_{1k} + A_{i2}\, B_{2k} + A_{i3}\, B_{3k}$ における i, k をフリーインデックスと呼ぶ. 通常の連続体力学の教科書では, ダミーインデックスが変化する範囲は 1 から 3 としているが, ここでは記述の簡略化のために, 混乱のない限りそれ以外の場合でも用いることにする.

$[M]$ は以下の $[V]$ と等価である.

$[V]$　与えられた表面力 $\boldsymbol{t}$ と体積力 $\rho\boldsymbol{g}$ に対し以下の条件を満たす $\boldsymbol{u} \in \boldsymbol{V}$ を求めよ.

$$\begin{aligned}\kappa\int_\Omega (\operatorname{div}\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u})\, \mathrm{d}\Omega + 2G\int_\Omega \varepsilon^D_{ij}(\boldsymbol{u})\varepsilon^D_{ij}(\delta\boldsymbol{u})\, \mathrm{d}\Omega & \\ = \int_{\partial\Omega} \delta\boldsymbol{u}\cdot\boldsymbol{t}\, \mathrm{d}S + \int_\Omega \rho\, \delta\boldsymbol{u}\cdot\boldsymbol{g}\, \mathrm{d}\Omega \qquad \forall\delta\boldsymbol{u} \in \boldsymbol{V} & \end{aligned} \tag{3.16}$$

この証明は以下のようになる. まず $[M]$ から $[V]$ を示す.

式 (3.13) から,

$$\boldsymbol{\Pi}(\boldsymbol{v}) - \boldsymbol{\Pi}(\boldsymbol{u}) \geq 0 \qquad \forall \boldsymbol{v} \in \boldsymbol{V} \tag{3.17}$$

が成立する. このとき, 任意の $\boldsymbol{v}$ に対し, $\boldsymbol{v} = \delta\boldsymbol{u} + \boldsymbol{u}$ なる $\delta\boldsymbol{u} \in \boldsymbol{V}$ が存在する[6]ので, 式 (3.17) は以下のように書き直すことができる.

$$\boldsymbol{\Pi}(\delta\boldsymbol{u} + \boldsymbol{u}) - \boldsymbol{\Pi}(\boldsymbol{u}) \geq 0 \qquad \forall \delta\boldsymbol{u} \in \boldsymbol{V} \tag{3.18}$$

式 (3.18) を式 (3.14) をもとに, 書き下せば以下のようになる.

$$\begin{aligned}
&\left\{ \frac{\kappa}{2} \int_\Omega \operatorname{div}(\delta\boldsymbol{u} + \boldsymbol{u}) \operatorname{div}(\delta\boldsymbol{u} + \boldsymbol{u}) \,\mathrm{d}\Omega \right. \\
&\quad \left. + G \int_\Omega \varepsilon_{ij}^D(\delta\boldsymbol{u} + \boldsymbol{u})\, \varepsilon_{ij}^D(\delta\boldsymbol{u} + \boldsymbol{u}) \,\mathrm{d}\Omega - \int_{\partial\Omega} \boldsymbol{t} \cdot (\delta\boldsymbol{u} + \boldsymbol{u}) \,\mathrm{d}S - \int_\Omega \rho\boldsymbol{g} \cdot (\delta\boldsymbol{u} + \boldsymbol{u}) \,\mathrm{d}\Omega \right\} \\
&\qquad - \left\{ \frac{\kappa}{2} \int_\Omega (\operatorname{div}\boldsymbol{u})(\operatorname{div}\boldsymbol{u}) \,\mathrm{d}\Omega - G \int_\Omega \varepsilon_{ij}^D(\boldsymbol{u})\, \varepsilon_{ij}^D(\boldsymbol{u}) \,\mathrm{d}\Omega \right. \\
&\qquad\quad \left. - \int_{\partial\Omega} \boldsymbol{t} \cdot \boldsymbol{u} \,\mathrm{d}S - \int_\Omega \rho\boldsymbol{g} \cdot \boldsymbol{u} \,\mathrm{d}\Omega \right\} \geq 0 \qquad \forall \delta\boldsymbol{u} \in \boldsymbol{V}
\end{aligned} \tag{3.19}$$

これを整理すると以下のようになる.

$$\begin{aligned}
&\kappa \int_\Omega (\operatorname{div}\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u}) \,\mathrm{d}\Omega + 2G \int_\Omega \varepsilon_{ij}^D(\boldsymbol{u})\, \varepsilon_{ij}^D(\delta\boldsymbol{u}) \,\mathrm{d}\Omega - \int_{\partial\Omega} \delta\boldsymbol{u} \cdot \boldsymbol{t} \,\mathrm{d}S - \int_\Omega \rho\, \delta\boldsymbol{u} \cdot \boldsymbol{g} \,\mathrm{d}\Omega \\
&\qquad + \frac{\kappa}{2} \int_\Omega (\operatorname{div}\delta\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u}) \,\mathrm{d}\Omega + G \int_\Omega \varepsilon_{ij}^D(\delta\boldsymbol{u})\, \varepsilon_{ij}^D(\delta\boldsymbol{u}) \,\mathrm{d}\Omega \geq 0 \qquad \forall \delta\boldsymbol{u} \in \boldsymbol{V}
\end{aligned} \tag{3.20}$$

式 (3.20) は任意の $\delta\boldsymbol{u}$ について成り立たなければならないので, α を任意のスカラーとして $\delta\boldsymbol{u}$ に $\alpha\,\delta\boldsymbol{u}$ を代入しても成り立たなければならない. このとき式 (3.20) は以下のように整理される.

$$\begin{aligned}
&\alpha \left\{ \kappa \int_\Omega (\operatorname{div}\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u}) \,\mathrm{d}\Omega + 2G \int_\Omega \varepsilon_{ij}^D(\boldsymbol{u})\, \varepsilon_{ij}^D(\delta\boldsymbol{u}) \,\mathrm{d}\Omega - \int_{\partial\Omega} \delta\boldsymbol{u} \cdot \boldsymbol{t} \,\mathrm{d}S - \int_\Omega \rho\, \delta\boldsymbol{u} \cdot \boldsymbol{g} \,\mathrm{d}\Omega \right\} \\
&\quad + \alpha^2 \left\{ \frac{\kappa}{2} \int_\Omega (\operatorname{div}\delta\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u}) \,\mathrm{d}\Omega + G \int_\Omega \varepsilon_{ij}^D(\delta\boldsymbol{u})\, \varepsilon_{ij}^D(\delta\boldsymbol{u}) \,\mathrm{d}\Omega \right\} \geq 0 \qquad \forall \delta\boldsymbol{u} \in \boldsymbol{V}
\end{aligned} \tag{3.21}$$

[6] $\delta\boldsymbol{u}$ の記号について: δ 自体には意味はなく, "$\delta\boldsymbol{u}$" で 1 文字として取扱う. 有限要素法の教科書にはこの $\delta\boldsymbol{u}$ は「仮想変位」として導入され, 物理的な意味が与えられている場合も多いが, 物理的意味がわかったからといって数学的な取扱いが簡略化できるわけでもないので, ここではあえて物理的意味には触れず, 純粋に数学的に取扱う. ただし, 記号だけは一致させることにする.

$\delta\boldsymbol{u} \equiv 0$ のとき, 式 (3.21) 左辺は 0 になるので式 (3.21) は成立する. $\delta\boldsymbol{u} \not\equiv 0$ のときは, 式 (3.21) は α についての 2 次式になっており, かつ α の係数は正である. 2 次方程式 $y = ax^2 + bx + c$ が $y \geq 0$ であるための必要十分条件は $\alpha > 0$ かつ $b^2 - 4ac \leq 0$ (判別式) であるが, 式 (3.21) の場合は $c = 0$ であり, $b = 0$ が必要十分条件となる. 即ち, 式 (3.16) が必要十分条件である.

$[V] \Rightarrow [M]$ については式 (3.16) を用いると, 以下のようになることから証明できる.

$$\begin{aligned}\Pi(\delta\boldsymbol{u}+\boldsymbol{u}) - \Pi(\boldsymbol{u}) =& \kappa\int_\Omega (\operatorname{div}\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u})\,\mathrm{d}\Omega + 2G\int_\Omega \varepsilon_{ij}^D(\boldsymbol{u})\,\varepsilon_{ij}^D(\delta\boldsymbol{u})\,\mathrm{d}\Omega - \int_{\partial\Omega}\delta\boldsymbol{u}\cdot\boldsymbol{t}\,\mathrm{d}S - \int_\Omega \rho\,\delta\boldsymbol{u}\cdot\boldsymbol{g}\,\mathrm{d}\Omega \\ &+ \frac{\kappa}{2}\int_\Omega (\operatorname{div}\delta\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u})\,\mathrm{d}\Omega + G\int_\Omega \varepsilon_{ij}^D(\delta\boldsymbol{u})\,\varepsilon_{ij}^D(\delta\boldsymbol{u})\,\mathrm{d}\Omega \\ =& \frac{\kappa}{2}\int_\Omega (\operatorname{div}\delta\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u})\,\mathrm{d}\Omega + G\int_\Omega \varepsilon_{ij}^D(\delta\boldsymbol{u})\,\varepsilon_{ij}^D(\delta\boldsymbol{u})\,\mathrm{d}\Omega \geq 0 \qquad \forall\delta\boldsymbol{u}\in\boldsymbol{V}\end{aligned} \tag{3.22}$$

次に $[V]$ と $[B]$ が等価であることを示す. $[V] \Rightarrow [B]$ を示すため, まず, 式 (3.16) の左辺が以下のように書き直せることを示す.

$$\kappa\int_\Omega (\operatorname{div}\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u})\,\mathrm{d}\Omega + 2G\int_\Omega \varepsilon_{ij}^D(\boldsymbol{u})\,\varepsilon_{ij}^D(\delta\boldsymbol{u})\,\mathrm{d}\Omega = \int_\Omega T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) \tag{3.23}$$

式 (3.5), 式 (3.7), 式 (3.8) により以下のようになる.

$$\begin{aligned}T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) &= \{\kappa(\operatorname{div}\boldsymbol{u})\delta_{ij} + 2G\varepsilon_{ij}^D(\boldsymbol{u})\}\,\varepsilon_{ij}(\delta\boldsymbol{u}) \\ &= \kappa(\operatorname{div}\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u}) + 2G\int_\Omega \varepsilon_{ij}^D(\boldsymbol{u})\,\varepsilon_{ij}^D(\delta\boldsymbol{u})\,\mathrm{d}\Omega\end{aligned} \tag{3.24}$$

よって, 式 (3.23) が成立する. $T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})$ の i, j についての対称性を用いると, 式 (3.23) の右辺は以下のように変形できる.

$$\begin{aligned}\int_\Omega T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})\,\mathrm{d}\Omega &= \int_\Omega T_{ij}\cdot\frac{1}{2}\left(\frac{\partial\delta u_i}{\partial x_j} + \frac{\partial\delta u_j}{\partial x_i}\right)\mathrm{d}\Omega \\ &= \int_\Omega T_{ij}\frac{\partial\delta u_i}{\partial x_j}\,\mathrm{d}\Omega \\ &= \int_\Omega \frac{\partial}{\partial x_j}(T_{ij}\delta u_i)\,\mathrm{d}\Omega - \int_\Omega \frac{\partial T_{ij}}{\partial x_j}\delta u_i\,\mathrm{d}\Omega\end{aligned} \tag{3.25}$$

式 (3.25) 左辺第一項に対し, Gauss の発散定理を適用すれば, 最終的に

$$\int_\Omega T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})\,\mathrm{d}\Omega = \int_{\partial\Omega} n_j(T_{ij}\delta u_i)\,\mathrm{d}S - \int_\Omega (\nabla_x\cdot\boldsymbol{T})\delta u_i\,\mathrm{d}\Omega \tag{3.26}$$

となる. これは外力項と合わせて, 以下のようになる.

$$\int_\Omega \delta u_i\big((\nabla_x\cdot\boldsymbol{T})_i - \rho g_i\big)\,\mathrm{d}\Omega + \int_{\partial\Omega}\delta u_i(\boldsymbol{T}^T\cdot\boldsymbol{n}-\boldsymbol{t})\,\mathrm{d}S = 0 \tag{3.27}$$

上式が成立するための条件が式 (3.2),(3.3) である. $[B]\Rightarrow[V]$ は この逆をたどることにより導くことができる. この方法を Galerkin 法と呼ぶ. また, 式 (3.16), (3.23) をあわせた

$$\int_\Omega T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})\,\mathrm{d}\Omega = \int_{\partial\Omega}\delta\boldsymbol{u}\cdot\boldsymbol{t}\,\mathrm{d}S + \int_\Omega \rho\,\delta\boldsymbol{u}\cdot\boldsymbol{g}\,\mathrm{d}\Omega \qquad \forall\delta\boldsymbol{u}\in\boldsymbol{V} \tag{3.28}$$

は仮想仕事の原理の式と呼ばれている.

3.1.3 線形弾性体の理論解 — 単純引張

次節以降で行う有限要素法の要素剛性マトリックスのプログラミングで最も大切なことは, コーディングする前に, 手計算でいくつかの理論解を求めておくことである. この理論解がなくてはデバッグは不可能である. 理論解でも不可欠なのは以下に示す単純引張と単純剪断で, この 2 つの問題が正しく解ければ要素剛性マトリックスのデバッグはほぼ完了したと考えて良い.

2.3 節で示した定式化では, 後述する混合型の定式化への拡張を考慮に入れて体積弾性係数 κ と剪断弾性係数 G を用いて定式化を行っているが, 通常の解析ではヤング率 E とポアソン比 ν を用いるのが一般的である. これらの材料定数には以下の関係がある.

$$\kappa = \frac{E}{3\,(1-2\nu)} \tag{3.29}$$

$$G = \frac{E}{2\,(1+\nu)} \tag{3.30}$$

図 3.2 に示すような 1 辺の長さ L の立方体の x_1 面 (x_2-x_3 平面に平行な面) に, x_1 面と垂直で大きさ P の引張力を作用させたときの変位を求めよ. ただし, 微小変形を仮定し, ヤング率を E, ポアソン比を ν とする.

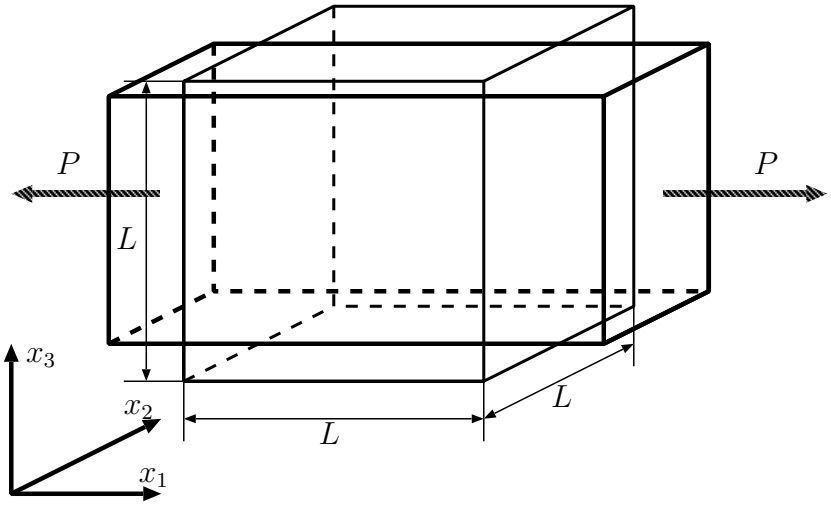


図 3.2: 単純引張

このような境界条件の例題を単純引張といって, 線形・非線形を問わず理論解が求められる数少ない例題である.

微小変形を仮定したときには荷重が作用している面の面積の変化は考慮しないので, 物体内に生じる応力は以下のようになる.

$$[T_{ij}] = \begin{bmatrix} \dfrac{P}{L^2} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.31}$$

一方, ひずみは変位を u_i として, 以下のようになる.

$$[\varepsilon_{ij}] = \begin{bmatrix} \dfrac{2u_1}{L} & 0 & 0 \\ 0 & \dfrac{2u_2}{L} & 0 \\ 0 & 0 & \dfrac{2u_3}{L} \end{bmatrix} \tag{3.32}$$

ただし, 変位は図 3.3 のように定義する. この例では $u_3 = u_2$ なので, 以下この関係を用い

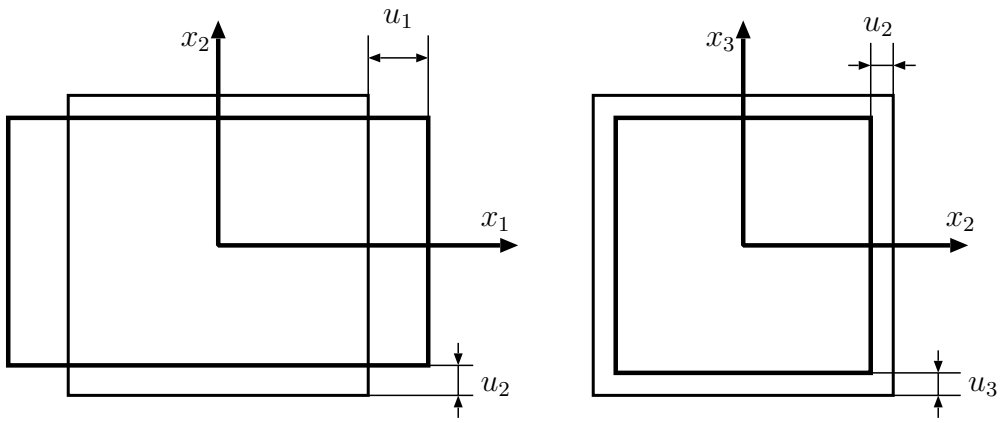


図 3.3: 変位の定義 (単純引張)

ることにする. 式 (3.5) の κ, G を E, ν に書き換えると以下のようになる.

$$\begin{aligned} T_{ij} &= \kappa\,(\operatorname{div}\boldsymbol{u})\,\delta_{ij} + 2G\varepsilon_{ij}^{D} \\ &= \frac{E}{3\,(1-2\nu)}(\operatorname{div}\boldsymbol{u})\,\delta_{ij} + \frac{E}{1+\nu}\left(\varepsilon_{ij} - \frac{1}{3}(\operatorname{div}\boldsymbol{u})\,\delta_{ij}\right) \\ &= \frac{E\nu}{(1-2\nu)(1+\nu)}(\operatorname{div}\boldsymbol{u})\,\delta_{ij} + \frac{E}{1+\nu}\varepsilon_{ij} \end{aligned} \tag{3.33}$$

これより応力とひずみの関係式は以下のようになる.

$$\begin{aligned} \begin{bmatrix} \dfrac{P}{L^2} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \frac{E\nu}{(1-2\nu)(1+\nu)} &\begin{bmatrix} \dfrac{2u_1+4u_2}{L} & 0 & 0 \\ 0 & \dfrac{2u_1+4u_2}{L} & 0 \\ 0 & 0 & \dfrac{2u_1+4u_2}{L} \end{bmatrix} \\ &+ \frac{E}{1+\nu}\begin{bmatrix} \dfrac{2u_1}{L} & 0 & 0 \\ 0 & \dfrac{2u_2}{L} & 0 \\ 0 & 0 & \dfrac{2u_2}{L} \end{bmatrix} \end{aligned} \tag{3.34}$$

これより下式の関係を得る.

$$\frac{P}{L^2} = \frac{E\nu}{(1-2\nu)(1+\nu)}\frac{2u_1+4u_2}{L} + \frac{E}{1+\nu}\frac{2u_1}{L} \tag{3.35}$$

$$0 = \frac{E\nu}{(1-2\nu)(1+\nu)}\frac{2u_1+4u_2}{L} + \frac{E}{1+\nu}\frac{2u_2}{L} \tag{3.36}$$

最終的に, u_1, u_2 は以下のようになる.

$$u_1 = \frac{P}{2EL} \tag{3.37}$$

$$u_2 = -\nu u_1 = -\frac{P\nu}{2EL} \tag{3.38}$$

3.1.4　線形弾性体の理論解 — 単純剪断

下図に示すような 1 辺の長さ L の立方体の x_3 面に, x_3 面と平行で x_1 方向の, 大きさ P の剪断力を作用させたときの変位を求めよ. ただし, 微小変形を仮定し, ヤング率を E, ポアソン比を ν とする.

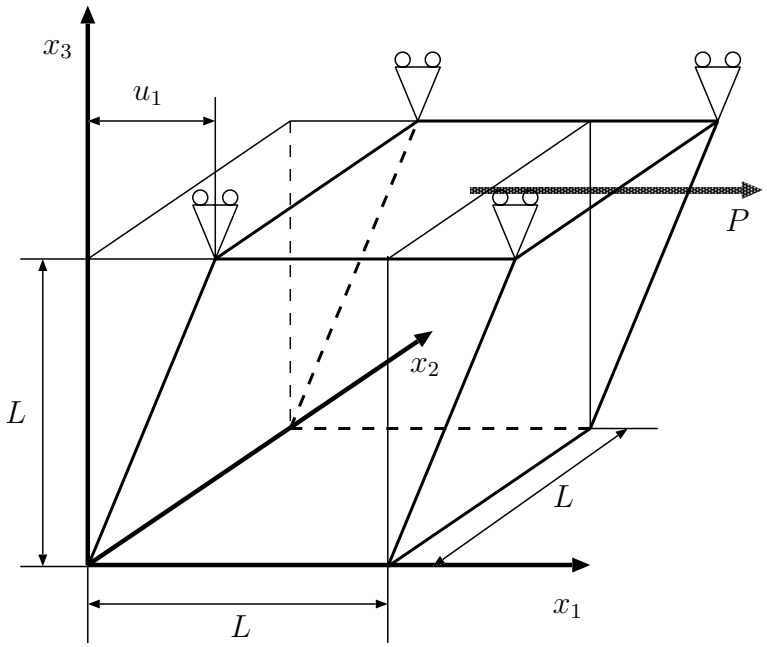


図 3.4: 単純剪断

このような境界条件の例題を単純剪断といって, やはり線形・非線形を問わず理論解が

求められる数少ない例題である. 物体内に生じる応力は以下のようになる.

$$[T_{ij}] = \begin{bmatrix} 0 & 0 & \dfrac{P}{L^2} \\ 0 & 0 & 0 \\ \dfrac{P}{L^2} & 0 & 0 \end{bmatrix} \tag{3.39}$$

一方ひずみは, 変位を u_i として以下のようになる.

$$[\varepsilon_{ij}] = \begin{bmatrix} 0 & 0 & \dfrac{u_1}{2L} \\ 0 & 0 & 0 \\ \dfrac{u_1}{2L} & 0 & 0 \end{bmatrix} \tag{3.40}$$

これより, 応力とひずみの関係式は以下のようになる.

$$\begin{bmatrix} 0 & 0 & \dfrac{P}{L^2} \\ 0 & 0 & 0 \\ \dfrac{P}{L^2} & 0 & 0 \end{bmatrix} = \frac{E}{1+\nu} \begin{bmatrix} 0 & 0 & \dfrac{u_1}{2L} \\ 0 & 0 & 0 \\ \dfrac{u_1}{2L} & 0 & 0 \end{bmatrix} \tag{3.41}$$

これより下式の関係を得る.

$$\frac{P}{L^2} = \frac{E}{1+\nu}\,\frac{u_1}{2L} \tag{3.42}$$

最終的に u_1 は以下のようになる.

$$u_1 = \frac{2\,(1+\nu)\,P}{EL} \tag{3.43}$$

3.2 有限要素定式化

3.1 節では線形弾性体の境界値問題は $[V]$ として弱形式定式化できることを示したが, ここではそれを基に有限要素定式化を行う. $[V]$ を再記すれば以下のようになる.

$[V]$ 与えられた外力表面力 $\boldsymbol{t}$ と体積力 $\rho\boldsymbol{g}$ に対し以下の条件を満たす $\boldsymbol{u} \in \boldsymbol{V}$ を求めよ.

$$\kappa \int_\Omega (\operatorname{div}\boldsymbol{u})(\operatorname{div}\delta\boldsymbol{u})\,\mathrm{d}\Omega + 2G \int_\Omega \varepsilon_{ij}^D(\boldsymbol{u})\,\varepsilon_{ij}^D(\delta\boldsymbol{u})\,\mathrm{d}\Omega = \int_{\partial\Omega} \delta\boldsymbol{u}\cdot\boldsymbol{t}\,\mathrm{d}S + \int_\Omega \rho\,\delta\boldsymbol{u}\cdot\boldsymbol{g}\,\mathrm{d}\Omega \qquad \forall\delta\boldsymbol{u} \in \boldsymbol{V} \tag{3.44}$$

式 (3.44) は, 以下のようにも書くことができるので, 以下の定式化では記述の簡略化のために式 (3.45) を用いる.

$$\int_{\Omega} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})\,\mathrm{d}\Omega = \int_{\partial\Omega} \delta\boldsymbol{u}\cdot\boldsymbol{t}\,\mathrm{d}S + \int_{\Omega} \rho\,\delta\boldsymbol{u}\cdot\boldsymbol{g}\,\mathrm{d}\Omega \qquad \forall\delta\boldsymbol{u}\in\boldsymbol{V} \tag{3.45}$$

3.2.1　有限要素分割と補間

2.1 節で述べたように, 有限要素法では解析対象の領域 Ω をいくつかの有限の大きさを持った要素に分割する. これを形式的に以下のように表す.

$$\Omega = \sum_{e} \Omega_e \tag{3.46}$$

これに伴い, 領域積分, 境界積分はそれぞれ次のようになる.

$$\int_{\Omega} \mathrm{d}\Omega = \sum_{e} \int_{\Omega_e} \mathrm{d}\Omega \tag{3.47}$$

$$\int_{\partial\Omega} \mathrm{d}S = \sum_{e} \int_{\partial\Omega_e} \mathrm{d}S \tag{3.48}$$

これより式 (3.45) は以下のように変更される.

$$\sum_{e}\left[\int_{\Omega_e} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})\,\mathrm{d}\Omega - \int_{\partial\Omega_e} \delta\boldsymbol{u}\cdot\boldsymbol{t}\,\mathrm{d}S - \int_{\Omega_e} \rho\,\delta\boldsymbol{u}\cdot\boldsymbol{g}\,\mathrm{d}\Omega\right] = 0 \qquad \forall\delta\boldsymbol{u}\in\boldsymbol{V} \tag{3.49}$$

式 (3.49) の被積分関数に含まれる $\boldsymbol{x}$, $\boldsymbol{u}$ は各要素内で同じ補間関数を用い, 以下のように表されるものとする.

$$x_i = N^{(j)} x_i^{(j)} \tag{3.50}$$

$$u_i = N^{(j)} u_i^{(j)} \tag{3.51}$$

以下では式 (3.49) を効率良く計算できるように, マトリックス表示に書き換えを行う. ここで示すようなマトリックス表示は, あくまでプログラミングのための手段であり, その意味では本質的ではなく, プログラマーが独自の方法で行ってよいものである. ここではなるべく標準的で, わかりやすく, 他のケースにも応用しやすい方法を示す.

3.2.2 応力-ひずみマトリックス ($[D]$ マトリックス)

式 (3.49) 左辺第 1 項の被積分関数 $T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})$ は, 総和規約を用いずに書けば以下のようになる.

$$\begin{aligned} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) = \quad & T_{11}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{12}(\boldsymbol{u})\,\varepsilon_{12}(\delta\boldsymbol{u}) + T_{13}(\boldsymbol{u})\,\varepsilon_{13}(\delta\boldsymbol{u}) \\ & + T_{21}(\boldsymbol{u})\,\varepsilon_{21}(\delta\boldsymbol{u}) + T_{22}(\boldsymbol{u})\,\varepsilon_{22}(\delta\boldsymbol{u}) + T_{23}(\boldsymbol{u})\,\varepsilon_{23}(\delta\boldsymbol{u}) \\ & + T_{31}(\boldsymbol{u})\,\varepsilon_{31}(\delta\boldsymbol{u}) + T_{32}(\boldsymbol{u})\,\varepsilon_{32}(\delta\boldsymbol{u}) + T_{33}(\boldsymbol{u})\,\varepsilon_{33}(\delta\boldsymbol{u}) \end{aligned} \tag{3.52}$$

式 (3.52) の右辺をなるべく少ない演算回数で計算できるように, T_{ij}, ε_{ij} の i, j についての対称性を用いて以下のように整理する.

$$\begin{aligned} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) = \quad & T_{11}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + \ T_{22}(\boldsymbol{u})\,\varepsilon_{22}(\delta\boldsymbol{u}) + \ T_{33}(\boldsymbol{u})\,\varepsilon_{33}(\delta\boldsymbol{u}) \\ & + 2T_{12}(\boldsymbol{u})\,\varepsilon_{12}(\delta\boldsymbol{u}) + 2T_{23}(\boldsymbol{u})\,\varepsilon_{23}(\delta\boldsymbol{u}) + 2T_{31}(\boldsymbol{u})\,\varepsilon_{31}(\delta\boldsymbol{u}) \\ = & \{\varepsilon(\delta\boldsymbol{u})\}^T\{T(\boldsymbol{u})\} \end{aligned} \tag{3.53}$$

ただし, $\{\varepsilon(\boldsymbol{v})\}$, $\{T(\boldsymbol{v})\}$ は下式により定義される.

$$\{\varepsilon(\boldsymbol{v})\} = \begin{Bmatrix} \varepsilon_{11}(\boldsymbol{v}) \\ \varepsilon_{22}(\boldsymbol{v}) \\ \varepsilon_{33}(\boldsymbol{v}) \\ 2\varepsilon_{12}(\boldsymbol{v}) \\ 2\varepsilon_{23}(\boldsymbol{v}) \\ 2\varepsilon_{31}(\boldsymbol{v}) \end{Bmatrix}, \quad \{T(\boldsymbol{v})\} = \begin{Bmatrix} T_{11}(\boldsymbol{v}) \\ T_{22}(\boldsymbol{v}) \\ T_{33}(\boldsymbol{v}) \\ T_{12}(\boldsymbol{v}) \\ T_{23}(\boldsymbol{v}) \\ T_{31}(\boldsymbol{v}) \end{Bmatrix} \tag{3.54}$$

式 (3.53) の右辺で非対角項の和には係数 2 がついているが, これを応力のベクトル表示にではなくひずみのベクトル表示に含めるのは, 歴史的に ε_{ij} の対角項はそのままに非対角項を 2 倍にしたものが, 工学ひずみとして使われてきているからである. その意味で $\{\varepsilon(\boldsymbol{v})\}$ は, 工学ひずみベクトルといえる.

次に応力 T_{ij} が, ひずみ ε_{ij} と式 (3.5) の関係にあることを用いて $\{T(\boldsymbol{v})\}$ と $\{\varepsilon(\boldsymbol{v})\}$ を下式のようにマトリックスとベクトルの積の形式で対応させる. このマトリックス $[D]$ は応力 - ひずみマトリックス, あるいは構成則マトリックス, あるいは単に $[D]$ マトリック

スと呼ばれている.

$$\{T(\boldsymbol{v})\} = [D]\{\varepsilon(\boldsymbol{v})\} \tag{3.55}$$

$\{T_{ij}(\boldsymbol{v})\}$ に含まれている T_{ij} の成分をそれぞれ書き下せば以下のようになる.

$$T_{11} = \kappa\,(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) + 2G\varepsilon_{11} - \frac{2G}{3}\,(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) \tag{3.56}$$

$$T_{22} = \kappa\,(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) + 2G\varepsilon_{22} - \frac{2G}{3}\,(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) \tag{3.57}$$

$$T_{33} = \kappa\,(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) + 2G\varepsilon_{33} - \frac{2G}{3}\,(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) \tag{3.58}$$

$$T_{12} = 2G\varepsilon_{12} \tag{3.59}$$

$$T_{23} = 2G\varepsilon_{23} \tag{3.60}$$

$$T_{31} = 2G\varepsilon_{31} \tag{3.61}$$

これを強引にマトリックス表示に書き直すと以下のようになる.

$$\begin{Bmatrix} T_{11} \\ T_{22} \\ T_{33} \\ T_{12} \\ T_{23} \\ T_{31} \end{Bmatrix} = \begin{bmatrix} \kappa & \kappa & \kappa & & & \\ \kappa & \kappa & \kappa & & 0 & \\ \kappa & \kappa & \kappa & & & \\ & & & & & \\ & 0 & & & & \\ & & & & & \end{bmatrix} \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ 2\varepsilon_{12} \\ 2\varepsilon_{23} \\ 2\varepsilon_{31} \end{Bmatrix} + \begin{bmatrix} \frac{4}{3}G & -\frac{2}{3}G & -\frac{2}{3}G & & & \\ -\frac{2}{3}G & \frac{4}{3}G & -\frac{2}{3}G & & 0 & \\ -\frac{2}{3}G & -\frac{2}{3}G & \frac{4}{3}G & & & \\ & & & G & & \\ & 0 & & & G & \\ & & & & & G \end{bmatrix} \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ 2\varepsilon_{12} \\ 2\varepsilon_{23} \\ 2\varepsilon_{31} \end{Bmatrix} \tag{3.62}$$

後述する混合法への拡張を考えて, 以下のように $[D_v]$, $[D_d]$ を定義する.

$$[D_v] = \begin{bmatrix} \kappa & \kappa & \kappa & & & \\ \kappa & \kappa & \kappa & & 0 & \\ \kappa & \kappa & \kappa & & & \\ & & & & & \\ & 0 & & & & \\ & & & & & \end{bmatrix} \tag{3.63}$$

$$[D_d] = \begin{bmatrix} \frac{4}{3}G & -\frac{2}{3}G & -\frac{2}{3}G & & & \\ -\frac{2}{3}G & \frac{4}{3}G & -\frac{2}{3}G & & \mathbf{0} & \\ -\frac{2}{3}G & -\frac{2}{3}G & \frac{4}{3}G & & & \\ & & & G & & \\ & \mathbf{0} & & & G & \\ & & & & & G \end{bmatrix} \tag{3.64}$$

これらを用い, 下式のように $[D]$ を定義すれば, 式 (3.55) の形式で対応付けを行うことができる.

$$[D] = [D_v] + [D_d] \tag{3.65}$$

以上を用いると, 式 (3.49) 左辺第 1 項の被積分関数 $T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})$ は以下のように表すことができる.

$$T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) = \{\varepsilon(\delta\boldsymbol{u})\}^T\,[D]\,\{\varepsilon(\boldsymbol{u})\} \tag{3.66}$$

3.2.3 節点変位-ひずみマトリックス ($[B]$ マトリックス)

変位と線形ひずみは式 (3.4) のように関係付けられ, 変位と節点変位は式 (3.50) のように関係付けられている. これらを総合し線形ひずみと節点変位を以下のようにマトリックスとベクトルの積の形式で対応させる. このマトリックス $[B]$ は, 節点変位-ひずみマトリックス, あるいは単に $[B]$ マトリックスと呼ばれている. ただし n は一要素あたりの節点数である.

$$\{\varepsilon(\boldsymbol{u})\} = [B]\{u_i^{(n)}\} \tag{3.67}$$

ただし, $\{u_i^{(n)}\}$ は下式により定義される.

$$\{u_i^{(n)}\} = \begin{Bmatrix} u_1^{(1)} \\ u_2^{(1)} \\ u_3^{(1)} \\ u_1^{(2)} \\ u_2^{(2)} \\ u_3^{(2)} \\ \vdots \\ u_1^{(n)} \\ u_2^{(n)} \\ u_3^{(n)} \end{Bmatrix} \tag{3.68}$$

ひずみの計算に必要な $\dfrac{\partial u_i}{\partial x_j}$ は, 節点変位が位置ベクトル $\boldsymbol{x}$ には依存しない量であることから, 次のようになる.

$$\frac{\partial u_i}{\partial x_j} = \frac{\partial N^{(n)}}{\partial x_j}\, u_i^{(n)} \tag{3.69}$$

これを基にすると式 (3.67) のマトリックス $[B]$ は, 式 (3.70) に示す 6×3 のサブマトリックス $[B^{(k)}]$ を用いて式 (3.71) のようにあらわされる.

$$[B^{(k)}] = \begin{bmatrix} \dfrac{\partial N^{(k)}}{\partial x_1} & & \\ & \dfrac{\partial N^{(k)}}{\partial x_2} & \\ & & \dfrac{\partial N^{(k)}}{\partial x_3} \\ \dfrac{\partial N^{(k)}}{\partial x_2} & \dfrac{\partial N^{(k)}}{\partial x_1} & \\ & \dfrac{\partial N^{(k)}}{\partial x_3} & \dfrac{\partial N^{(k)}}{\partial x_2} \\ \dfrac{\partial N^{(k)}}{\partial x_3} & & \dfrac{\partial N^{(k)}}{\partial x_1} \end{bmatrix} \tag{3.70}$$

$$[B] = \left[[B^{(1)}][B^{(2)}] \dots [B^{(n)}] \right] \tag{3.71}$$

この $[B]$ を用いると, 式 (3.49) 左辺第 1 項の被積分関数 $T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})$ は以下のように表すことができる.

$$\begin{aligned} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) &= [\varepsilon(\delta\boldsymbol{u})]^T\,[D]\,[\varepsilon(\boldsymbol{u})] \\ &= \{\delta u_i^{(n)}\}^T\,[B]^T\,[D]\,[B]\,\{u_i^{(n)}\} \end{aligned} \tag{3.72}$$

$\{\delta u_i^{(n)}\}$, $\{u_i^{(n)}\}$ は節点での値であり, 領域内では定数となり領域積分には関係ないので, 積分記号の外に出すことができる. 即ち,

$$\int_{\Omega_e} \{\delta u_i^{(n)}\}^T\,[B]^T\,[D]\,[B]\,\{u_i^{(n)}\}\,\mathrm{d}\Omega = \{\delta u_i^{(n)}\}^T \left[\int_{\Omega_e} [B]^T\,[D]\,[B]\,\mathrm{d}\Omega\right] \{u_i^{(n)}\} \tag{3.73}$$

この積分されたマトリックス即ち,

$$[K^{(e)}] = \int_{\Omega_e} [B]^T\,[D]\,[B]\,\mathrm{d}\Omega \tag{3.74}$$

を要素剛性マトリックスという.

3.2.4 外力ベクトル

式 (3.49) 左辺第 2, 3 項についても, 節点変位のベクトルを積分記号の外に出すために以下のようにする.

$$\begin{aligned} \int_{\partial\Omega_e} \delta u_i \cdot t_i\,\mathrm{d}S &= \int_{\partial\Omega_e} \{\delta u_i^{(n)}\}^T\,[N]^T\,\{t\}\,\mathrm{d}S \\ &= \{\delta u_i^{(n)}\}^T \int_{\partial\Omega_e} [N]^T\,\{t\}\,\mathrm{d}S \end{aligned} \tag{3.75}$$

$$\begin{aligned} \int_{\Omega_e} \rho\,\delta u_i \cdot g_i\,\mathrm{d}\Omega &= \int_{\Omega_e} \rho\,\{\delta u_i^{(n)}\}^T\,[N]^T\,\{g\}\,\mathrm{d}\Omega \\ &= \{\delta u_i^{(n)}\}^T \int_{\Omega_e} \rho\,[N]^T\,\{g\}\,\mathrm{d}\Omega \end{aligned} \tag{3.76}$$

ただし

$$[N] = \begin{bmatrix} N^{(1)} & & & N^{(2)} & & & \cdots & N^{(n)} & \\ & N^{(1)} & & & N^{(2)} & & \cdots & & N^{(n)} \\ & & N^{(1)} & & & N^{(2)} & & & N^{(n)} \end{bmatrix} \tag{3.77}$$

$$\{t\} = \begin{Bmatrix} t_1 \\ t_2 \\ t_3 \end{Bmatrix} \tag{3.78}$$

$$\{g\} = \begin{Bmatrix} g_1 \\ g_2 \\ g_3 \end{Bmatrix} \tag{3.79}$$

これらを用い外力ベクトル $\{F^{(e)}\}$ を以下のように定義する.

$$\{F^{(e)}\} = \int_{\partial\Omega_e} [N]^T \{t\} \, \mathrm{d}S + \int_{\Omega_e} \rho \, [N]^T \{g\} \, \mathrm{d}\Omega \tag{3.80}$$

以上をまとめると, 式 (3.49) は以下のように変更される.

$$\sum_e \left[\{\delta u_i^{(n)}\}^T \left([K^{(e)}] \{u_i^{(n)}\} - \{F^{(e)}\} \right) \right] = 0 \tag{3.81}$$

2.1 節で行ったように式 (3.81) の左辺を $\sum$ を用いず, また $\{\delta u_i^{(n)}\}$, $\{u_i^{(n)}\}$ は要素毎になっているのを全体の節点番号につけ直されたものに変更すると, 最終的には以下のような形式の連立 1 次方程式を解けばよいことがわかる. 構造解析の場合, この方程式は剛性方程式と呼ばれることもある.

$$\begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & & & \vdots \\ \vdots & & & \vdots \\ K_{n1} & & \cdots & K_{nn} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{Bmatrix} \tag{3.82}$$

この係数マトリックスを全体マトリックスという. 特に今回のような構造解析を行うためのものであれば, 全体剛性マトリックス, あるいは単に剛性マトリックスという. また右辺のベクトルは外力ベクトルという.

3.3 2 次元の有限要素定式化

構造解析を行う際には, ある条件の下で問題を単純化できる場合がある. 例えば前述の単純引張や単純剪断ならば, 3 次元的な定式化を行っているにも関わらず 1 変数しかない. これほど単純化できるのは上記の 2 例ぐらいしかないが, 2 変数の問題に単純化できる場合は多い. この例としては, 平面ひずみ問題と平面応力問題がある.

平面ひずみ問題とは, 例えば図 3.5 に示すような十分に長い壁が, 長手方向に垂直で均一な荷重を受けている場合である. このときに, 壁の長手方向中央部分を取り出すと x_2 方向の変位は 0 かつ x_2 方向の微分も 0 である. 従って, ひずみ ε_{ij} の 9 成分のうち ε_{22}, ε_{12}, ε_{21}, ε_{23}, ε_{32} は 0 になり有限要素でモデル化する際にも, 全節点で x_2 方向の変位は 0 である.

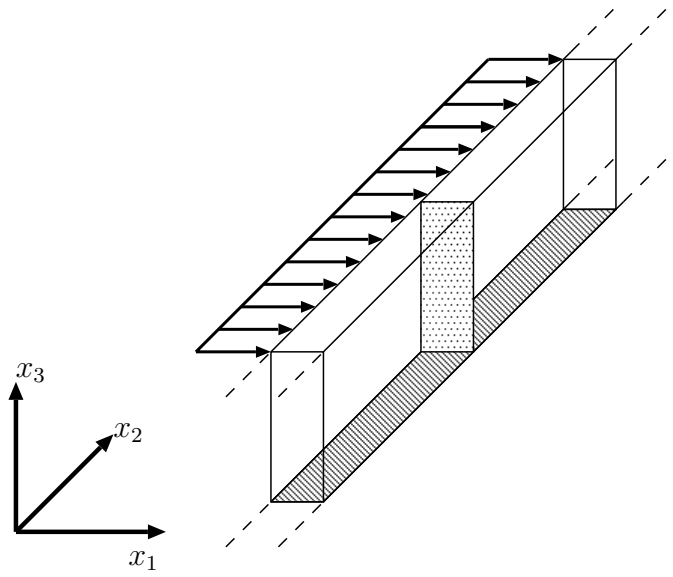


図 3.5: 平面ひずみ問題の例

平面応力問題とは, 平面ひずみ問題と逆に, 十分に板厚が薄い場合である. このとき, 板厚方向の応力は 0 かつ板厚面外の剪断応力が 0 である. 即ち図 3.6 の例では T_{22}, T_{12}, T_{21}, T_{23}, T_{32} が 0 になる.

平面ひずみ問題でも平面応力問題でも, 解くべき方程式は式 (3.49) で, ひずみあるいは

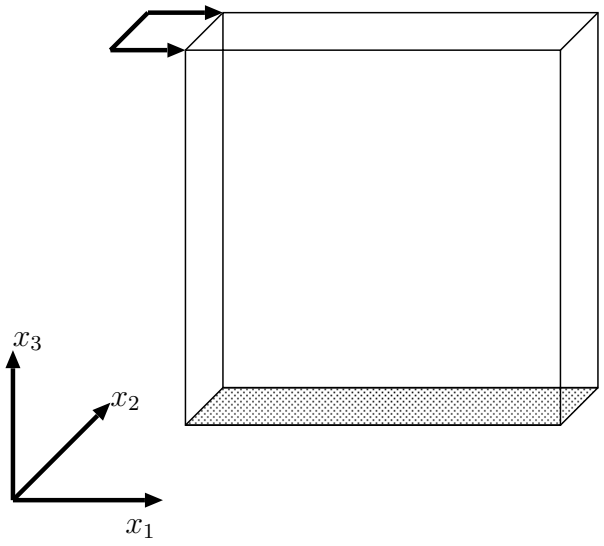


図 3.6: 平面応力問題の例

応力 9 成分のうち 5 成分まで 0 であることを用いて効率化されるだけである.

3.3.1　応力-ひずみマトリックス ($[D]$ マトリックス) — 平面ひずみ問題

まず平面ひずみ問題の場合, 式 (3.49) 左辺第 1 項の被積分関数 $T_{ij}(\boldsymbol{u})\,\varepsilon(\delta\boldsymbol{u})$ は総和規則を用いずに書けば以下のようになる.

$$\begin{aligned} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) = \quad & T_{11}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{12}(\boldsymbol{u})\,\varepsilon_{12}(\delta\boldsymbol{u}) + T_{13}(\boldsymbol{u})\,\varepsilon_{13}(\delta\boldsymbol{u}) \\ & + T_{21}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{22}(\boldsymbol{u})\,\varepsilon_{22}(\delta\boldsymbol{u}) + T_{23}(\boldsymbol{u})\,\varepsilon_{23}(\delta\boldsymbol{u}) \\ & + T_{31}(\boldsymbol{u})\,\varepsilon_{31}(\delta\boldsymbol{u}) + T_{32}(\boldsymbol{u})\,\varepsilon_{32}(\delta\boldsymbol{u}) + T_{33}(\boldsymbol{u})\,\varepsilon_{33}(\delta\boldsymbol{u}) \end{aligned} \tag{3.83}$$

ここで x_3 方向の変位 0, かつ x_3 方向の微分が 0 であると仮定すれば $\varepsilon_{33} = \varepsilon_{31} = \varepsilon_{13} = \varepsilon_{23} = \varepsilon_{32} = 0$ である. これを用いると式 (3.83) は以下のように簡略化される.

$$\begin{aligned} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) = \quad & T_{11}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{12}(\boldsymbol{u})\,\varepsilon_{12}(\delta\boldsymbol{u}) \\ & + T_{21}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{22}(\boldsymbol{u})\,\varepsilon_{22}(\delta\boldsymbol{u}) \end{aligned} \tag{3.84}$$

式 (3.84) の右辺をなるべく少ない演算回数で計算できるように, $T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})$ の i, j についての対称性を用いて以下のように整理する.

$$\begin{aligned} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) =& T_{11}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{22}(\boldsymbol{u})\,\varepsilon_{22}(\delta\boldsymbol{u}) + 2T_{12}(\boldsymbol{u})\,\varepsilon_{12}(\delta\boldsymbol{u}) \\ =& \{\varepsilon(\delta\boldsymbol{u})\}^T\{T(\boldsymbol{u})\} \end{aligned} \tag{3.85}$$

ただし $\{\varepsilon(\boldsymbol{v})\}$, $\{T(\boldsymbol{v})\}$ は下式により定義される.

$$\{\varepsilon(\boldsymbol{v})\} = \begin{Bmatrix} \varepsilon_{11}(\boldsymbol{v}) \\ \varepsilon_{22}(\boldsymbol{v}) \\ \varepsilon_{12}(\boldsymbol{v}) \end{Bmatrix} \tag{3.86}$$

$$\{T(\boldsymbol{v})\} = \begin{Bmatrix} \{T_{11}(\boldsymbol{v})\} \\ \{T_{22}(\boldsymbol{v})\} \\ \{T_{12}(\boldsymbol{v})\} \end{Bmatrix} \tag{3.87}$$

次に応力 T_{ij} がひずみ ε_{ij} と式 (3.5) の関係にあることを用いて, $\{T(\boldsymbol{u})\}$ と $\{\varepsilon(\boldsymbol{u})\}$ を下式のようにマトリックスとベクトルの積の形で対応させる.

$$\{T(\boldsymbol{u})\} = [D]\{\varepsilon(\boldsymbol{u})\} \tag{3.88}$$

$\{T(\boldsymbol{u})\}$ に含まれている T_{ij} の成分をそれぞれ書き下せば以下のようになる.

$$T_{11} = \kappa(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) + 2G\varepsilon_{11} - \frac{2G}{3}(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) \tag{3.89}$$

$$T_{22} = \kappa(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) + 2G\varepsilon_{22} - \frac{2G}{3}(\varepsilon_{11}+\varepsilon_{22}+\varepsilon_{33}) \tag{3.90}$$

$$T_{12} = 2G\varepsilon_{12} \tag{3.91}$$

ここで $\varepsilon_{33} = \varepsilon_{31} = \varepsilon_{13} = \varepsilon_{23} = \varepsilon_{32} = 0$ を代入すると, 以下のようなる.

$$T_{11} = \kappa(\varepsilon_{11}+\varepsilon_{22}) + 2G\varepsilon_{11} - \frac{2G}{3}(\varepsilon_{11}+\varepsilon_{22}) \tag{3.92}$$

$$T_{22} = \kappa(\varepsilon_{11}+\varepsilon_{22}) + 2G\varepsilon_{22} - \frac{2G}{3}(\varepsilon_{11}+\varepsilon_{22}) \tag{3.93}$$

$$T_{12} = 2G\varepsilon_{12} \tag{3.94}$$

これを強引にマトリックス表示に書き直すと以下のようになる.

$$\begin{Bmatrix} T_{11} \\ T_{22} \\ T_{12} \end{Bmatrix} = \begin{bmatrix} \kappa & \kappa & 0 \\ \kappa & \kappa & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{Bmatrix} + \begin{bmatrix} \dfrac{4G}{3} & -\dfrac{2G}{3} & 0 \\ -\dfrac{2G}{3} & \dfrac{4G}{3} & 0 \\ 0 & 0 & G \end{bmatrix} \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{Bmatrix} \tag{3.95}$$

後述する混合法への拡張を考えて以下のように $[D_v]$, $[D_d]$ を定義する.

$$[D_v] = \begin{bmatrix} \kappa & \kappa & 0 \\ \kappa & \kappa & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.96}$$

$$[D_d] = \begin{bmatrix} \dfrac{4G}{3} & -\dfrac{2G}{3} & 0 \\ -\dfrac{2G}{3} & \dfrac{4G}{3} & 0 \\ 0 & 0 & G \end{bmatrix} \tag{3.97}$$

これらを用い下式のように $[D]$ を定義すれば, 式 (3.88) の形式で対応付けを行うことができる.

$$[D] = [D_v] + [D_d] \tag{3.98}$$

以上を用いると, 式 (3.49) 左辺第 1 項の被積分関数 $T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})$ は以下のように表すことができる.

$$T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) = \{\varepsilon(\delta\boldsymbol{u})\}^T [D]\,\{\varepsilon(\boldsymbol{u})\} \tag{3.99}$$

3.3.2　応力-ひずみマトリックス ($[D]$ マトリックス) — 平面応力問題

次に平面応力の場合, 式 (3.49) 左辺第 1 項の被積分関数 $T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})$ は総和規約を用いずに書けば以下のようになる.

$$\begin{aligned} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) = \quad & T_{11}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{12}(\boldsymbol{u})\,\varepsilon_{12}(\delta\boldsymbol{u}) + T_{13}(\boldsymbol{u})\,\varepsilon_{13}(\delta\boldsymbol{u}) \\ & + T_{21}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{22}(\boldsymbol{u})\,\varepsilon_{22}(\delta\boldsymbol{u}) + T_{23}(\boldsymbol{u})\,\varepsilon_{23}(\delta\boldsymbol{u}) \\ & + T_{31}(\boldsymbol{u})\,\varepsilon_{31}(\delta\boldsymbol{u}) + T_{32}(\boldsymbol{u})\,\varepsilon_{32}(\delta\boldsymbol{u}) + T_{33}(\boldsymbol{u})\,\varepsilon_{33}(\delta\boldsymbol{u}) \end{aligned} \tag{3.100}$$

ここで x_3 方向に平面応力状態にあると仮定すれば, $T_{33} = T_{31} = T_{13} = T_{23} = T_{32} = 0$ である. これを用いると式 (3.100) は以下のように簡略化される.

$$\begin{aligned} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) = \quad & T_{11}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{12}(\boldsymbol{u})\,\varepsilon_{12}(\delta\boldsymbol{u}) \\ & + T_{21}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{22}(\boldsymbol{u})\,\varepsilon_{22}(\delta\boldsymbol{u}) \end{aligned} \tag{3.101}$$

式 (3.101) の右辺をなるべく少ない演算回数で計算できるように $T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})$ の i, j についての対称性を用いて以下のように整理する.

$$\begin{aligned} T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) =& T_{11}(\boldsymbol{u})\,\varepsilon_{11}(\delta\boldsymbol{u}) + T_{22}(\boldsymbol{u})\,\varepsilon_{22}(\delta\boldsymbol{u}) + 2T_{12}(\boldsymbol{u})\,\varepsilon_{12}(\delta\boldsymbol{u}) \\ =& \{\varepsilon(\delta\boldsymbol{u})\}^T\{T(\boldsymbol{u})\} \end{aligned} \tag{3.102}$$

ただし $\{\varepsilon(\boldsymbol{v})\}$, $\{T(\boldsymbol{v})\}$ は下式により定義される.

$$\{\varepsilon(\boldsymbol{v})\} = \begin{Bmatrix} \varepsilon_{11}(\boldsymbol{v}) \\ \varepsilon_{22}(\boldsymbol{v}) \\ \varepsilon_{12}(\boldsymbol{v}) \end{Bmatrix} \tag{3.103}$$

$$\{T(\boldsymbol{v})\} = \begin{Bmatrix} \{T_{11}(\boldsymbol{v})\} \\ \{T_{22}(\boldsymbol{v})\} \\ \{T_{12}(\boldsymbol{v})\} \end{Bmatrix} \tag{3.104}$$

次に応力 T_{ij} がひずみ ε_{ij} と式 (3.5) の関係にあることを用いて $\{T(\boldsymbol{u})\}$ と $\{\varepsilon(\boldsymbol{u})\}$ を下式のようにマトリックスとベクトルの積の形で対応させる.

$$\{T(\boldsymbol{u})\} = [D]\{\varepsilon(\boldsymbol{u})\} \tag{3.105}$$

$\{T(\boldsymbol{u})\}$ に含まれている T_{ij} の成分をそれぞれ書き下せば以下のようになる.

$$T_{11} = \kappa\,(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}) + 2G\varepsilon_{11} - \frac{2G}{3}\,(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}) \tag{3.106}$$

$$T_{22} = \kappa\,(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}) + 2G\varepsilon_{22} - \frac{2G}{3}\,(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}) \tag{3.107}$$

$$T_{33} = \kappa\,(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}) + 2G\varepsilon_{33} - \frac{2G}{3}\,(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}) \tag{3.108}$$

$$T_{12} = 2G\varepsilon_{12} \tag{3.109}$$

$$T_{23} = 2G\varepsilon_{23} \tag{3.110}$$

$$T_{31} = 2G\varepsilon_{31} \tag{3.111}$$

ここで $T_{33} = T_{23} = T_{31} = 0$ を用いると

$$\kappa\left(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}\right) + 2G\varepsilon_{33} - \frac{2G}{3}\left(\varepsilon_{11} + \varepsilon_{22} + \varepsilon_{33}\right) = 0 \tag{3.112}$$

$$\varepsilon_{23} = 0 \tag{3.113}$$

$$\varepsilon_{31} = 0 \tag{3.114}$$

式 (3.112) からさらに変形すると以下の関係が得られる.

$$\varepsilon_{33} = \frac{\dfrac{2G}{3} - \kappa}{\dfrac{4G}{3} + \kappa}\left(\varepsilon_{11} + \varepsilon_{22}\right) \tag{3.115}$$

式 (3.115) を式 (3.106), (3.107) に代入すると以下の関係が得られる.

$$T_{11} = \frac{2G\left(\dfrac{2G}{3} + 2\kappa\right)}{\dfrac{4G}{3} + \kappa}\varepsilon_{11} + \frac{2G\left(\kappa - \dfrac{2G}{3}\right)}{\dfrac{4G}{3} + \kappa}\varepsilon_{22} \tag{3.116}$$

$$T_{22} = \frac{2G\left(\kappa - \dfrac{2G}{3}\right)}{\dfrac{4G}{3} + \kappa}\varepsilon_{11} + \frac{2G\left(\dfrac{2G}{3} + 2\kappa\right)}{\dfrac{4G}{3} + \kappa}\varepsilon_{22} \tag{3.117}$$

式 (3.116), (3.117), (3.109) を強引にマトリックス表示すると以下のようになる.

$$\begin{Bmatrix} T_{11} \\ T_{22} \\ T_{12} \end{Bmatrix} = \begin{bmatrix} \dfrac{2G\left(\dfrac{2G}{3} + 2\kappa\right)}{\dfrac{4G}{3} + \kappa} & \dfrac{2G\left(\kappa - \dfrac{2G}{3}\right)}{\dfrac{4G}{3} + \kappa} & 0 \\ \dfrac{2G\left(\kappa - \dfrac{2G}{3}\right)}{\dfrac{4G}{3} + \kappa} & \dfrac{2G\left(\dfrac{2G}{3} + 2\kappa\right)}{\dfrac{4G}{3} + \kappa} & 0 \\ 0 & 0 & G \end{bmatrix} \begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ 2\varepsilon_{12} \end{Bmatrix} \tag{3.118}$$

以上を用いると式 (3.119) のように $[D]$ を定義すれば式 (3.105) の形式で対応付けを行うことができ, 式 (3.49) 左辺第 1 項の被積分関数 $T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u})$ は式 (3.120) のように表す

ことができる.

$$[D] = \begin{bmatrix} \dfrac{2G\left(\dfrac{2G}{3}+2\kappa\right)}{\dfrac{4G}{3}+\kappa} & \dfrac{2G\left(\kappa-\dfrac{2G}{3}\right)}{\dfrac{4G}{3}+\kappa} & 0 \\ \dfrac{2G\left(\kappa-\dfrac{2G}{3}\right)}{\dfrac{4G}{3}+\kappa} & \dfrac{2G\left(\dfrac{2G}{3}+2\kappa\right)}{\dfrac{4G}{3}+\kappa} & 0 \\ 0 & 0 & G \end{bmatrix} \tag{3.119}$$

$$T_{ij}(\boldsymbol{u})\,\varepsilon_{ij}(\delta\boldsymbol{u}) = \{\varepsilon(\delta\boldsymbol{u})\}^T[D]\,\{\varepsilon(\boldsymbol{u})\} \tag{3.120}$$

3.3.3 節点変位-ひずみマトリックス ($[B]$ マトリックス)

変位と線形ひずみは式 (3.4) のように関係付けられ, 変化と節点変位は式 (3.50) のように関係付けられている. さらに平面ひずみ, あるいは平面応力を仮定した場合, 式 (3.99), (3.120) に用いられているひずみのベクトル表示 $\{\varepsilon(\boldsymbol{v})\}$ には ε_{11}, ε_{22}, ε_{12} 成分しか含まれておらず, x_3 方向の変位は使われていない. これらを総合し, 線形ひずみと節点変位を以下のようにマトリックスとベクトルの積の形で対応させる.

$$\{\varepsilon(\boldsymbol{u})\} = [B]\,\{u_i^{(n)}\} \tag{3.121}$$

ただし $\{u_i^{(n)}\}$ は下式により定義される.

$$\{u_i^{(n)}\} = \begin{Bmatrix} u_1^{(1)} \\ u_2^{(1)} \\ u_1^{(2)} \\ u_2^{(2)} \\ \vdots \\ u_1^{(n)} \\ u_2^{(n)} \end{Bmatrix} \tag{3.122}$$

ひずみの計算に必要な $\dfrac{\partial u_i}{\partial x_j}$ は, 節点変位が位置ベクトル $\boldsymbol{x}$ には依存しない量であることから, 次のようになる.

$$\frac{\partial u_i}{\partial x_j} = \frac{\partial N^{(n)}}{\partial x_j}\, u_i^{(n)} \tag{3.123}$$

これを基にすると式 (3.121) のマトリックス $[B]$ は, 式 (3.124) にしめす 3×2 のサブマトリックス $[B^{(k)}]$ を用いて式 (3.125) のようにあらわされる.

$$[B^{(k)}] = \begin{bmatrix} \dfrac{\partial N^{(k)}}{\partial x_1} & \\ & \dfrac{\partial N^{(k)}}{\partial x_2} \\ \dfrac{\partial N^{(k)}}{\partial x_2} & \dfrac{\partial N^{(k)}}{\partial x_1} \end{bmatrix} \tag{3.124}$$

$$[B] = \left[[B^{(1)}][B^{(2)}]\dots[B^{(n)}]\right] \tag{3.125}$$

以上より求められた $[B]$, $[D]$ を用いて要素剛性マトリックスは以下のようになる.

$$[K^{(e)}] = \int_{\Omega_e} [B]^T\,[D]\,[B]\,\mathrm{d}\Omega \tag{3.126}$$

3.3.4　外力ベクトル

式 (3.49) 左辺第 2, 3 項についても, 節点変位のベクトルを積分記号の外に出すために以下のようにする.

$$\begin{aligned} \int_{\partial\Omega_e} \delta u_i \cdot t_i\,\mathrm{d}S &= \int_{\partial\Omega_e} \{\delta u_i^{(n)}\}^T\,[N]^T\,\{t\}\,\mathrm{d}S \\ &= \{\delta u_i^{(n)}\}^T \int_{\partial\Omega_e} [N]^T\,\{t\}\,\mathrm{d}S \end{aligned} \tag{3.127}$$

$$\begin{aligned} \int_{\Omega_e} \rho\,\delta u_i \cdot g_i\,\mathrm{d}\Omega &= \int_{\Omega_e} \rho\,\{\delta u_i^{(n)}\}^T\,[N]^T\,\{g\}\,\mathrm{d}\Omega \\ &= \{\delta u_i^{(n)}\}^T \int_{\Omega_e} \rho\,[N]^T\,\{g\}\,\mathrm{d}\Omega \end{aligned} \tag{3.128}$$

ただし

$$[N] = \begin{bmatrix} N^{(1)} & & N^{(2)} & & & N^{(n)} & \\ & N^{(1)} & & N^{(2)} & \cdots & & N^{(n)} \end{bmatrix} \tag{3.129}$$

$$\{t\} = \begin{Bmatrix} t_1 \\ t_2 \end{Bmatrix} \tag{3.130}$$

$$\{g\} = \begin{Bmatrix} g_1 \\ g_2 \end{Bmatrix} \tag{3.131}$$

これらを用い外力ベクトル $\{F^{(e)}\}$ を次のように定義する.

$$\{F^{(e)}\} = \int_{\partial\Omega_e} [N]^T \{t\} \, \mathrm{d}S + \int_{\Omega_e} \rho \, [N]^T \{g\} \, \mathrm{d}\Omega \tag{3.132}$$

以上により求められた要素剛性マトリックスと外力ベクトルより, 3 次元の場合とまったく同様に剛性方程式を作ることができる.

3.3.5 平面問題の理論解 — 単純引っ張り ((i) 平面ひずみ)

平面ひずみを仮定したときの単純引張の理論解を求めよ.

図 3.7 のように x_1-x_2 平面内の平面ひずみであると仮定する. 微小変形を仮定したときは, 荷重が作用している面の面積の変化は考慮しないので物体内に生じる応力は以下のようになる.

$$[T_{ij}] = \begin{bmatrix} \dfrac{P}{L} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.133}$$

一方, ひずみは変位を u_i として, 以下のようになる.

$$[\varepsilon_{ij}] = \begin{bmatrix} \dfrac{2u_1}{L} & 0 & 0 \\ 0 & \dfrac{2u_2}{L} & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.134}$$

ただし, 変位は図 3.8 のように定義する. 式 (3.5) の κ, G を E, ν に書き換えると以下の

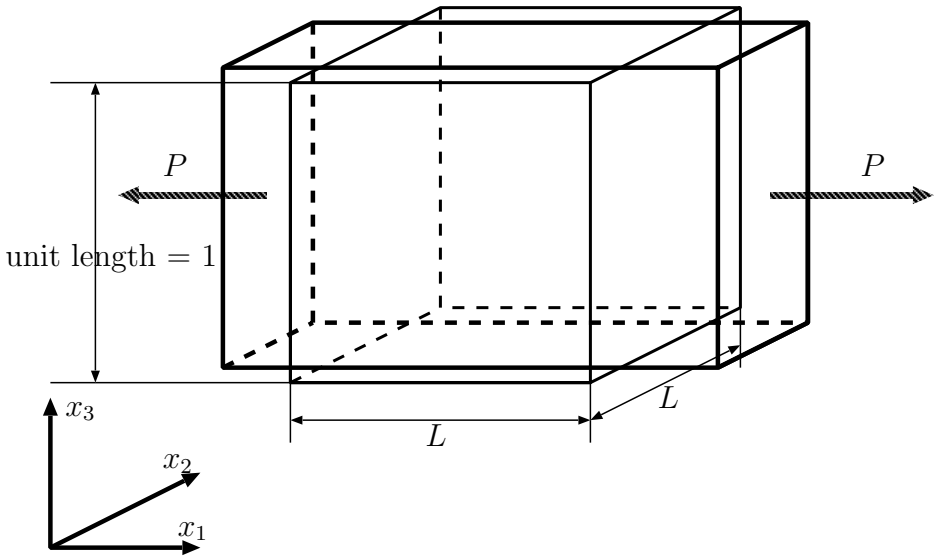


図 3.7: x_1-x_2 平面内の平面ひずみ (単純引張)

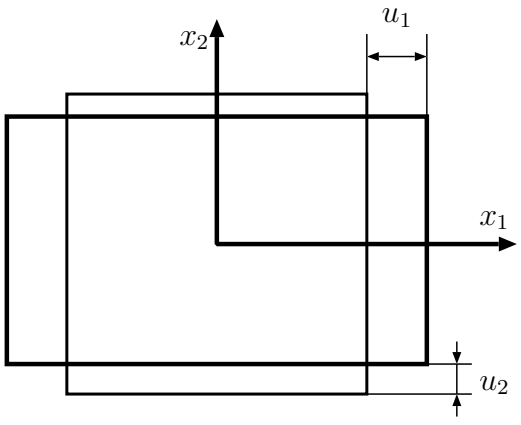


図 3.8: 変位の定義 (単純引張, 平面ひずみ)

ようになる.

$$\begin{aligned} T_{ij} &= \kappa\,(\operatorname{div}\boldsymbol{u})\,\delta_{ij} + 2G\varepsilon_{ij}^{D} \\ &= \frac{E\nu}{(1-2\nu)(1+\nu)}(\operatorname{div}\boldsymbol{u})\,\delta_{ij} + \frac{E}{1+\nu}\varepsilon_{ij} \end{aligned} \tag{3.135}$$

これより応力とひずみの関係式は以下のようになる.

$$\begin{aligned} \begin{bmatrix} \frac{P}{L} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} &= \frac{E\nu}{(1-2\nu)(1+\nu)} \begin{bmatrix} \frac{2u_1+2u_2}{L} & 0 & 0 \\ 0 & \frac{2u_1+2u_2}{L} & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ &\quad + \frac{E}{1+\nu} \begin{bmatrix} \frac{2u_1}{L} & 0 & 0 \\ 0 & \frac{2u_2}{L} & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{3.136}$$

これより下式の関係を得る.

$$\frac{P}{L} = \frac{E\nu}{(1-2\nu)(1+\nu)}\frac{2u_1+u_2}{L} + \frac{E}{1+\nu}\frac{2u_1}{L} \tag{3.137}$$

$$0 = \frac{E\nu}{(1-2\nu)(1+\nu)}\frac{2u_1+2u_2}{L} + \frac{E}{1+\nu}\frac{2u_2}{L} \tag{3.138}$$

最終的に, u_1, u_2 は以下のようになる.

$$u_1 = \frac{P(1-\nu^2)}{2E} \tag{3.139}$$

$$u_2 = -\frac{\nu}{1-\nu}u_1 = -\frac{P\nu(1+\nu)}{2E} \tag{3.140}$$

3.3.6 平面問題の理論解 — 単純引っ張り ((ii) 平面応力)

平面応力を仮定したときの単純引張の理論解を求めよ.

3 次元の単純引張の応力状態は平面応力の特殊な場合と考えることもできるので, 両者の解は同じになる. ここでは確認のため式 (3.115), (3.116), (3.117) をもとに単純引張の解を求める.

図 3.9 に示すような 1 辺の長さ L の立方体の x_1 面に, x_1 面に垂直に P の引張力を作

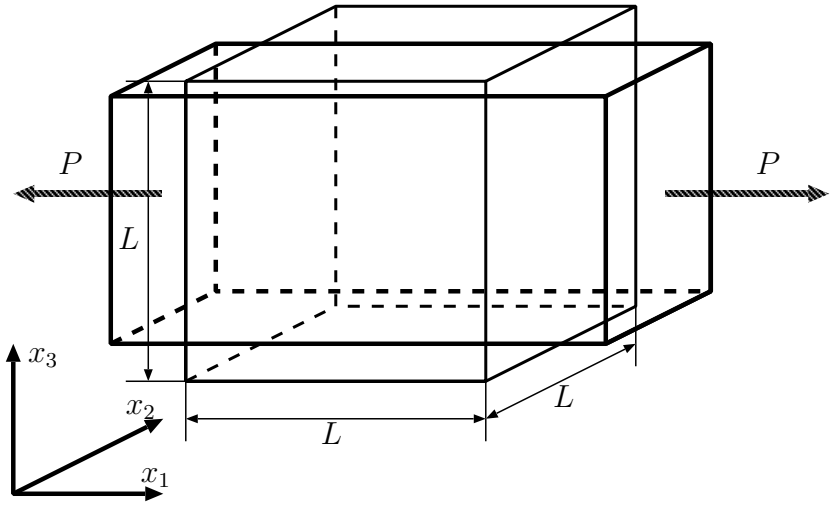


図 3.9: 平面応力 (単純引張)

用させたときの応力は

$$[T_{ij}] = \begin{bmatrix} \dfrac{P}{L^2} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.141}$$

一方, ひずみは変位を u_i として, 以下のようになる.

$$[\varepsilon_{ij}] = \begin{bmatrix} \dfrac{2u_1}{L} & 0 & 0 \\ 0 & \dfrac{2u_2}{L} & 0 \\ 0 & 0 & \dfrac{2u_3}{L} \end{bmatrix} \tag{3.142}$$

ただし, 変位は図 3.10 のように定義する.

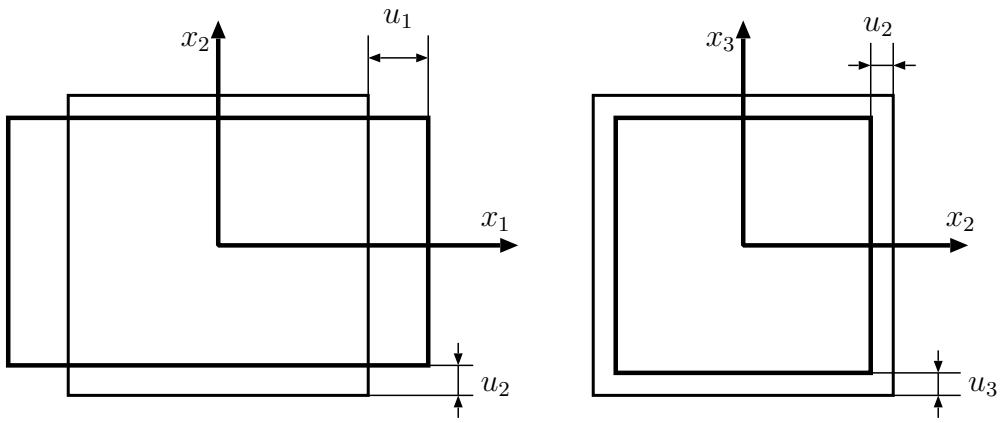


図 3.10: 変位の定義

x_3 方向に平面応力状態にあるとすれば, 応力とひずみの関係は以下のようになる.

$$T_{11} = \frac{2G\left(\frac{2G}{3}+2\kappa\right)}{\frac{4G}{3}+\kappa}\varepsilon_{11} + \frac{2G\left(\kappa-\frac{2G}{3}\right)}{\frac{4G}{3}+\kappa}\varepsilon_{22} \tag{3.143}$$

$$T_{22} = \frac{2G\left(\kappa-\frac{2G}{3}\right)}{\frac{4G}{3}+\kappa}\varepsilon_{11} + \frac{2G\left(\frac{2G}{3}+2\kappa\right)}{\frac{4G}{3}+\kappa}\varepsilon_{22} \tag{3.144}$$

$$T_{12} = 2G\varepsilon_{12} \tag{3.145}$$

$$\varepsilon_{33} = \frac{\frac{2G}{3}-\kappa}{\frac{4G}{3}+\kappa}\left(\varepsilon_{11}+\varepsilon_{22}\right) \tag{3.146}$$

これより

$$\frac{P}{L^2} = \frac{E}{(1+\nu)(1-\nu)}\frac{2u_1}{L} + \frac{E\nu}{(1+\nu)(1-\nu)}\frac{2u_2}{L} \tag{3.147}$$

$$0 = \frac{E\nu}{(1+\nu)(1-\nu)}\frac{2u_1}{L} + \frac{E}{(1+\nu)(1-\nu)}\frac{2u_2}{L} \tag{3.148}$$

$$\frac{2u_3}{L} = -\frac{\nu}{1-\nu}\left(\frac{2u_1}{L}+\frac{2u_2}{L}\right) \tag{3.149}$$

よって

$$u_1 = \frac{P}{2EL} \tag{3.150}$$

$$u_2 = -\nu u_1 = -\frac{P\nu}{2EL} \tag{3.151}$$

$$u_3 = -\nu u_1 = -\frac{P\nu}{2EL} \tag{3.152}$$

3.3.7　平面問題の理論解 — 単純せん断 (i) 平面ひずみ

平面ひずみ, 仮定したときの, 単純剪断の理論解を求めよ.

3 次元の単純剪断は, 平面ひずみ状態にあるので, 両者の解は一致する. ここでは確認のため式 (3.92), (3.93), (3.94) を元に単純剪断の解を求める.

図 3.11 のように x_1-x_2 平面内の平面ひずみ状態にあるとすると,

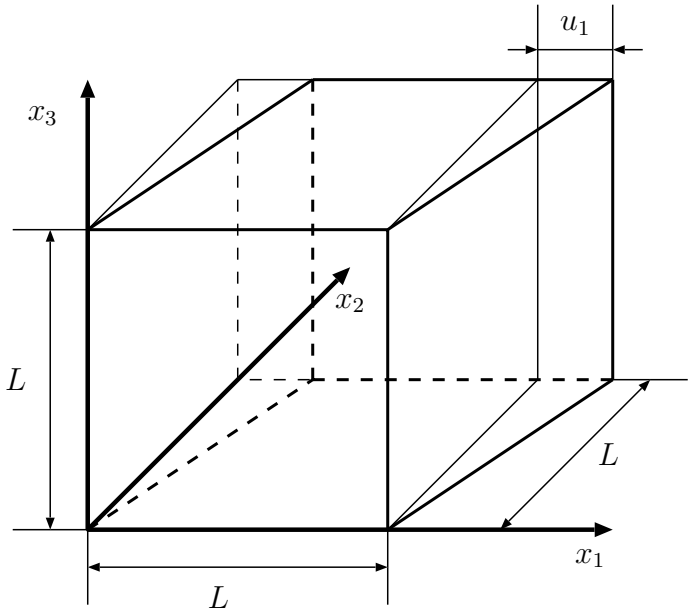


図 3.11: $x_1 - x_2$ 平面内の平面ひずみ (単純剪断)

$$T_{11} = \left(\kappa + \frac{4G}{3}\right)\varepsilon_{11} + \left(\kappa - \frac{2G}{3}\right)\varepsilon_{22} \tag{3.153}$$

$$T_{22} = \left(\kappa - \frac{2G}{3}\right)\varepsilon_{11} + \left(\kappa + \frac{4G}{3}\right)\varepsilon_{22} \tag{3.154}$$

$$T_{12} = 2G\,\varepsilon_{12} \tag{3.155}$$

これより

$$\frac{P}{L^2} = \frac{E}{1+\nu}\,\frac{u_1}{2L} \tag{3.156}$$

よって

$$u_1 = \frac{2\,(1+\nu)\,P}{EL} \tag{3.157}$$

3.3.8 平面問題の理論解 — 単純せん断 (ii) 平面応力

平面応力を仮定したときの, 単純剪断の理論解を求めよ.

3 次元の単純剪断は, 平面応力状態でもあるので, やはり両者の解は一致する. ここでは確認のため (3.115), (3.116), (3.117) をもとに単純剪断の解を求める.

x_3 方向に平面応力状態にあるとすると,

$$T_{11} = \frac{2G\left(\dfrac{2G}{3} + 2\kappa\right)}{\dfrac{4G}{3} + \kappa}\varepsilon_{11} + \frac{2G\left(\kappa - \dfrac{2G}{3}\right)}{\dfrac{4G}{3} + \kappa}\varepsilon_{22} \tag{3.158}$$

$$T_{22} = \frac{2G\left(\kappa - \dfrac{2G}{3}\right)}{\dfrac{4G}{3} + \kappa}\varepsilon_{11} + \frac{2G\left(\dfrac{2G}{3} + 2\kappa\right)}{\dfrac{4G}{3} + \kappa}\varepsilon_{22} \tag{3.159}$$

$$T_{12} = 2G\varepsilon_{12} \tag{3.160}$$

これより, 平面ひずみの場合と同様に,

$$u_1 = \frac{2\,(1+\nu)\,P}{EL} \tag{3.161}$$

第4章　各種ソリッド 要素

この節では ??章で行った有限要素定式化で抽象化したまま取り扱った補間関数に具体的なものを代入して実際にプログラミングを行う.

有限要素法の中身を少しだけ勉強したことがある人には有限要素といえば 3 節点三角形要素を思い浮かべる人も多いだろう. しかしながらこの 3 節点要素はある意味で非常に特殊な要素であり, 最終的なプログラムを簡潔なものにすることができる反面, あまりに簡潔なために 3 節点三角形要素用のプログラムを「拡張」して他の要素に対応させようとすると結局 0 から作り直すのに近い労力を要求される. そのためまず最初に多少複雑になるが, 他への拡張が容易な 8 節点六面体ソリッド 要素を例にとって説明を行う. その後, 各種 2 次元 (四角形, 三角形), 3 次元 (六面体, 四面体) 要素を導入する.

4.1　8 節点六面体ソリッド 要素

4.1.1　補間関数

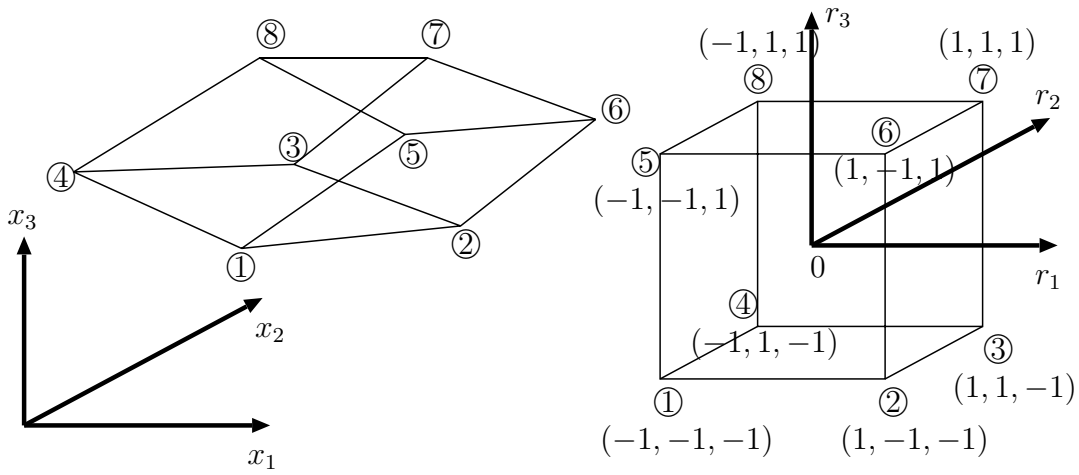
2.1 節では線分内の座標 x を式 (4.1),(4.2) の補間関数を用いて $r(-1 \leq r \leq 1)$ に「座標変換」した.

$$N^{(1)} = \frac{1}{2}(1-r) \tag{4.1}$$

$$N^{(2)} = \frac{1}{2}(1+r) \tag{4.2}$$

8 要素六面体要素も基本的にこれと全く同じで, 3 次元になったので $r(-1 \leq r \leq 1)$ の代わりに r_1, r_2, r_3 $(-1 \leq r_1 \leq 1, -1 \leq r_2 \leq 1, -1 \leq r_3 \leq 1)$ と 3 変数使用し, それにともなって補間関数が掛け合わされる. 即ち物理座標系で図 4.1 のような配置になっている節点を表 4.1 のように対応させる. これは, 物理座標系内の六面体を r_1-r_2-r_3 座標系の立方体

に写像することを意味している. この r_1- r_2-r_3 座標系のことを, 自然座標系と呼ぶ. 補間関数の具体的な形は式 (4.1), (4.2) をそのまま掛け合わせて下式のようになる.



(a) 物理座標系　　(b) 自然座標系

図 4.1: 物理座標系と自然座標系の対応

$$N^{(1)} = \frac{1}{8}(1-r_1)(1-r_2)(1-r_3) \tag{4.3}$$

$$N^{(2)} = \frac{1}{8}(1+r_1)(1-r_2)(1-r_3) \tag{4.4}$$

$$N^{(3)} = \frac{1}{8}(1+r_1)(1+r_2)(1-r_3) \tag{4.5}$$

$$N^{(4)} = \frac{1}{8}(1-r_1)(1+r_2)(1-r_3) \tag{4.6}$$

$$N^{(5)} = \frac{1}{8}(1-r_1)(1-r_2)(1+r_3) \tag{4.7}$$

$$N^{(6)} = \frac{1}{8}(1+r_1)(1-r_2)(1+r_3) \tag{4.8}$$

$$N^{(7)} = \frac{1}{8}(1+r_1)(1+r_2)(1+r_3) \tag{4.9}$$

$$N^{(8)} = \frac{1}{8}(1-r_1)(1+r_2)(1+r_3) \tag{4.10}$$

節点	r_1	r_2	r_3
1	-1	-1	-1
2	1	-1	-1
3	1	1	-1
4	-1	1	-1
5	-1	-1	1
6	1	-1	1
7	1	1	1
8	-1	1	1

表 4.1: 節点の対応

4.1.2 補間関数の微分

ひずみを計算する際に必要な変位 u_i の x_j に関する微分は微分の連鎖則を用いることにより下式のように求められる.

$$\frac{\partial u_i}{\partial x_j} = \frac{\partial N^{(n)}}{\partial x_j} u_i^{(n)} = \left(\frac{\partial N^{(n)}}{\partial r_1}\frac{\partial r_1}{\partial x_j} + \frac{\partial N^{(n)}}{\partial r_2}\frac{\partial r_2}{\partial x_j} + \frac{\partial N^{(n)}}{\partial r_3}\frac{\partial r_3}{\partial x_j} \right) u_i^{(n)} \tag{4.11}$$

ここで, $\dfrac{\partial N^{(n)}}{\partial x_j}$ はやはり微分の連鎖則を用いて, 以下のように求められる. まず下式の

ようなヤコビ（Jacob）マトリックス $[J]$ を求める.

$$\begin{aligned}\begin{bmatrix}\dfrac{\partial N^{(n)}}{\partial r_1}\\ \dfrac{\partial N^{(n)}}{\partial r_2}\\ \dfrac{\partial N^{(n)}}{\partial r_3}\end{bmatrix} &= \begin{bmatrix}\dfrac{\partial x_1}{\partial r_1}\dfrac{\partial x_2}{\partial r_1}\dfrac{\partial x_3}{\partial r_1}\\ \dfrac{\partial x_1}{\partial r_2}\dfrac{\partial x_2}{\partial r_2}\dfrac{\partial x_3}{\partial r_2}\\ \dfrac{\partial x_1}{\partial r_3}\dfrac{\partial x_2}{\partial r_3}\dfrac{\partial x_3}{\partial r_3}\end{bmatrix}\begin{bmatrix}\dfrac{\partial N^{(n)}}{\partial x_1}\\ \dfrac{\partial N^{(n)}}{\partial x_2}\\ \dfrac{\partial N^{(n)}}{\partial x_3}\end{bmatrix}\\ &= [J]\begin{bmatrix}\dfrac{\partial N^{(n)}}{\partial x_1}\\ \dfrac{\partial N^{(n)}}{\partial x_2}\\ \dfrac{\partial N^{(n)}}{\partial x_3}\end{bmatrix}\end{aligned} \tag{4.12}$$

このヤコビマトリックスの各成分 $\dfrac{\partial x_i}{\partial r_j}$ は以下のようになる.

$$\frac{\partial x_i}{\partial r_j} = \frac{\partial N^{(n)}}{\partial r_j} x_i^{(n)} \tag{4.13}$$

このヤコビマトリックスを用いれば, $\dfrac{\partial N^{(n)}}{\partial x_i}$ は下式のように求められる.

$$\begin{bmatrix}\dfrac{\partial N^{(n)}}{\partial x_1}\\ \dfrac{\partial N^{(n)}}{\partial x_2}\\ \dfrac{\partial N^{(n)}}{\partial x_3}\end{bmatrix} = [J]^{-1}\begin{bmatrix}\dfrac{\partial N^{(n)}}{\partial r_1}\\ \dfrac{\partial N^{(n)}}{\partial r_2}\\ \dfrac{\partial N^{(n)}}{\partial r_3}\end{bmatrix} \tag{4.14}$$

なおここで, 補間関数の微分は以下のようになる.

$$\frac{\partial N^{(1)}}{\partial r_1} = -\frac{1}{8}(1-r_2)(1-r_3) \quad (4.15) \qquad \frac{\partial N^{(5)}}{\partial r_1} = -\frac{1}{8}(1-r_2)(1+r_3) \quad (4.27)$$

$$\frac{\partial N^{(1)}}{\partial r_2} = -\frac{1}{8}(1-r_1)(1-r_3) \quad (4.16) \qquad \frac{\partial N^{(5)}}{\partial r_2} = -\frac{1}{8}(1-r_1)(1+r_3) \quad (4.28)$$

$$\frac{\partial N^{(1)}}{\partial r_3} = -\frac{1}{8}(1-r_1)(1-r_2) \quad (4.17) \qquad \frac{\partial N^{(5)}}{\partial r_3} = \frac{1}{8}(1-r_1)(1-r_2) \quad (4.29)$$

$$\frac{\partial N^{(2)}}{\partial r_1} = \frac{1}{8}(1-r_2)(1-r_3) \quad (4.18) \qquad \frac{\partial N^{(6)}}{\partial r_1} = \frac{1}{8}(1-r_2)(1+r_3) \quad (4.30)$$

$$\frac{\partial N^{(2)}}{\partial r_2} = -\frac{1}{8}(1+r_1)(1-r_3) \quad (4.19) \qquad \frac{\partial N^{(6)}}{\partial r_2} = -\frac{1}{8}(1+r_1)(1+r_3) \quad (4.31)$$

$$\frac{\partial N^{(2)}}{\partial r_3} = -\frac{1}{8}(1+r_1)(1-r_2) \quad (4.20) \qquad \frac{\partial N^{(6)}}{\partial r_3} = \frac{1}{8}(1+r_1)(1-r_2) \quad (4.32)$$

$$\frac{\partial N^{(3)}}{\partial r_1} = \frac{1}{8}(1+r_2)(1-r_3) \quad (4.21) \qquad \frac{\partial N^{(7)}}{\partial r_1} = \frac{1}{8}(1+r_2)(1+r_3) \quad (4.33)$$

$$\frac{\partial N^{(3)}}{\partial r_2} = \frac{1}{8}(1+r_1)(1-r_3) \quad (4.22) \qquad \frac{\partial N^{(7)}}{\partial r_2} = \frac{1}{8}(1+r_1)(1+r_3) \quad (4.34)$$

$$\frac{\partial N^{(3)}}{\partial r_3} = -\frac{1}{8}(1+r_1)(1+r_2) \quad (4.23) \qquad \frac{\partial N^{(7)}}{\partial r_3} = \frac{1}{8}(1+r_1)(1+r_2) \quad (4.35)$$

$$\frac{\partial N^{(4)}}{\partial r_1} = -\frac{1}{8}(1+r_2)(1-r_3) \quad (4.24) \qquad \frac{\partial N^{(8)}}{\partial r_1} = -\frac{1}{8}(1+r_2)(1+r_3) \quad (4.36)$$

$$\frac{\partial N^{(4)}}{\partial r_2} = \frac{1}{8}(1-r_1)(1-r_3) \quad (4.25) \qquad \frac{\partial N^{(8)}}{\partial r_2} = \frac{1}{8}(1-r_1)(1+r_3) \quad (4.37)$$

$$\frac{\partial N^{(4)}}{\partial r_3} = -\frac{1}{8}(1-r_1)(1+r_2) \quad (4.26) \qquad \frac{\partial N^{(8)}}{\partial r_3} = \frac{1}{8}(1-r_1)(1+r_2) \quad (4.38)$$

4.1.3 数値積分

このヤコビマトリックスを用いると, 領域積分は以下のように表される.

$$\int_{\Omega_e} d\Omega = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \det[J] dr_1\, dr_2\, dr_3 \quad (4.39)$$

この積分は通常, Gauss 積分などの数値積分法により実行される. 前述のように 1 次元の関数 $f(x)$ の Gauss 積分はサンプリング点 x_i, 重み w_i を用いて下式のように表される.

$$\int_{-1}^{1} f(x)\, dx \approx \sum_i w_i f(x_i) \quad (4.40)$$

これを 3 次元関数に拡張する方法はいくつか提案されているが, ここでは最も単純に総和を 3 重にしたものを用いることにする. 即ち,

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f(x,y,z)\,dx\,dy\,dz \approx \sum_i \sum_j \sum_k w_i w_j w_k f(x_i, y_j, z_k) \tag{4.41}$$

4.1.4　課題

単純引っ張り, 単純せん断の近似解を有限要素法により求めよ.

4.1.5　バージョン 8 — 補間関数と数値積分

バージョン 7 を 3 次元に拡張する.

拡張に当たって, 節点座標を 3 次元に対応させる必要が生じる. そこで以後の拡張にも対応できるように, プログラム中の変数を以下のように変更する.

- 節点数の最大値 `MXNODE` (1000)
- 要素数の最大値 `MXELEM` (1000)
- 1 節点あたりの自由度数の最大値 `MXDOFN` (3)
- 1 要素あたりの節点数の最大値 `MXNOEL` (8)
- 総節点数 `npoin`
- 節点座標 `coords`(`MXDOFN`,`MXNODE`) [1]
- 総要素数 `nelem`

[1] `coords` や `lnods` のような 2 次元配列の添字が, 例えば `coords(3,1000)` のようになっているのは, 同じことを C で書くと `coords[1000][3]` あるいは `*coords[3]` となるからである. 理屈からいったら `coords(1000,3)`（C なら`coords[3][1000]` あるいは `*coords[1000]`）でも良いのだが, `coords` に格納されているのが座標値であることを考えれば結びつきの強い（= 同時にアクセスされる）ものが近接したメモリー上に格納されている方がより自然である. また要素剛性を作るときなどは`lnods` のうち 1 要素分しか必要ない. そのときは main から要素剛性を call するときに `lnods(1,i)` のアドレスを渡してサブルーチン内では 1 次元配列として取り扱えばコーディングが簡潔化できる.

- 各要素の節点数 ntnoel(MXELEM)
- コネクティビティ lnods(MXNOEL,MXELEM) [1]
- 1節点あたりの自由度数 ndofn
- 総自由度数 ntotdf=npoin×ndofn
- 各要素の節点数 ntnoel (MXELEM)

次にこれに対応した入力ファイルをつくる. まず1辺1の立方体で1要素しかないメッシュをつくる.

ファイル	プログラム中の変数
3 2	ndofn nint
8	npoin
1 0.0 0.0 0.0	coords(1,1) coords(2,1) coords(3,1)
2 1.0 0.0 0.0	coords(1,2) coords(2,2) coords(3,2)
3 0.0 1.0 0.0	coords(1,3) coords(2,3) coords(3,3)
4 1.0 1.0 0.0	coords(1,4) coords(2,4) coords(3,4)
5 0.0 0.0 1.0	coords(1,5) coords(2,5) coords(3,5)
6 1.0 0.0 1.0	coords(1,6) coords(2,6) coords(3,6)
7 0.0 1.0 1.0	coords(1,7) coords(2,7) coords(3,7)
8 1.0 1.0 1.0	coords(1,8) coords(2,8) coords(3,8)
1	nelem
1 8 1 2 4 3 5 6 8 7	ntnoel(1) lnods(1,1) lnods(2,1) ... lnods(8,1)

表 4.2: データファイルの例

要素剛性のプログラミングで発生するバグは分類すれば (i)D, B マトリックス, (ii) ヤコビマトリックス, (iii) 数値積分 である. このうち (ii) 及び (iii) が正しくできているのを検証するのによく用いられている方法は要素の物理空間における体積を求めて実際のもの

と比較することである. 要素の物理空間における体積は式 (4.39) の積分をそのまま実行すれば求められる. バージョン 8 では与えられたメッシュに対し, 要素の体積を求めるプログラムを作成する.

3 重の数値積分のループは, 例えば以下のようにコーディングすれば良い. [2]

$\mathtt{vol} = 0$

for $i1 = 1 \sim \mathtt{nint}$

for $i2 = 1 \sim \mathtt{nint}$

for $i3 = 1 \sim \mathtt{nint}$

$r1 = (\text{sampring point } i1)$

$r2 = (\text{sampring point } i2)$

$r3 = (\text{sampring point } i3)$

$(\partial N/\partial r(r_1, r_2, r_3))$

$(J, \det J)$

$w_1 = (\text{weight } i1)$

$w_2 = (\text{weight } i2)$

$w_3 = (\text{weight } i3)$

$\mathtt{vol} = \mathtt{vol} + \det J \cdot w1 \cdot w2 \cdot w3$

end for

end for

end for

4.1.6　fortran コーディング例

入力データ例

[2] $\dfrac{\partial N}{\partial r}$ のコーディングは, あまり細工せず, 全成分を直接書き下すとある程度系統的に成分が並ぶので間違いをおかしても探しやすい. これは今後取り扱う要素についても言える.

```
3 2
8
 1 0.000000D0   0.000000D0   0.000000D0
 2 1.000000D0   0.000000D0   0.000000D0
 3 0.000000D0   1.000000D0   0.000000D0
 4 1.000000D0   1.000000D0   0.000000D0
 5 0.000000D0   0.000000D0   2.000000D0
 6 1.000000D0   0.000000D0   2.000000D0
 7 0.000000D0   1.000000D0   2.000000D0
 8 1.000000D0   1.000000D0   2.000000D0
1
 1  8  1  2  4  3  5  6  8  7
```

fortran コーディング例

```
      implicit real*8(a-h,o-z)
      parameter (MXNODE = 100)
      parameter (MXELEM = 100)
      parameter (MXDOFN = 3)
      parameter (MXNOEL = 8)
      dimension coords(MXDOFN,MXNODE),lnods(MXNOEL,MXELEM)
      dimension ntnoel(MXELEM)
c
      open(10,file='temp.dat')
c
      read(10,*) ndofn,nint
      read(10,*) npoin
      do i = 1, npoin
          read(10,*) itemp, (coords(j,i),j=1,ndofn)
      end do
c
      read(10,*) nelem
      do i = 1,nelem
          read(10,*) itemp, ntnoel(i),(lnods(j,i),j=1,ntnoel(i))
      end do
c
      do ielem = 1,nelem
          call element(coords,MXDOFN,lnods(1,ielem),nint,ntnoel(ielem))
      end do
c
 1000 format(2x,e13.6,2x,e13.6)
c
      stop
      end
```

```
c
c######################################################################
      subroutine element(coords,MXDOFN,lnods,nint,ntnoel)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(MXDOFN,*),lnods(*)
      dimension gsp(4,4),wgh(4,4),dndx(3,8)
c
      data gsp /  0.D0          , 0.D0          , 0.D0,          0.D0,
     1           -0.577350269189626D0, 0.577350269189626D0, 0.D0,0.D0,
     1           -0.774596669241483D0, 0.D0, 0.774596669241483D0,0.D0,
     1           -0.861136311594053D0,-0.339981043584856D0,
     1            0.339981043584856D0, 0.861136311594053D0 /
      data wgh /  2.D0          , 0.D0          , 0.D0,          0.D0,
     1            1.D0          , 1.D0          , 0.D0,          0.D0,
     1            0.555555555555556D0, 0.888888888888889D0,
     1            0.555555555555556D0, 0.D0,
     1            0.347854845137454D0, 0.652145154862546D0,
     1            0.652145154862546D0, 0.347854845137454D0 /
c
      vol = 0.D0
      do ir1 = 1,nint
      do ir2 = 1,nint
      do ir3 = 1,nint
c
          r1 = gsp(ir1,nint)
          r2 = gsp(ir2,nint)
          r3 = gsp(ir3,nint)
c
          call shap3d(r1,r2,r3,coords,MXDOFN,lnods,detjac,dndx)
          detwei= detjac*wgh(ir1,nint)*wgh(ir2,nint)*wgh(ir3,nint)
          vol = vol + detwei
c
      end do
      end do
      end do
c
      write(*,*) vol
c
      return
      end
c
C######################################################################
      subroutine shap3d(r1,r2,r3,coords,MXDOFN,lnods,detjac,dndx)
C######################################################################
```

```
      implicit real*8(a-h,o-z)
      dimension coords(MXDOFN,*),lnods(*)
      dimension ajacob(3,3),dndr(3,8)
      dimension dndx(3,*),ajainv(3,3)
c
      dndr(1,1) = -0.125D0 * (1.D0 - r2) * (1.D0 - r3)
      dndr(2,1) = -0.125D0 * (1.D0 - r1) * (1.D0 - r3)
      dndr(3,1) = -0.125D0 * (1.D0 - r1) * (1.D0 - r2)
c
      dndr(1,2) =  0.125D0 * (1.D0 - r2) * (1.D0 - r3)
      dndr(2,2) = -0.125D0 * (1.D0 + r1) * (1.D0 - r3)
      dndr(3,2) = -0.125D0 * (1.D0 + r1) * (1.D0 - r2)
c
      dndr(1,3) =  0.125D0 * (1.D0 + r2) * (1.D0 - r3)
      dndr(2,3) =  0.125D0 * (1.D0 + r1) * (1.D0 - r3)
      dndr(3,3) = -0.125D0 * (1.D0 + r1) * (1.D0 + r2)
c
      dndr(1,4) = -0.125D0 * (1.D0 + r2) * (1.D0 - r3)
      dndr(2,4) =  0.125D0 * (1.D0 - r1) * (1.D0 - r3)
      dndr(3,4) = -0.125D0 * (1.D0 - r1) * (1.D0 + r2)
c
      dndr(1,5) = -0.125D0 * (1.D0 - r2) * (1.D0 + r3)
      dndr(2,5) = -0.125D0 * (1.D0 - r1) * (1.D0 + r3)
      dndr(3,5) =  0.125D0 * (1.D0 - r1) * (1.D0 - r2)
c
      dndr(1,6) =  0.125D0 * (1.D0 - r2) * (1.D0 + r3)
      dndr(2,6) = -0.125D0 * (1.D0 + r1) * (1.D0 + r3)
      dndr(3,6) =  0.125D0 * (1.D0 + r1) * (1.D0 - r2)
c
      dndr(1,7) =  0.125D0 * (1.D0 + r2) * (1.D0 + r3)
      dndr(2,7) =  0.125D0 * (1.D0 + r1) * (1.D0 + r3)
      dndr(3,7) =  0.125D0 * (1.D0 + r1) * (1.D0 + r2)
c
      dndr(1,8) = -0.125D0 * (1.D0 + r2) * (1.D0 + r3)
      dndr(2,8) =  0.125D0 * (1.D0 - r1) * (1.D0 + r3)
      dndr(3,8) =  0.125D0 * (1.D0 - r1) * (1.D0 + r2)
c
      call jacb3d(dndr,coords,MXDOFN,lnods,ajacob,detjac,ajainv)
c
      return
      end
c
C######################################################################
      subroutine jacb3d(dndr,coords,MXDOFN,lnods,ajacob,detjac,ajainv)
C######################################################################
```

```
      implicit real*8 (a-h,o-z)
      dimension ajacob(3,*),ajainv(3,*)
      dimension dndr(3,*)
      dimension coords(MXDOFN,*),lnods(*)
C
      do i=1,3
          do j=1,3
              ajacob(i,j) = 0.D0
          end do
      end do
C
      do ki=1,8
          do i = 1,3
              do j = 1,3
                  ajacob(j,i)=ajacob(j,i)+dndr(j,ki)*coords(i,lnods(ki))
              end do
          end do
      end do
C
      detjac = ajacob(1,1)*ajacob(2,2)*ajacob(3,3)
     1       + ajacob(2,1)*ajacob(3,2)*ajacob(1,3)
     1       + ajacob(3,1)*ajacob(1,2)*ajacob(2,3)
     1       - ajacob(1,1)*ajacob(3,2)*ajacob(2,3)
     1       - ajacob(3,1)*ajacob(2,2)*ajacob(1,3)
     1       - ajacob(2,1)*ajacob(1,2)*ajacob(3,3)
C
      return
      end
c
```

4.1.7　バージョン 9 — merge の拡張

できた要素剛性を全体剛性に加える merge の操作で発生するバグはインデックスの間違いといえる. merge の debug は, 要素剛性に 1 とか 2 などわかりやすい値のダミーデータを入れて, できあがった全体剛性を確認することで行う.

1 節点数あたりの自由度数を可変にするためには merge は以下のようにコーディング

すれば良い.

for $i = 1 \sim \texttt{nelem}$
  call element(i)
  for $j = 1 \sim \texttt{ntnoel}(i)$
    for $k = 1 \sim \texttt{ndofn}$
      $\mathrm{ip}((j-1)\cdot\texttt{ndofn}+k) = (\texttt{lnods}(j,i)-1)\cdot\texttt{ndofn}+k$
    end for
  end for
  for $j = 1 \sim \texttt{ntnoel}(i)\cdot\texttt{ndofn}$
    for $k = 1 \sim \texttt{ntnoel}(i)\cdot\texttt{ndofn}$
      $K_{ip(j),ip(k)} = K_{ip(j),ip(k)} + A_{jk}$
    end for
  end for
end for

4.1.8 バージョン 10 — 単純引っ張り

実際に要素剛性をつくり, 全体剛性に merge して以下に示す境界条件処理をして, Gauss の消去法により解を求める. まず $1/8$ 領域を 1 要素で解析し x_1, x_2, x_3 方向それぞれに荷重を作用させたときにすべて正しく解が求められることが確かめられたら, 次に $1/8$ 領域を 8 要素に分割したものについても行う. また材料定数をいくつかのものについて確認する.

4.1.9 外力ベクトル

簡単のため図 4.2 のように $x_1 - x_2$ 平面に平行な面⑤-⑥-⑦-⑧ 上に荷重が作用しており

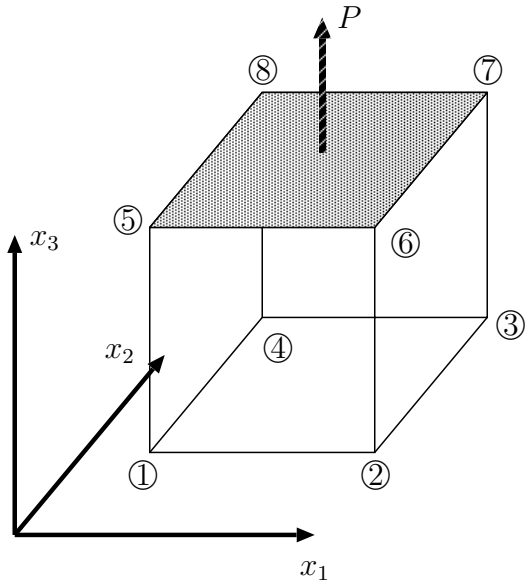


図 4.2: 単純引っ張り

⑤-⑥//⑦-⑧//x_1 軸, ⑥-⑦//⑧-⑤//x_2 軸,

$$x_1^{(6)} = x_1^{(5)} + a \tag{4.42}$$
$$x_1^{(7)} = x_1^{(5)} + a \tag{4.43}$$
$$x_1^{(8)} = x_1^{(5)} \tag{4.44}$$
$$x_2^{(6)} = x_2^{(5)} \tag{4.45}$$
$$x_2^{(7)} = x_2^{(5)} + b \tag{4.46}$$
$$x_2^{(8)} = x_2^{(5)} + b \tag{4.47}$$

であるとする. ⑤-⑥-⑦-⑧面上では $r_3 = 1$ なので $\dfrac{\partial N}{\partial r}$ は以下のようになる.

$$\frac{\partial N^{(5)}}{\partial r_1} = -\frac{1}{4}(1 - r_2) \tag{4.48}$$
$$\frac{\partial N^{(5)}}{\partial r_2} = -\frac{1}{4}(1 - r_1) \tag{4.49}$$
$$\frac{\partial N^{(6)}}{\partial r_1} = \frac{1}{4}(1 - r_2) \tag{4.50}$$
$$\frac{\partial N^{(6)}}{\partial r_2} = -\frac{1}{4}(1 + r_1) \tag{4.51}$$
$$\frac{\partial N^{(7)}}{\partial r_1} = \frac{1}{4}(1 + r_2) \tag{4.52}$$
$$\frac{\partial N^{(7)}}{\partial r_2} = \frac{1}{4}(1 + r_1) \tag{4.53}$$
$$\frac{\partial N^{(8)}}{\partial r_1} = -\frac{1}{4}(1 + r_2) \tag{4.54}$$
$$\frac{\partial N^{(8)}}{\partial r_2} = \frac{1}{4}(1 - r_1) \tag{4.55}$$

これを用いると $\dfrac{\partial x}{\partial r}$ は以下のように求められる.

$$\begin{aligned}\frac{\partial x_1}{\partial r_1} &= -\frac{1}{4}(1 - r_2)x_1^{(5)} + \frac{1}{4}(1 - r_2)x_1^{(6)} + \frac{1}{4}(1 + r_2)x_1^{(7)} - \frac{1}{4}(1 + r_2)x_1^{(8)} \\ &= -\frac{1}{4}(1 - r_2)x_1^{(5)} + \frac{1}{4}(1 - r_2)(x_1^{(5)} + a) + \frac{1}{4}(1 + r_2)(x_1^{(5)} + a) - \frac{1}{4}(1 + r_2)x_1^{(5)} \\ &= \frac{a}{2}\end{aligned} \tag{4.56}$$

$$\begin{aligned}\frac{\partial x_2}{\partial r_1} &= -\frac{1}{4}(1-r_2)x_2^{(5)} + \frac{1}{4}(1-r_2)x_2^{(6)} + \frac{1}{4}(1+r_2)x_2^{(7)} - \frac{1}{4}(1+r_2)x_2^{(8)} \\ &= -\frac{1}{4}(1-r_2)x_2^{(5)} + \frac{1}{4}(1-r_2)x_2^{(5)} + \frac{1}{4}(1+r_2)(x_2^{(5)}+b) - \frac{1}{4}(1+r_2)(x_2^{(5)}+b) \\ &= 0 \end{aligned} \tag{4.57}$$

$$\begin{aligned}\frac{\partial x_1}{\partial r_2} &= -\frac{1}{4}(1-r_1)x_1^{(5)} - \frac{1}{4}(1+r_1)x_1^{(6)} + \frac{1}{4}(1+r_1)x_1^{(7)} + \frac{1}{4}(1-r_1)x_1^{(8)} \\ &= -\frac{1}{4}(1-r_1)x_1^{(5)} - \frac{1}{4}(1+r_1)(x_1^{(5)}+a) + \frac{1}{4}(1+r_1)(x_1^{(5)}+a) + \frac{1}{4}(1-r_1)x_1^{(5)} \\ &= 0 \end{aligned} \tag{4.58}$$

$$\begin{aligned}\frac{\partial x_2}{\partial r_2} &= -\frac{1}{4}(1-r_1)x_2^{(5)} - \frac{1}{4}(1+r_1)x_2^{(6)} + \frac{1}{4}(1+r_1)x_2^{(7)} + \frac{1}{4}(1-r_1)x_2^{(8)} \\ &= -\frac{1}{4}(1-r_1)x_2^{(5)} - \frac{1}{4}(1+r_1)x_2^{(5)} + \frac{1}{4}(1+r_1)(x_2^{(5)}+b) + \frac{1}{4}(1-r_1)(x_2^{(5)}+b) \\ &= \frac{b}{2} \end{aligned} \tag{4.59}$$

以上から, 単位面積あたりの荷重を P とすれば, 各節点の外力は以下のようになる.

$$\{F^{(e)}\} = \int_{\partial\Omega_e} [N] \begin{Bmatrix} 0 \\ 0 \\ P \end{Bmatrix} dS \tag{4.60}$$

$$= \int_{-1}^{1}\int_{-1}^{1} \begin{bmatrix} N^{(1)} & & \cdots & N^{(8)} & & \\ & N^{(1)} & & \cdots & & N^{(8)} \\ & & N^{(1)} & & \cdots & & N^{(8)} \end{bmatrix}^T \begin{Bmatrix} 0 \\ 0 \\ P \end{Bmatrix} \frac{ab}{4}\, dr_1\, dr_2 \tag{4.61}$$

$$F_1^{(i)} = F_2^{(i)} = 0 \quad (i = 1 \sim 8) \tag{4.62}$$

$$F_3^{(i)} = 0 \quad (i = 1 \sim 4) \quad (\because r_3 = 1) \tag{4.63}$$

$$F_3^{(5)} = P\int_{-1}^{1}\int_{-1}^{1} N^{(5)}\frac{ab}{4}\,dr_1\,dr_2 \tag{4.64}$$
$$= P\int_{-1}^{1}\int_{-1}^{1} \frac{1}{4}(1-r_1)(1-r_2)\frac{ab}{4}\,dr_1\,dr_2 \tag{4.65}$$
$$= \frac{abP}{4} \tag{4.66}$$
$$F_3^{(6)} = P\int_{-1}^{1}\int_{-1}^{1} N^{(6)}\frac{ab}{4}\,dr_1\,dr_2 \tag{4.67}$$
$$= P\int_{-1}^{1}\int_{-1}^{1} \frac{1}{4}(1+r_1)(1-r_2)\frac{ab}{4}\,dr_1\,dr_2 \tag{4.68}$$
$$= \frac{abP}{4} \tag{4.69}$$
$$F_3^{(7)} = P\int_{-1}^{1}\int_{-1}^{1} N^{(7)}\frac{ab}{4}\,dr_1\,dr_2 \tag{4.70}$$
$$= P\int_{-1}^{1}\int_{-1}^{1} \frac{1}{4}(1+r_1)(1+r_2)\frac{ab}{4}\,dr_1\,dr_2 \tag{4.71}$$
$$= \frac{abP}{4} \tag{4.72}$$
$$F_3^{(8)} = P\int_{-1}^{1}\int_{-1}^{1} N^{(8)}\frac{ab}{4}\,dr_1\,dr_2 \tag{4.73}$$
$$= P\int_{-1}^{1}\int_{-1}^{1} \frac{1}{4}(1-r_1)(1+r_2)\frac{ab}{4}\,dr_1\,dr_2 \tag{4.74}$$
$$= \frac{abP}{4} \tag{4.75}$$

以上をまとめると, 8 節点六面体ソリッド要素のある面に等分布荷重が作用している場合, その面に含まれる 4 つの節点に 1/4 づつ荷重を振り分ければ良いことがわかる.

4.1.10　境界条件

有限要素法で静的な構造解析を行う際には解くべき方程式がつりあい方程式であるため, 解析を行うものが, 適切に不動点を設定する必要がある. 例えば今回行うような単純引っ張りであれば, 図 4.3 のように直方体の相対する 2 面を引っ張るだけで, 中心がどこにあるかは式の上からは何も決まらない.

このときの変形を考えてみると変形前後で重心が動かないと解析者が仮定すれば図 4.4

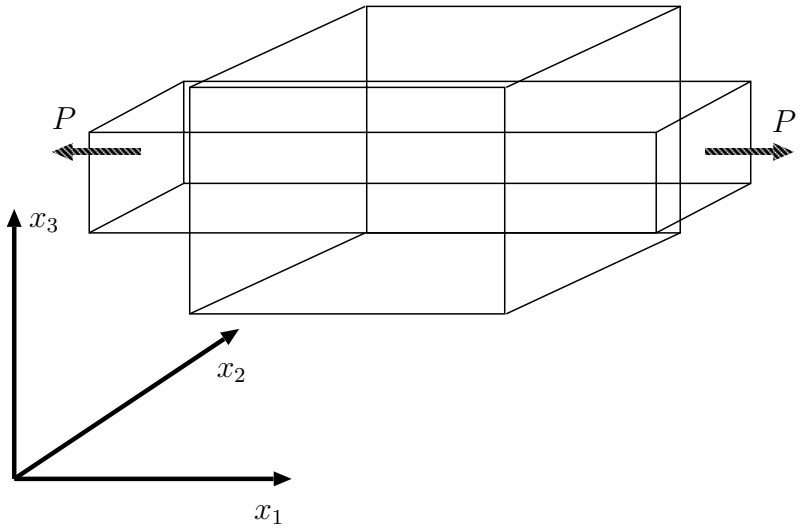
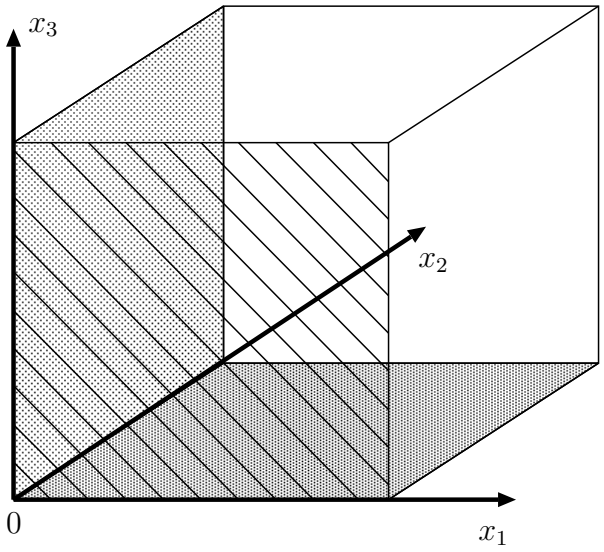


図 4.3: 単純引っ張りの境界条件

の変形には3つの対称面がある. 直方体の重心の座標を (0,0,0) と仮定し, 3つの対称面で分割される 1/8 領域の1つとして第1象限を取り出してみると, 以下のような境界条件になっている.



$x_1 - x_2$ 平面上の点: x_3 方向固定

$x_2 - x_3$ 平面上の点: x_1 方向固定

$x_3 - x_1$ 平面上の点: x_2 方向固定

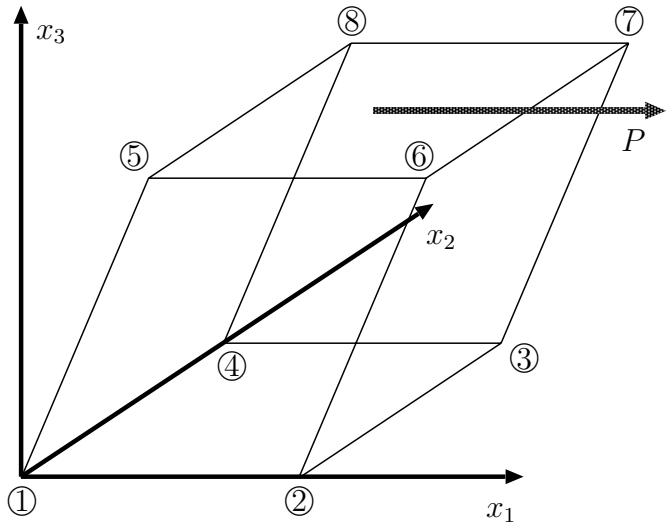
図 4.4: 1/8 対象モデル

4.1.11　バージョン 11 — 単純せん断

前述のように, 単純せん断は線形/非線形を問わず理論解が求められる数少ない例題の 1 つであるが, 実は有限要素法で単純せん断の変形モードを実現するのは難しい. 今回の 8 節点六面体ソリッド要素は可能だが, 後述するような 20 節点や 27 節点の六面体ソリッド要素では合理的に単純せん断の変形モードを実現できない.

debug の観点からは単純引っ張りと単純せん断が正しく解析できれば, D マトリックスはほぼ bug はないといえる. 要素の種類が変わっても, 用いる構成式が同じなら D マトリックスは同じであるため, 通常は新しい D マトリックスは 8 節点六面体ソリッド要素で単純引っ張りと単純せん断の解析を行うことにより debug し, その他の要素は 8 節点六面体ソリッド要素のプログラムを流用あるいは拡張して作成し単純引っ張りだけで debug する.

単純せん断の解析は 1 要素で, 以下のような境界条件で行う.



①,②,③,④　$x_1x_2x_3$ 方向固定.
⑤,⑥,⑦,⑧ x_2, x_3 方向固定.
⑤,⑥,⑦,⑧ の x_1 方向にそれぞれ $P/4$ の荷重を作用させる.

これを, ヤング率, ポアソン比をかえ, また x_2, x_3 方向についても正しい解が得られることを確認する.

図 4.5: 単純せん断

入力データ例

```
3 2
8
 1 0.000000D0   0.000000D0   0.000000D0
 2 1.000000D0   0.000000D0   0.000000D0
 3 0.000000D0   1.000000D0   0.000000D0
 4 1.000000D0   1.000000D0   0.000000D0
```

```
 5 0.000000D0   0.000000D0   1.000000D0
 6 1.000000D0   0.000000D0   1.000000D0
 7 0.000000D0   1.000000D0   1.000000D0
 8 1.000000D0   1.000000D0   1.000000D0
1
 1  8  1  2  4  3  5  6  8  7
12
 1  1  1  0.D0
 2  1  2  0.D0
 3  1  3  0.D0
 4  2  2  0.D0
 5  2  3  0.D0
 6  3  1  0.D0
 7  3  3  0.D0
 8  4  3  0.D0
 9  5  1  0.D0
10  5  2  0.D0
11  6  2  0.D0
12  7  1  0.D0
```

fortran コーディング例

```
      implicit real*8(a-h,o-z)
      parameter (MXNODE = 100)
      parameter (MXELEM = 100)
      parameter (MXDOFN = 3)
      parameter (MXNOEL = 8)
      parameter (MXFIX1 = 100)
      dimension coords(MXDOFN,MXNODE),lnods(MXNOEL,MXELEM)
      dimension ntnoel(MXELEM)
      dimension astiff(MXDOFN*MXNOEL,MXDOFN*MXNOEL)
      dimension a(MXNODE*MXDOFN * MXNODE*MXDOFN),b(MXNODE*MXDOFN)
      dimension idispl(MXFIX1),bvalue(MXFIX1)
c
      open(10,file='temp.dat')
c
      read(10,*) ndofn,nint
      read(10,*) npoin
      do i = 1, npoin
          read(10,*) itemp, (coords(j,i),j=1,ndofn)
      end do
c
      read(10,*) nelem
      do i = 1,nelem
          read(10,*) itemp, ntnoel(i),(lnods(j,i),j=1,ntnoel(i))
```

```
      end do
c
      read(10,*) ntdisp
      write(*,*) 'ntdisp',ntdisp
      do i = 1,ntdisp
          read(10,*) itemp1, i1, i2, bvalue(i)
          idispl(i) = (i1-1)*ndofn+i2
          write(*,*) i,itemp1,idispl(i), i1, i2, bvalue(i)
      end do
c
      yo = 1.D0
      po = 0.3D0
c
      ntotdf = npoin * ndofn
c
      do i = 1,ntotdf*ntotdf
          a(i) = 0.D0
      end do
c
      do ielem = 1,nelem
          call element(coords,MXDOFN,lnods(1,ielem),nint,ntnoel(ielem),
     1                 ndofn,astiff,yo,po)
          call merge(a,lnods(1,ielem),astiff,ntnoel(ielem),ntotdf,ndofn)
      end do
c
      do i = 1,24
          write(*,*) a(i+23*ntotdf)
      end do
      do i = 1,ntotdf
          b(i) = 0.D0
      end do
      b((5-1)*ndofn+3) = 0.25D0
      b((6-1)*ndofn+3) = 0.25D0
      b((7-1)*ndofn+3) = 0.25D0
      b((8-1)*ndofn+3) = 0.25D0
      do ii = 1,ntdisp
          i = idispl(ii)
          do j = 1,ntotdf
              b(j) = b(j) - a((i-1)*ntotdf+j) * bvalue(ii)
          end do
      end do
      do ii = 1,ntdisp
          i = idispl(ii)
          write(*,*) i
          do j = 1,ntotdf
```

```
              a((j-1)*ntotdf+i) = 0.D0
              a((i-1)*ntotdf+j) = 0.D0
          end do
          a((i-1)*ntotdf+i) = 1.D0
          b(i) = bvalue(ii)
      end do
      do i = 1,ntotdf
          write(*,*) i,b(i)
      end do
c
      call ugauss_m(a,b,ntotdf)
c
      do i = 1,npoin
          write(*,1000) i,(b((i-1)*ndofn+j),j=1,3)
      end do
c
 1000 format(i6,2x,e13.6,2x,e13.6,2x,e13.6)
c
      stop
      end
c
c######################################################################
      subroutine merge(a,lnods,astiff,ntnoel,ntotdf,ndofn)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(*),lnods(*),astiff(ntnoel*ndofn,*)
      dimension ip(100)
c
      do i = 1,ntnoel
          do j = 1, ndofn
              ip((i-1)*ndofn+j) = (lnods(i)-1)*ndofn + j
          end do
      end do
c
      do j = 1,ntnoel*ndofn
          do i = 1,ntnoel*ndofn
              a((ip(j)-1)*ntotdf+ip(i)) =  a((ip(j)-1)*ntotdf+ip(i))
     1                 + astiff(i,j)
          end do
      end do
c
      return
      end
c
c######################################################################
```

```
      subroutine element(coords,MXDOFN,lnods,nint,ntnoel,ndofn,
     1                   astiff,yo,po)
c#######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(MXDOFN,*),lnods(*)
      dimension gsp(4,4),wgh(4,4),dndx(3,8),astiff(ntnoel*ndofn,*)
      dimension dmat(6,6),dndx(3,8),bmat(6,24)
c
      data gsp /  0.D0          , 0.D0          , 0.D0,          0.D0,
     1           -0.577350269189626D0, 0.577350269189626D0, 0.D0,0.D0,
     1           -0.774596669241483D0, 0.D0, 0.774596669241483D0,0.D0,
     1           -0.861136311594053D0,-0.339981043584856D0,
     1            0.339981043584856D0, 0.861136311594053D0 /
      data wgh /  2.D0          , 0.D0          , 0.D0,          0.D0,
     1            1.D0          , 1.D0          , 0.D0,          0.D0,
     1            0.555555555555556D0, 0.888888888888889D0,
     1            0.555555555555556D0, 0.D0,
     1            0.347854845137454D0, 0.652145154862546D0,
     1            0.652145154862546D0, 0.347854845137454D0 /
c
      matsize = ntnoel*ndofn
      do j = 1,matsize
          do i = 1,matsize
              astiff(i,j) = 0.D0
          end do
      end do
c
      vol = 0.D0
c
      call cal_dmat(yo,po,dmat)
c
      do ir1 = 1,nint
      do ir2 = 1,nint
      do ir3 = 1,nint
c
          r1 = gsp(ir1,nint)
          r2 = gsp(ir2,nint)
          r3 = gsp(ir3,nint)
c
          call shap3d(r1,r2,r3,coords,MXDOFN,lnods,detjac,dndx)
c
          call blu3d(dndx,bmat)
c
          detwei= detjac*wgh(ir1,nint)*wgh(ir2,nint)*wgh(ir3,nint)
          vol = vol + detwei
```

```
c
          call kuu3d(astiff,dmat,bmat,detwei)
c
      end do
      end do
      end do
c
      write(*,*) vol
c
      return
      end
c
C######################################################################
      subroutine shap3d(r1,r2,r3,coords,MXDOFN,lnods,detjac,dndx)
C######################################################################
      implicit real*8(a-h,o-z)
      dimension coords(MXDOFN,*),lnods(*)
      dimension ajacob(3,3),dndr(3,8)
      dimension dndx(3,*),ajainv(3,3)
c
      dndr(1,1) = -0.125D0 * (1.D0 - r2) * (1.D0 - r3)
      dndr(2,1) = -0.125D0 * (1.D0 - r1) * (1.D0 - r3)
      dndr(3,1) = -0.125D0 * (1.D0 - r1) * (1.D0 - r2)
c
      dndr(1,2) =  0.125D0 * (1.D0 - r2) * (1.D0 - r3)
      dndr(2,2) = -0.125D0 * (1.D0 + r1) * (1.D0 - r3)
      dndr(3,2) = -0.125D0 * (1.D0 + r1) * (1.D0 - r2)
c
      dndr(1,3) =  0.125D0 * (1.D0 + r2) * (1.D0 - r3)
      dndr(2,3) =  0.125D0 * (1.D0 + r1) * (1.D0 - r3)
      dndr(3,3) = -0.125D0 * (1.D0 + r1) * (1.D0 + r2)
c
      dndr(1,4) = -0.125D0 * (1.D0 + r2) * (1.D0 - r3)
      dndr(2,4) =  0.125D0 * (1.D0 - r1) * (1.D0 - r3)
      dndr(3,4) = -0.125D0 * (1.D0 - r1) * (1.D0 + r2)
c
      dndr(1,5) = -0.125D0 * (1.D0 - r2) * (1.D0 + r3)
      dndr(2,5) = -0.125D0 * (1.D0 - r1) * (1.D0 + r3)
      dndr(3,5) =  0.125D0 * (1.D0 - r1) * (1.D0 - r2)
c
      dndr(1,6) =  0.125D0 * (1.D0 - r2) * (1.D0 + r3)
      dndr(2,6) = -0.125D0 * (1.D0 + r1) * (1.D0 + r3)
      dndr(3,6) =  0.125D0 * (1.D0 + r1) * (1.D0 - r2)
c
      dndr(1,7) =  0.125D0 * (1.D0 + r2) * (1.D0 + r3)
```

```
      dndr(2,7) =  0.125D0 * (1.D0 + r1) * (1.D0 + r3)
      dndr(3,7) =  0.125D0 * (1.D0 + r1) * (1.D0 + r2)
c
      dndr(1,8) = -0.125D0 * (1.D0 + r2) * (1.D0 + r3)
      dndr(2,8) =  0.125D0 * (1.D0 - r1) * (1.D0 + r3)
      dndr(3,8) =  0.125D0 * (1.D0 - r1) * (1.D0 + r2)
c
      call jacb3d(dndr,coords,MXDOFN,lnods,ajacob,detjac,ajainv)
c
      do i = 1,8
          do j = 1, 3
              temp = 0.D0
              do k = 1,3
                  temp = temp + ajainv(j,k) * dndr(k,i)
              end do
              dndx(j,i) = temp
          end do
      end do
c
      return
      end
c
C######################################################################
      subroutine jacb3d(dndr,coords,MXDOFN,lnods,ajacob,detjac,ajainv)
C######################################################################
      implicit real*8 (a-h,o-z)
      dimension ajacob(3,*),ajainv(3,*)
      dimension dndr(3,*)
      dimension coords(MXDOFN,*),lnods(*)
C
      do i=1,3
          do j=1,3
              ajacob(i,j) = 0.D0
          end do
      end do
C
      do ki=1,8
          do i = 1,3
              do j = 1,3
                  ajacob(j,i)=ajacob(j,i)+dndr(j,ki)*coords(i,lnods(ki))
              end do
          end do
      end do
C
      detjac = ajacob(1,1)*ajacob(2,2)*ajacob(3,3)
```

```
     1       + ajacob(2,1)*ajacob(3,2)*ajacob(1,3)
     1       + ajacob(3,1)*ajacob(1,2)*ajacob(2,3)
     1       - ajacob(1,1)*ajacob(3,2)*ajacob(2,3)
     1       - ajacob(3,1)*ajacob(2,2)*ajacob(1,3)
     1       - ajacob(2,1)*ajacob(1,2)*ajacob(3,3)
C
      deta = 1.D0 / detjac
c
      ajainv(1,1) =
     1      ( ajacob(2,2)*ajacob(3,3)-ajacob(2,3)*ajacob(3,2) ) *deta
      ajainv(2,1) =
     1    - ( ajacob(2,1)*ajacob(3,3)-ajacob(2,3)*ajacob(3,1) ) *deta
      ajainv(3,1) =
     1      ( ajacob(2,1)*ajacob(3,2)-ajacob(2,2)*ajacob(3,1) ) *deta
C
      ajainv(1,2) =
     1    - ( ajacob(1,2)*ajacob(3,3)-ajacob(1,3)*ajacob(3,2) ) *deta
      ajainv(2,2) =
     1      ( ajacob(1,1)*ajacob(3,3)-ajacob(1,3)*ajacob(3,1) ) *deta
      ajainv(3,2) =
     1    - ( ajacob(1,1)*ajacob(3,2)-ajacob(1,2)*ajacob(3,1) ) *deta
C
      ajainv(1,3) =
     1      ( ajacob(1,2)*ajacob(2,3)-ajacob(1,3)*ajacob(2,2) ) *deta
      ajainv(2,3) =
     1    - ( ajacob(1,1)*ajacob(2,3)-ajacob(1,3)*ajacob(2,1) ) *deta
      ajainv(3,3) =
     1      ( ajacob(1,1)*ajacob(2,2)-ajacob(1,2)*ajacob(2,1) ) *deta
C
      return
      end
c
c######################################################################
      subroutine ugauss_m(a,b,n)
c######################################################################
      implicit real*8(a-h,o-z)
      dimension a(n,*),b(*)
c
      a(1,2) = a(1,2) / a(1,1)
c
      a(2,2) = a(2,2) - a(1,2) * a(1,1) * a(1,2)
c
      do j = 3,n
          do i = 2, j-1
              temp_u = 0.D0
```

```
              do k = 1, i-1
                  temp_u = temp_u + a(k,i) * a(k,j)
              end do
              a(i,j) = a(i,j) - temp_u
          end do
          do i = 1,j-1
              a(i,j) = a(i,j) / a(i,i)
          end do
          temp = 0.D0
          do k = 1,j-1
              temp = temp + a(k,j) * a(k,k) * a(k,j)
          end do
          a(j,j) = a(j,j) - temp
      end do
c
c
      do i = 2,n
          do j = 1,i-1
              b(i) = b(i) - a(j,i) * b(j)
          end do
      end do
c
      do i = 1,n
          b(i) = b(i) / a(i,i)
      end do
c
      do i = n,2,-1
          do j = 1,i-1
              b(j) = b(j) - a(j,i) * b(i)
          end do
      end do
c
      return
      end
c
C######################################################################
      subroutine cal_dmat(yo,po,dmat)
C######################################################################
      implicit real*8(a-h,o-z)
      dimension dmat(6,*)
c
      do i = 1,6
          do j = 1,6
              dmat(i,j) = 0.D0
          end do
```

```
      end do
c
      akappa = yo/3.D0/(1.D0-2.D0*po)
      g = yo/2.D0/(1.D0+po)
c
      do i = 1,3
          do j = 1,3
              dmat(i,j) = akappa-2.D0/3.D0 * g
          end do
          dmat(i,i) = akappa + 4.D0/3.D0 * g
          dmat(i+3,i+3) = g
      end do
c
      return
      end
c
C######################################################################
      subroutine kuu3d(astiff,dmat,bmat,detwei)
C######################################################################
      implicit real*8(a-h,o-z)
      dimension bmat(6,*),dmat(6,*)
      dimension astiff(3*8,*)
      dimension tmpwrk(6,24)
c
      do i = 1,6
          do k = 1,24
              temp = 0.D0
              do j = 1,6
                  temp = temp + dmat(i,j) * bmat(j,k)
              end do
              tmpwrk(i,k) = temp
          end do
      end do
c
      do i = 1,24
          do k = 1,24
              temp = 0.D0
              do j = 1,6
                  temp = temp + bmat(j,i) * tmpwrk(j,k)
              end do
              astiff(i,k) = astiff(i,k) + detwei*temp
          end do
      end do
c
      return
```

```
      end
c
C######################################################################
      subroutine blu3d(dndx,bmat)
C######################################################################
      implicit real*8(a-h,o-z)
      dimension dndx(3,*)
      dimension bmat(6,*)
c
      do i = 1, 3*8
          do j = 1,6
              bmat(j,i) = 0.D0
          end do
      end do
c
      do i = 1,8
          i1 = 1 + (i-1) * 3
          i2 = 2 + (i-1) * 3
          i3 = 3 + (i-1) * 3
c
          bmat(1,i1) = dndx(1,i)
          bmat(2,i2) = dndx(2,i)
          bmat(3,i3) = dndx(3,i)
          bmat(4,i1) = dndx(2,i)
          bmat(4,i2) = dndx(1,i)
          bmat(5,i2) = dndx(3,i)
          bmat(5,i3) = dndx(2,i)
          bmat(6,i1) = dndx(3,i)
          bmat(6,i3) = dndx(1,i)
      end do
c
      return
      end
c
```

4.2　4節点四角形ソリッド要素

4節点四角形要素も, 8節点六面体要素と全く同じ方法で構成されている. 即ち物理座標系で図4.6のような配置になっている. 節点を表4.3のように対応させる.

節点	r_1	r_2
1	-1	-1
2	1	-1
3	1	1
4	-1	1

表 4.3: 節点の対応

補間関数の具体的な形は以下のようになる.

$$N^{(1)} = \frac{1}{4}(1-r_1)(1-r_2) \tag{4.76}$$

$$N^{(2)} = \frac{1}{4}(1+r_1)(1-r_2) \tag{4.77}$$

$$N^{(3)} = \frac{1}{4}(1+r_1)(1+r_2) \tag{4.78}$$

$$N^{(4)} = \frac{1}{4}(1-r_1)(1+r_2) \tag{4.79}$$

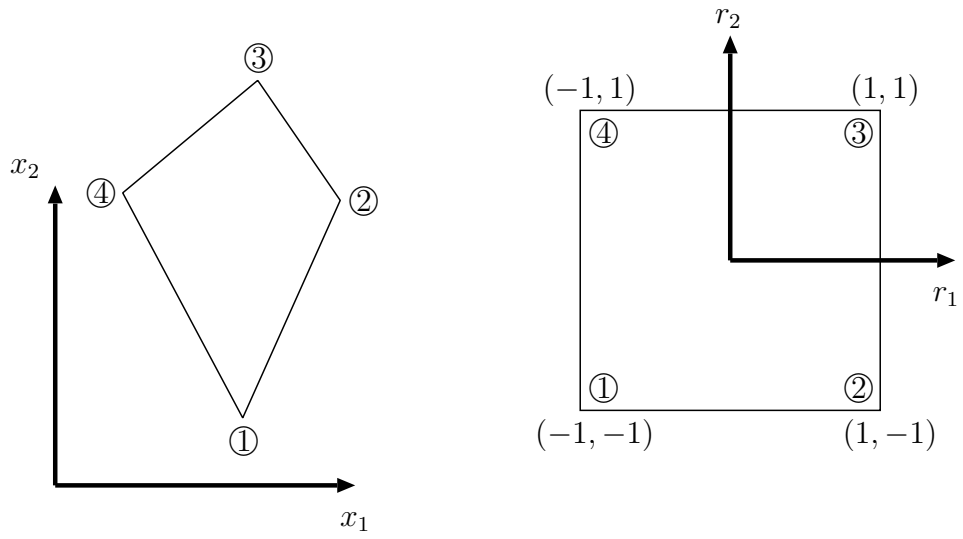
ひずみを計算する際に必要な u_i の x_j に関する微分は, 微分の連鎖則を用いることにより以下のように求められる.

$$\frac{\partial u_i}{\partial x_j} = \frac{\partial N^{(n)}}{\partial x_j} u_i^{(n)} = \left(\frac{\partial N^{(n)}}{\partial r_1}\frac{\partial r_1}{\partial x_j} + \frac{\partial N^{(n)}}{\partial r_2}\frac{\partial r_2}{\partial x_j} \right) u_i^{(n)} \tag{4.80}$$

ここで, $\dfrac{\partial N^{(n)}}{\partial x_j}$ はやはり微分の連鎖則を用いて, 以下のように求められる.
まず以下のようにヤコビマトリックス $[J]$ を求める.

$$\begin{bmatrix} \dfrac{\partial N^{(n)}}{\partial r_1} \\ \dfrac{\partial N^{(n)}}{\partial r_2} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial x_1}{\partial r_1} & \dfrac{\partial x_2}{\partial r_1} \\ \dfrac{\partial x_1}{\partial r_2} & \dfrac{\partial x_2}{\partial r_2} \end{bmatrix} \begin{bmatrix} \dfrac{\partial N^{(n)}}{\partial x_1} \\ \dfrac{\partial N^{(n)}}{\partial x_2} \end{bmatrix} \tag{4.81}$$

$$= [J] \begin{bmatrix} \dfrac{\partial N^{(n)}}{\partial x_1} \\ \dfrac{\partial N^{(n)}}{\partial x_2} \end{bmatrix} \tag{4.82}$$



(a) 物理座標系　　(b) 自然座標系

図 4.6: 物理座標系と自然座標系の対応

このヤコビマトリックスの各成分 $\dfrac{\partial x_i}{\partial r_j}$ は以下のようになる.

$$\frac{\partial x_i}{\partial r_j} = \frac{\partial N^{(n)}}{\partial r_j} x_i^{(n)} \tag{4.83}$$

このヤコビマトリックスを用いれば $\dfrac{\partial N^{(n)}}{\partial x_i}$ は以下のように求められる.

$$\begin{bmatrix} \dfrac{\partial N^{(n)}}{\partial x_1} \\ \dfrac{\partial N^{(n)}}{\partial x_2} \end{bmatrix} = [J]^{-1} \begin{bmatrix} \dfrac{\partial N^{(n)}}{\partial r_1} \\ \dfrac{\partial N^{(n)}}{\partial r_2} \end{bmatrix} \tag{4.84}$$

なおここで補間関数の微分は以下のようになる.

$$\frac{\partial N^{(1)}}{\partial r_1} = -\frac{1}{4}(1-r_2) \qquad (4.85) \qquad \frac{\partial N^{(3)}}{\partial r_1} = \frac{1}{4}(1+r_2) \qquad (4.89)$$

$$\frac{\partial N^{(1)}}{\partial r_2} = -\frac{1}{4}(1-r_1) \qquad (4.86) \qquad \frac{\partial N^{(3)}}{\partial r_2} = \frac{1}{4}(1+r_1) \qquad (4.90)$$

$$\frac{\partial N^{(2)}}{\partial r_1} = \frac{1}{4}(1-r_2) \qquad (4.87) \qquad \frac{\partial N^{(4)}}{\partial r_1} = -\frac{1}{4}(1+r_2) \qquad (4.91)$$

$$\frac{\partial N^{(2)}}{\partial r_2} = -\frac{1}{4}(1+r_1) \qquad (4.88) \qquad \frac{\partial N^{(4)}}{\partial r_2} = \frac{1}{4}(1-r_1) \qquad (4.92)$$

このヤコビマトリックスを用いると, 領域積分は以下のように表される.

$$\int_{\Omega_e} d\Omega = \int_{-1}^{1}\int_{-1}^{1} \det[J] dr_1\, dr_2 \qquad (4.93)$$

この積分は通常, Gauss 積分などの数値積分法により実行される. ここでは, 3 次元の場合と同様に 1 次元の Gauss 積分を二重にしたものを用いることにする. 即ち,

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)\, dx\, dy \approx \sum_i \sum_j w_i w_j f(x_i, y_j) \qquad (4.94)$$

4.3 Lagrange 族

2.2 節では, 近似の精度を上げるため, 線分内に 3 つの節点をとり以下の補間関数を導入した.

$$N^{(1)} = -\frac{1}{2} r(1-r) \qquad (4.95)$$

$$N^{(2)} = \frac{1}{2} r(1+r) \qquad (4.96)$$

$$N^{(3)} = 1 - r^2 \qquad (4.97)$$

これを $r(-1 \le r \le 1)$ の代わりに $r_1, r_2(-1 \le r_1 \le 1, -1 \le r_2 \le 1)$ と 2 変数使用し, それにともなって補間関数を掛け合わすと, 9 節点の四角形要素を構成できる. 物理座標系と自然座標系の対応は図 4.7 のようになっている.

補間関数及び補間関数の微分の具体的な形は, 1 次元の補間関数を以下のように表せば

節点	r_1	r_2
1	-1	-1
2	1	-1
3	1	1
4	-1	1
5	0	-1
6	1	0
7	0	1
8	-1	0
9	0	0

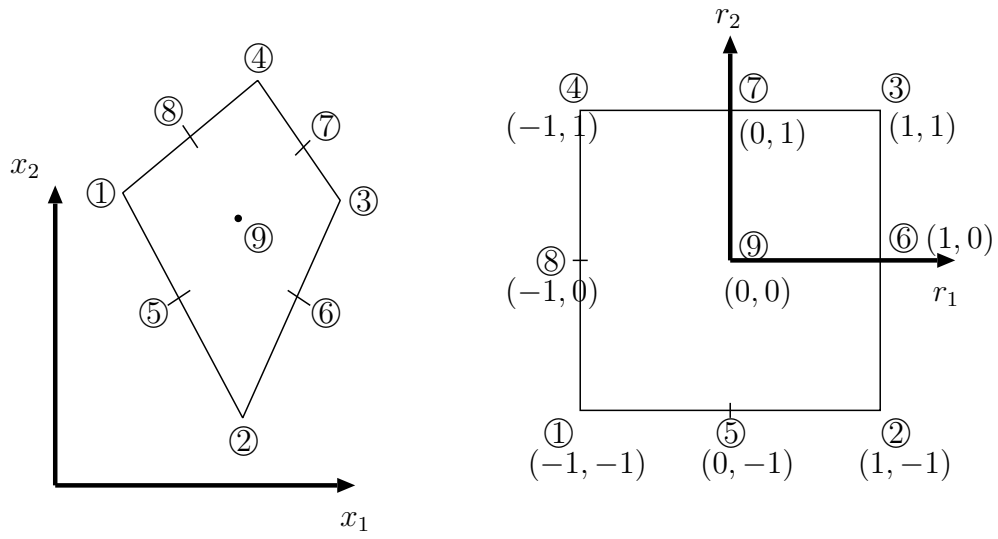
表 4.4: 節点の対応

系統的に表すことができる.

$$N_{i-} = -\frac{1}{2} r_i (1 - r_i) \tag{4.98}$$

$$N_{i0} = 1 - r_i^2 \tag{4.99}$$

$$N_{i+} = \frac{1}{2} r_i (1 + r_i) \tag{4.100}$$



(a) 物理座標系　　(b) 自然座標系

図 4.7: 物理座標系と自然座標系の対応

$$N^{(1)} = N_{1-}N_{2-} = \frac{1}{4}r_1r_2(1-r_1)(1-r_2) \tag{4.101}$$

$$N^{(2)} = N_{1+}N_{2-} = -\frac{1}{4}r_1r_2(1+r_1)(1-r_2) \tag{4.102}$$

$$N^{(3)} = N_{1+}N_{2+} = \frac{1}{4}r_1r_2(1+r_1)(1+r_2) \tag{4.103}$$

$$N^{(4)} = N_{1-}N_{2+} = -\frac{1}{4}r_1r_2(1-r_1)(1+r_2) \tag{4.104}$$

$$N^{(5)} = N_{10}N_{2-} = -\frac{1}{2}r_2(1-r_1^2)(1-r_2) \tag{4.105}$$

$$N^{(6)} = N_{1+}N_{20} = \frac{1}{2}r_1(1+r_1)(1-r_2^2) \tag{4.106}$$

$$N^{(7)} = N_{10}N_{2+} = \frac{1}{2}r_2(1-r_1^2)(1+r_2) \tag{4.107}$$

$$N^{(8)} = N_{1-}N_{20} = -\frac{1}{2}r_1(1-r_1)(1-r_2^2) \tag{4.108}$$

$$N^{(9)} = N_{10}N_{20} = (1-r_1^2)(1-r_2^2) \tag{4.109}$$

$$\frac{\partial N^{(1)}}{\partial r_1} = \frac{\partial N_{1-}}{\partial r_1} N_{2-} \tag{4.110}$$

$$\frac{\partial N^{(1)}}{\partial r_2} = N_{1-} \frac{\partial N_{2-}}{\partial r_2} \tag{4.111}$$

$$\frac{\partial N^{(2)}}{\partial r_1} = \frac{\partial N_{1+}}{\partial r_1} N_{2-} \tag{4.112}$$

$$\frac{\partial N^{(2)}}{\partial r_2} = N_{1+} \frac{\partial N_{2-}}{\partial r_2} \tag{4.113}$$

$$\frac{\partial N^{(3)}}{\partial r_1} = \frac{\partial N_{1+}}{\partial r_1} N_{2+} \tag{4.114}$$

$$\frac{\partial N^{(3)}}{\partial r_2} = N_{1+} \frac{\partial N_{2+}}{\partial r_2} \tag{4.115}$$

$$\frac{\partial N^{(4)}}{\partial r_1} = \frac{\partial N_{1-}}{\partial r_1} N_{2+} \tag{4.116}$$

$$\frac{\partial N^{(4)}}{\partial r_2} = N_{1-} \frac{\partial N_{2+}}{\partial r_2} \tag{4.117}$$

$$\frac{\partial N^{(5)}}{\partial r_1} = \frac{\partial N_{10}}{\partial r_1} N_{2-} \tag{4.118}$$

$$\frac{\partial N^{(5)}}{\partial r_2} = N_{10} \frac{\partial N_{2-}}{\partial r_2} \tag{4.119}$$

$$\frac{\partial N^{(6)}}{\partial r_1} = \frac{\partial N_{1+}}{\partial r_1} N_{20} \tag{4.120}$$

$$\frac{\partial N^{(6)}}{\partial r_2} = N_{1+} \frac{\partial N_{20}}{\partial r_2} \tag{4.121}$$

$$\frac{\partial N^{(7)}}{\partial r_1} = \frac{\partial N_{10}}{\partial r_1} N_{2+} \tag{4.122}$$

$$\frac{\partial N^{(7)}}{\partial r_2} = N_{10} \frac{\partial N_{2+}}{\partial r_2} \tag{4.123}$$

$$\frac{\partial N^{(8)}}{\partial r_1} = \frac{\partial N_{1-}}{\partial r_1} N_{20} \tag{4.124}$$

$$\frac{\partial N^{(8)}}{\partial r_2} = N_{1-} \frac{\partial N_{20}}{\partial r_2} \tag{4.125}$$

$$\frac{\partial N^{(9)}}{\partial r_1} = \frac{\partial N_{10}}{\partial r_1} N_{20} \tag{4.126}$$

$$\frac{\partial N^{(9)}}{\partial r_2} = N_{10} \frac{\partial N_{20}}{\partial r_2} \tag{4.127}$$

以上の議論は, ベースにする 1 次元の補間関数としてより高次のものを用いても全く同じで, 更に 3 次元に拡張することも容易である. このような要素を一般に Lagrange 族と呼ぶ.

Lagrange 族の名称の由来は補間関数に Lagrange の多項式が用いられていることにある. Lagrange の多項式は, 一般形で以下のような形式になっている.

$$H_i(x) = \frac{(x - x_{(1)})(x - x_{(2)}) \dots (x - x_{(i-1)})(x - x_{(i+1)}) \dots (x - x_{(n)})}{(x_{(i)} - x_{(1)})(x_{(i)} - x_{(2)}) \dots (x_{(i)} - x_{(i-1)})(x_{(i)} - x_{(i+1)}) \dots (x_{(i)} - x_{(n)})} \tag{4.128}$$

これは

$$H_i(x_{(j)}) = \delta_{ij} \tag{4.129}$$

であることから,

$$\phi(x) \approx \sum_i \phi(x_i) H_i(x) \tag{4.130}$$

のように関数 ϕ を近似することができる.

実際に Lagrange の多項式から 1 次元の補間関数を求めると, 以下のようになる.

$n = 2 \quad (x_{(1)} = -1, x_{(2)} = 1)$ の場合

$$H_1(x) = \frac{1}{2}(1 - x) \tag{4.131}$$

$$H_2(x) = \frac{1}{2}(1 + x) \tag{4.132}$$

$n = 3 \quad (x_{(1)} = -1, x_{(2)} = 1, x_{(3)} = 0)$ の場合

$$H_1 = \frac{1}{2}x(x - 1) \tag{4.133}$$

$$H_2 = \frac{1}{2}x(x + 1) \tag{4.134}$$

$$H_3 = 1 - x^2 \tag{4.135}$$

$n = 4 \quad (x_{(1)} = -1, x_{(2)} = 1, x_{(3)} = -\frac{1}{3}, x_{(4)} = \frac{1}{3})$ の場合

$$H_1 = -\frac{1}{16}(x - 1)(9x^2 - 1) \tag{4.136}$$

$$H_2 = \quad \frac{1}{16}(x + 1)(9x^2 - 1) \tag{4.137}$$

$$H_3 = \quad \frac{9}{16}(x^2 - 1)(3x - 1) \tag{4.138}$$

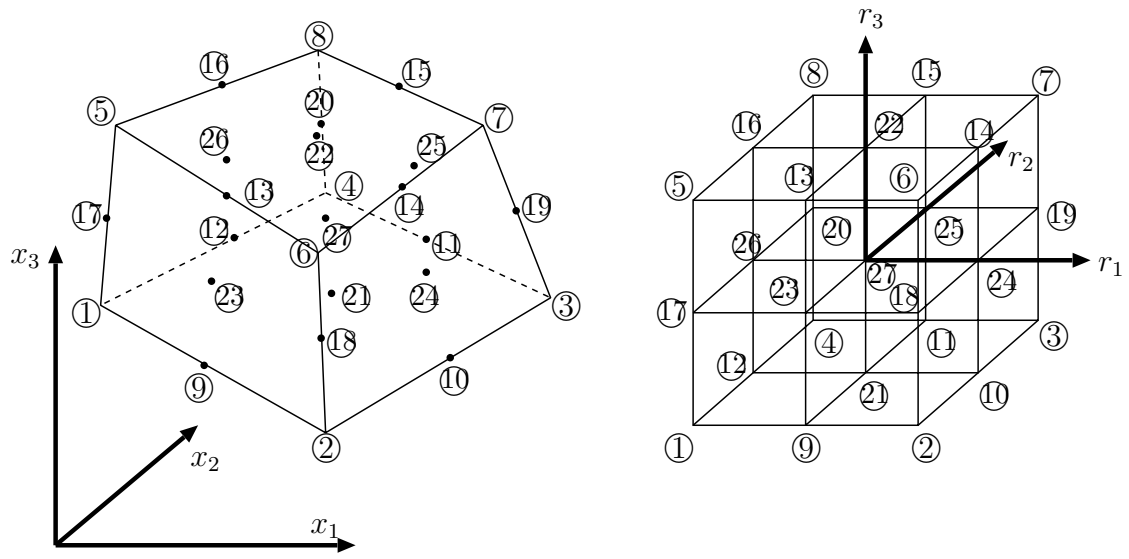
$$H_4 = -\frac{9}{16}(x^2 - 1)(3x + 1) \tag{4.139}$$

ただし, 一般には $n = 2$ 以上の 2 次元, 3 次元の補間関数は計算コストが大きすぎるためあまり用いられない. $n = 2$ の 3 次元の補間関数は表 4.5, 図 4.8 のようになる.

節点	r_1	r_2	r_3	節点	r_1	r_2	r_3
1	-1	-1	-1	15	0	1	1
2	1	-1	-1	16	-1	0	1
3	1	1	-1	17	-1	-1	0
4	-1	1	-1	18	1	-1	0
5	-1	-1	1	19	1	1	0
6	1	-1	1	20	-1	1	0
7	1	1	1	21	0	0	-1
8	-1	1	1	22	0	0	1
9	0	-1	-1	23	0	-1	0
10	1	0	-1	24	1	0	0
11	0	1	-1	25	0	1	0
12	-1	0	-1	26	-1	0	0
13	0	-1	1	27	0	0	0
14	1	0	1				

表 4.5: 節点の対応

$$N^{(1)} = N_{1-}N_{2-}N_{3-} \quad (4.140)$$
$$N^{(2)} = N_{1+}N_{2-}N_{3-} \quad (4.141)$$
$$N^{(3)} = N_{1+}N_{2+}N_{3-} \quad (4.142)$$
$$N^{(4)} = N_{1-}N_{2+}N_{3-} \quad (4.143)$$
$$N^{(5)} = N_{1-}N_{2-}N_{3+} \quad (4.144)$$
$$N^{(6)} = N_{1+}N_{2-}N_{3+} \quad (4.145)$$
$$N^{(7)} = N_{1+}N_{2+}N_{3+} \quad (4.146)$$
$$N^{(8)} = N_{1-}N_{2+}N_{3+} \quad (4.147)$$
$$N^{(9)} = N_{10}N_{2-}N_{3-} \quad (4.148)$$
$$N^{(10)} = N_{1+}N_{20}N_{3-} \quad (4.149)$$
$$N^{(11)} = N_{10}N_{2+}N_{3-} \quad (4.150)$$
$$N^{(12)} = N_{1-}N_{20}N_{3-} \quad (4.151)$$
$$N^{(13)} = N_{10}N_{2-}N_{3+} \quad (4.152)$$
$$N^{(14)} = N_{1-}N_{20}N_{3+} \quad (4.153)$$
$$N^{(15)} = N_{10}N_{2+}N_{3+} \quad (4.154)$$
$$N^{(16)} = N_{1-}N_{20}N_{3+} \quad (4.155)$$
$$N^{(17)} = N_{1-}N_{2-}N_{30} \quad (4.156)$$
$$N^{(18)} = N_{1+}N_{2-}N_{30} \quad (4.157)$$
$$N^{(19)} = N_{1+}N_{2+}N_{30} \quad (4.158)$$
$$N^{(20)} = N_{1-}N_{2+}N_{30} \quad (4.159)$$
$$N^{(21)} = N_{10}N_{20}N_{3-} \quad (4.160)$$
$$N^{(22)} = N_{10}N_{20}N_{3+} \quad (4.161)$$
$$N^{(23)} = N_{10}N_{2-}N_{30} \quad (4.162)$$
$$N^{(24)} = N_{1+}N_{20}N_{30} \quad (4.163)$$
$$N^{(25)} = N_{10}N_{2+}N_{30} \quad (4.164)$$
$$N^{(26)} = N_{1-}N_{20}N_{30} \quad (4.165)$$
$$N^{(27)} = N_{10}N_{20}N_{30} \quad (4.166)$$



(a) 物理座標系　　(b) 自然座標系

図 4.8: 物理座標系と自然座標系の対応

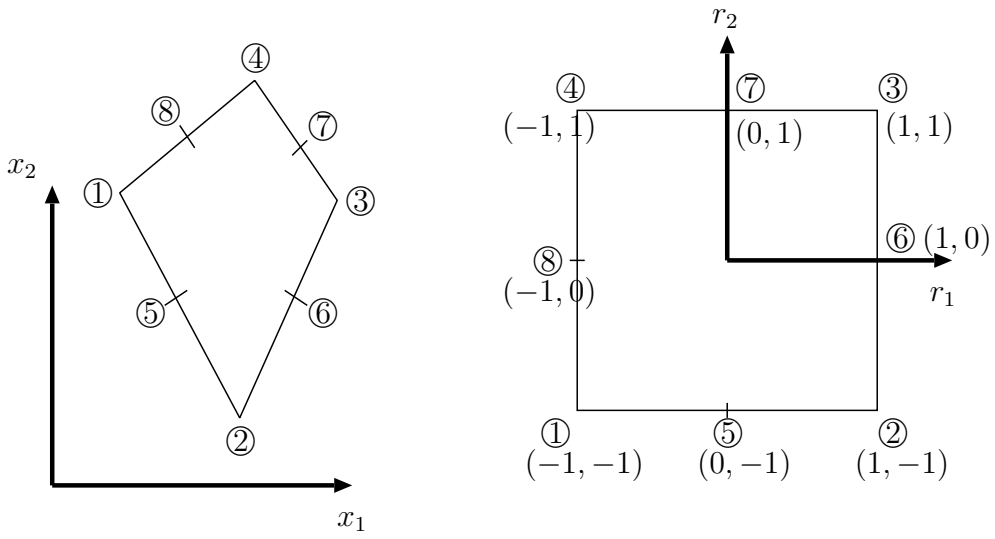
4.4 serendipity 族

Lagrange 族の補間関数は構成が明解でコーディングも系統的に行うことができるが, 特に 3 次元の場合には自由度が多い分だけ計算負荷がかかる. そこで四角形の辺上にしか節点がない要素を構成する.

このような要素は serendipity 族と呼ばれている. 四角形の場合は 8 節点で, 補間関数の具体的な形は表 4.6, 図 4.9 のようになる.

節点	r_1	r_2	節点	r_1	r_2
1	-1	-1	5	0	-1
2	1	-1	6	1	0
3	1	1	7	0	1
4	-1	1	8	-1	0

表 4.6: 節点の対応



(a) 物理座標系　　(b) 自然座標系

図 4.9: 物理座標系と自然座標系の対応

$$N^{(1)} = \frac{1}{4}(1-r_1)(1-r_2)(-1-r_1-r_2) \tag{4.167}$$

$$N^{(2)} = \frac{1}{4}(1+r_1)(1-r_2)(-1+r_1-r_2) \tag{4.168}$$

$$N^{(3)} = \frac{1}{4}(1+r_1)(1+r_2)(-1+r_1+r_2) \tag{4.169}$$

$$N^{(4)} = \frac{1}{4}(1-r_1)(1+r_2)(-1-r_1+r_2) \tag{4.170}$$

$$N^{(5)} = \frac{1}{2}(1-r_2)(1-r_1^2) \tag{4.171}$$

$$N^{(6)} = \frac{1}{2}(1+r_1)(1-r_2^2) \tag{4.172}$$

$$N^{(7)} = \frac{1}{2}(1+r_2)(1-r_1^2) \tag{4.173}$$

$$N^{(8)} = \frac{1}{2}(1-r_1)(1-r_2^2) \tag{4.174}$$

補間関数の微分は以下のようになる.

$$\frac{\partial N^{(1)}}{\partial r_1} = \frac{1}{4}(1-r_2)(2r_1+r_2) \quad (4.175) \qquad \frac{\partial N^{(5)}}{\partial r_1} = -r_1(1-r_2) \quad (4.183)$$

$$\frac{\partial N^{(1)}}{\partial r_2} = \frac{1}{4}(1-r_1)(2r_2+r_1) \quad (4.176) \qquad \frac{\partial N^{(5)}}{\partial r_2} = -\frac{1}{2}(1-r_1^2) \quad (4.184)$$

$$\frac{\partial N^{(2)}}{\partial r_1} = \frac{1}{4}(1-r_2)(2r_1-r_2) \quad (4.177) \qquad \frac{\partial N^{(6)}}{\partial r_1} = \frac{1}{2}(1-r_2^2) \quad (4.185)$$

$$\frac{\partial N^{(2)}}{\partial r_2} = \frac{1}{4}(1+r_1)(2r_2-r_1) \quad (4.178) \qquad \frac{\partial N^{(6)}}{\partial r_2} = -r_2(1+r_1) \quad (4.186)$$

$$\frac{\partial N^{(3)}}{\partial r_1} = \frac{1}{4}(1+r_2)(2r_1+r_2) \quad (4.179) \qquad \frac{\partial N^{(7)}}{\partial r_1} = -r_1(1+r_2) \quad (4.187)$$

$$\frac{\partial N^{(3)}}{\partial r_2} = \frac{1}{4}(1+r_1)(2r_2+r_1) \quad (4.180) \qquad \frac{\partial N^{(7)}}{\partial r_2} = \frac{1}{2}(1-r_1^2) \quad (4.188)$$

$$\frac{\partial N^{(4)}}{\partial r_1} = \frac{1}{4}(1+r_2)(2r_1-r_2) \quad (4.181) \qquad \frac{\partial N^{(8)}}{\partial r_1} = -\frac{1}{2}(1-r_2^2) \quad (4.189)$$

$$\frac{\partial N^{(4)}}{\partial r_2} = \frac{1}{4}(1-r_1)(2r_2-r_1) \quad (4.182) \qquad \frac{\partial N^{(8)}}{\partial r_2} = -r_2(1-r_1) \quad (4.190)$$

この補間関数は当初偶然発見され, それ故 serendipity 族と呼ばれているが, その後の研究によってここに示すような補間関数を系統的に構成する方法が確立している.

まず 4 節点の補間関数をもとに, 8 節点の補間関数をつくることを考える. 4 節点の補間関数は以下のようになっている.

$$N^{(1)} = \frac{1}{4}(1-r_1)(1-r_2) \quad (4.191)$$

$$N^{(2)} = \frac{1}{4}(1+r_1)(1-r_2) \quad (4.192)$$

$$N^{(3)} = \frac{1}{4}(1+r_1)(1+r_2) \quad (4.193)$$

$$N^{(4)} = \frac{1}{4}(1-r_1)(1+r_2) \quad (4.194)$$

これらは対応する節点で 1, その他の節点では 0 の値をとるという特徴がある. 今 5 番目の節点として, 自然座標系で $(0,-1)$ の点を考える. このとき,

$$N^{(5)} = \frac{1}{2}(1-r_2)(1-r_1^2) \quad (4.195)$$

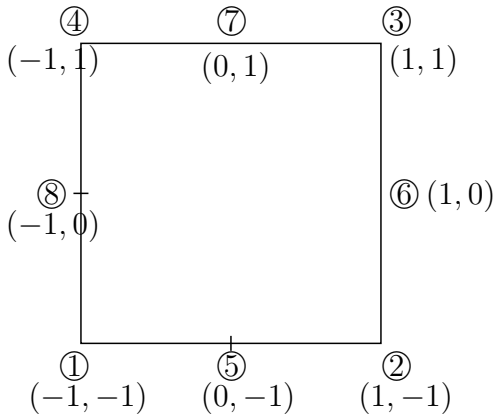


図 4.10: 8 節点要素の構成

という関数を考えれば, これは, ⑤で 1, ①∼④では 0 になる. ただし $N^{(1)} \sim N^{(4)}$ は⑤では 0 にならないので, 以下のように新たな補間関数を作る.

$$\tilde{N}^{(1)} = N^{(1)} - \frac{1}{2}N^{(5)} \tag{4.196}$$

$$\tilde{N}^{(2)} = N^{(2)} - \frac{1}{2}N^{(5)} \tag{4.197}$$

$$\tilde{N}^{(3)} = N^{(3)} \tag{4.198}$$

$$\tilde{N}^{(4)} = N^{(4)} \tag{4.199}$$

このとき, $\tilde{N}^{(1)} \sim \tilde{N}^{(4)}$ は対応する節点で 1, 他の節点は⑤を含めて 0 になっている.

同様に 6, 7, 8 番目の節点としてそれぞれ (1,0), (0,1), (-1,0) を考える. 対応する補間関数は以下のようになる.

$$N^{(6)} = \frac{1}{2}(1 + r_1)(1 - r_2^2) \tag{4.200}$$

$$N^{(7)} = \frac{1}{2}(1 + r_2)(1 - r_1^2) \tag{4.201}$$

$$N^{(8)} = \frac{1}{2}(1 - r_1)(1 - r_2^2) \tag{4.202}$$

このとき,

$$\tilde{N}^{(1)} = N^{(1)} - \frac{1}{2}N^{(5)} - \frac{1}{2}N^{(8)} \tag{4.203}$$

$$\tilde{N}^{(2)} = N^{(2)} - \frac{1}{2}N^{(5)} - \frac{1}{2}N^{(6)} \tag{4.204}$$

$$\tilde{N}^{(3)} = N^{(3)} - \frac{1}{2}N^{(6)} - \frac{1}{2}N^{(7)} \tag{4.205}$$

$$\tilde{N}^{(4)} = N^{(4)} - \frac{1}{2}N^{(7)} - \frac{1}{2}N^{(8)} \tag{4.206}$$

とすれば, 対応する節点で 1, 他の節点では 0 になる. 実際にこれを計算すると,

$$N^{(1)} = \frac{1}{4}(1-r_1)(1-r_2)(-1-r_1-r_2) \tag{4.207}$$

$$N^{(2)} = \frac{1}{4}(1+r_1)(1-r_2)(-1+r_1-r_2) \tag{4.208}$$

$$N^{(3)} = \frac{1}{4}(1+r_1)(1+r_2)(-1+r_1+r_2) \tag{4.209}$$

$$N^{(4)} = \frac{1}{4}(1-r_1)(1+r_2)(-1-r_1+r_2) \tag{4.210}$$

となり, 前述の 8 節点要素の形状関数と一致する.

さらに 8 節点の要素から 9 節点の要素をつくることもできる. 9 番目の節点を (0,0) にとると,

$$N^{(9)} = (1-r_1^2)(1-r_2^2) \tag{4.211}$$

とすれば対応する節点で 1, 他の節点では 0 になる. これをもとに $N^{(1)} \sim N^{(8)}$ を以下のようにすれば 9 節点の補間関数が得られる.

$$N^{(1)} = \frac{1}{4}(1-r_1)(1-r_2)(-1-r_1-r_2) + \frac{1}{4}N^{(9)} \tag{4.212}$$

$$N^{(2)} = \frac{1}{4}(1+r_1)(1-r_2)(-1+r_1-r_2) + \frac{1}{4}N^{(9)} \tag{4.213}$$

$$N^{(3)} = \frac{1}{4}(1+r_1)(1+r_2)(-1+r_1+r_2) + \frac{1}{4}N^{(9)} \tag{4.214}$$

$$N^{(4)} = \frac{1}{4}(1-r_1)(1+r_2)(-1-r_1+r_2) + \frac{1}{4}N^{(9)} \tag{4.215}$$

$$N^{(5)} = \frac{1}{2}(1-r_2)(1-r_1^2) - \frac{1}{2}N^{(9)} \tag{4.216}$$

$$N^{(6)} = \frac{1}{2}(1+r_1)(1-r_2^2) - \frac{1}{2}N^{(9)} \tag{4.217}$$

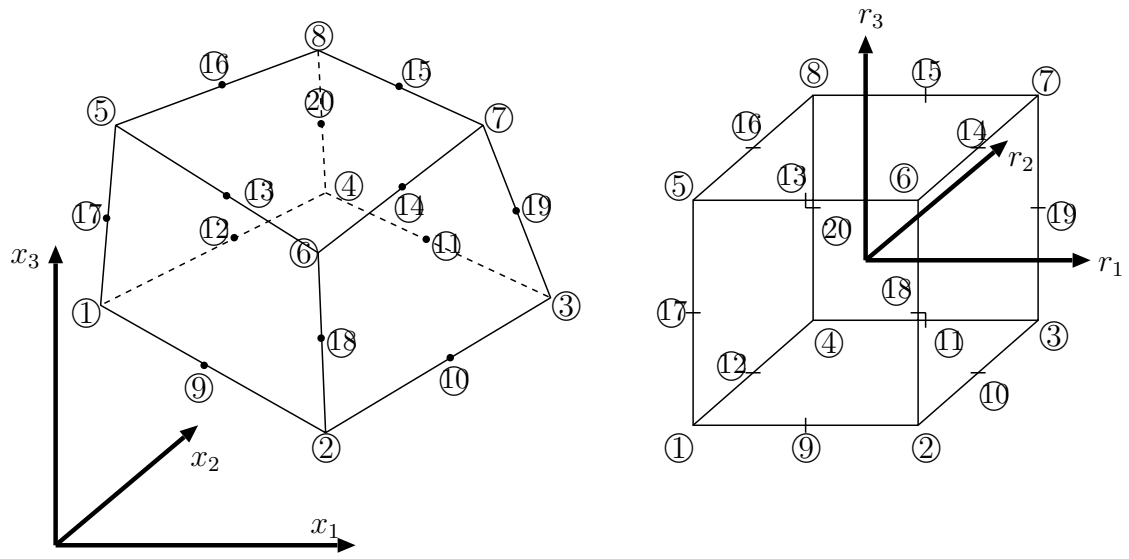
$$N^{(7)} = \frac{1}{2}(1+r_2)(1-r_1^2) - \frac{1}{2}N^{(9)} \tag{4.218}$$

$$N^{(8)} = \frac{1}{2}(1-r_1)(1-r_2^2) - \frac{1}{2}N^{(9)} \tag{4.219}$$

この要素の 3 次元版は 20 節点である. 補間関数の具体的な形は表 (4.7), 図 (4.11) のようになる.

節点	r_1	r_2	r_3	節点	r_1	r_2	r_3
1	−1	−1	−1	11	0	1	−1
2	1	−1	−1	12	−1	0	−1
3	1	1	−1	13	0	−1	1
4	−1	1	−1	14	1	0	1
5	−1	−1	1	15	0	1	1
6	1	−1	1	16	−1	0	1
7	1	1	1	17	−1	−1	0
8	−1	1	1	18	1	−1	0
9	0	−1	−1	19	1	1	0
10	1	0	−1	20	−1	1	0

表 4.7: 節点の対応



(a) 物理座標系　　(b) 自然座標系

図 4.11: 物理座標系と自然座標系の対応

$$N^{(1)} = \frac{1}{8}(1-r_1)(1-r_2)(1-r_3)(-2-r_1-r_2-r_3) \tag{4.220}$$

$$N^{(2)} = \frac{1}{8}(1+r_1)(1-r_2)(1-r_3)(-2+r_1-r_2-r_3) \tag{4.221}$$

$$N^{(3)} = \frac{1}{8}(1+r_1)(1+r_2)(1-r_3)(-2+r_1+r_2-r_3) \tag{4.222}$$

$$N^{(4)} = \frac{1}{8}(1-r_1)(1+r_2)(1-r_3)(-2-r_1+r_2-r_3) \tag{4.223}$$

$$N^{(5)} = \frac{1}{8}(1-r_1)(1-r_2)(1+r_3)(-2-r_1-r_2+r_3) \tag{4.224}$$

$$N^{(6)} = \frac{1}{8}(1+r_1)(1-r_2)(1+r_3)(-2+r_1-r_2+r_3) \tag{4.225}$$

$$N^{(7)} = \frac{1}{8}(1+r_1)(1+r_2)(1+r_3)(-2+r_1+r_2+r_3) \tag{4.226}$$

$$N^{(8)} = \frac{1}{8}(1-r_1)(1+r_2)(1+r_3)(-2-r_1+r_2+r_3) \tag{4.227}$$

$$N^{(9)} = \frac{1}{4}(1-r_1^2)(1-r_2)(1-r_3) \tag{4.228}$$

$$N^{(10)} = \frac{1}{4}(1+r_1)(1-r_2^2)(1-r_3) \tag{4.229}$$

$$N^{(11)} = \frac{1}{4}(1-r_1^2)(1+r_2)(1-r_3) \tag{4.230}$$

$$N^{(12)} = \frac{1}{4}(1-r_1)(1-r_2^2)(1-r_3) \tag{4.231}$$

$$N^{(13)} = \frac{1}{4}(1-r_1^2)(1-r_2)(1+r_3) \tag{4.232}$$

$$N^{(14)} = \frac{1}{4}(1+r_1)(1-r_2^2)(1+r_3) \tag{4.233}$$

$$N^{(15)} = \frac{1}{4}(1-r_1^2)(1+r_2)(1+r_3) \tag{4.234}$$

$$N^{(16)} = \frac{1}{4}(1-r_1)(1-r_2^2)(1+r_3) \tag{4.235}$$

$$N^{(17)} = \frac{1}{4}(1-r_1)(1-r_2)(1-r_3^2) \tag{4.236}$$

$$N^{(18)} = \frac{1}{4}(1+r_1)(1-r_2)(1-r_3^2) \tag{4.237}$$

$$N^{(19)} = \frac{1}{4}(1+r_1)(1+r_2)(1-r_3^2) \tag{4.238}$$

$$N^{(20)} = \frac{1}{4}(1-r_1)(1+r_2)(1-r_3^2) \tag{4.239}$$

この補間関数はやはり 8 節点六面体要素のものから構成できる．8 節点の $N^{(1)} \sim N^{(8)}$ に対し

$$N^{(9)} = \sim \tag{4.240}$$

$$N^{(10)} = \sim \tag{4.241}$$

$$N^{(11)} = \sim \tag{4.242}$$

$$\vdots \tag{4.243}$$

$N^{(9)} \sim N^{(20)}$ を上記の式 (4.228) $\sim$ (4.239) のように定義すれば, $N^{(9)} \sim N^{(20)}$ は対応する節点で 1, 他は 0 である．これをもとに以下のように変更することで 20 節点の補間関数を構成することができる．

⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳

$$N^{(1)} = N^{(1)} - \frac{1}{2}N^{(9)} \quad -\frac{1}{2}N^{(12)} \quad -\frac{1}{2}N^{(17)} \tag{4.244}$$

$$N^{(2)} = N^{(2)} - \frac{1}{2}N^{(9)} - \frac{1}{2}N^{(10)} \quad -\frac{1}{2}N^{(18)} \tag{4.245}$$

$$N^{(3)} = N^{(3)} \quad -\frac{1}{2}N^{(10)} - \frac{1}{2}N^{(11)} \quad -\frac{1}{2}N^{(19)} \tag{4.246}$$

$$N^{(4)} = N^{(4)} \quad -\frac{1}{2}N^{(11)} - \frac{1}{2}N^{(12)} \quad -\frac{1}{2}N^{(20)} \tag{4.247}$$

$$N^{(5)} = N^{(5)} \quad -\frac{1}{2}N^{(13)} \quad -\frac{1}{2}N^{(16)} - \frac{1}{2}N^{(17)} \tag{4.248}$$

$$N^{(6)} = N^{(6)} \quad -\frac{1}{2}N^{(13)} - \frac{1}{2}N^{(14)} \quad -\frac{1}{2}N^{(18)} \tag{4.249}$$

$$N^{(7)} = N^{(7)} \quad -\frac{1}{2}N^{(14)} - \frac{1}{2}N^{(15)} \quad -\frac{1}{2}N^{(19)} \tag{4.250}$$

$$N^{(8)} = N^{(8)} \quad -\frac{1}{2}N^{(15)} - \frac{1}{2}N^{(16)} \quad -\frac{1}{2}N^{(20)} \tag{4.251}$$

4.5 三角形要素の補間関数と数値積分

三角形要素の補間関数は以下のように定義される面積座標系で表される.

$$L_1 + L_2 + L_3 = 1 \tag{4.252}$$

自然座標系 r_1, r_2 と L_1, L_2, L_3 を以下のように対応させる.

$$r_1 = L_1 \tag{4.253}$$

$$r_2 = L_2 \tag{4.254}$$

$$1 - r_1 - r_2 = L_3 \tag{4.255}$$

三角形内部の領域は, 図 (4.13) のような自然座標系の領域と対応している.

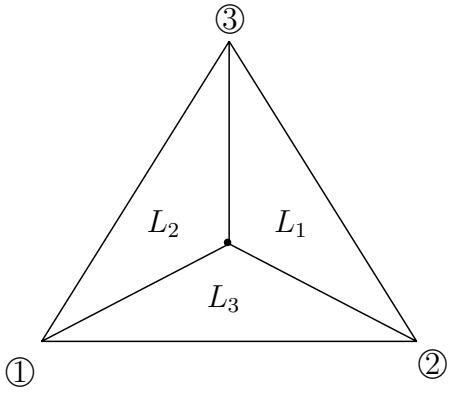


図 4.12: 三角形の面積座標

物理空間での積分 $\int_V dV$ を自然座標系に変換する.

$$\begin{bmatrix} \dfrac{\partial N^{(i)}}{\partial r_1} \\ \dfrac{\partial N^{(i)}}{\partial r_2} \end{bmatrix} = \underbrace{\begin{bmatrix} \dfrac{\partial x}{\partial r_1} & \dfrac{\partial y}{\partial r_1} \\ \dfrac{\partial x}{\partial r_2} & \dfrac{\partial y}{\partial r_2} \end{bmatrix}}_{[J]} \begin{bmatrix} \dfrac{\partial N^{(i)}}{\partial x} \\ \dfrac{\partial N^{(i)}}{\partial y} \end{bmatrix} \tag{4.256}$$

すなわち

$$\begin{bmatrix} \dfrac{\partial N^{(i)}}{\partial x} \\ \dfrac{\partial N^{(i)}}{\partial y} \end{bmatrix} = [J^{-1}] \begin{bmatrix} \dfrac{\partial N^{(i)}}{\partial r_1} \\ \dfrac{\partial N^{(i)}}{\partial r_2} \end{bmatrix} \tag{4.257}$$

$\dfrac{\partial x}{\partial r_1}, \dfrac{\partial y}{\partial r_1}, \dfrac{\partial x}{\partial r_2}, \dfrac{\partial y}{\partial r_2}$ はアイソパラメトリック要素であれば

$$\frac{\partial x}{\partial r_1} = \sum \frac{\partial N^{(i)}}{\partial r_1} x_i \ldots \tag{4.258}$$

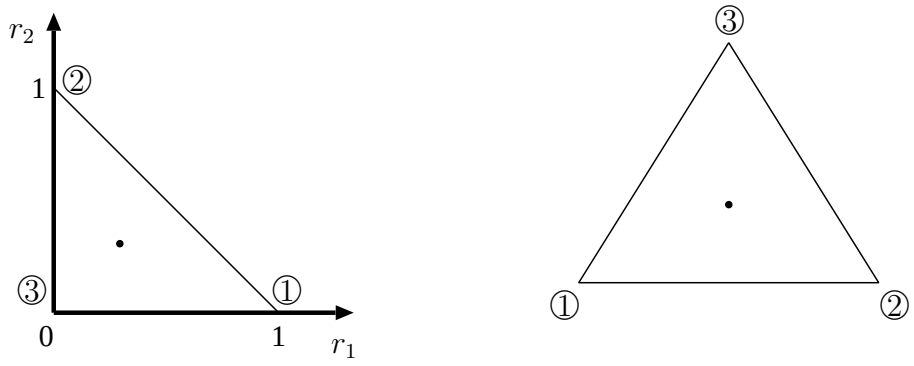


図 4.13: 三角形要素の自然座標と面積座標

である. (漢字判別不可) の $\dfrac{\partial N^{(i)}}{\partial r_1}, \ldots$ は以下のようにして面積座標系と対応させる.

$$\frac{\partial N^{(i)}}{\partial r_1} = \frac{\partial N^{(i)}}{\partial L_1}\frac{\partial L_1}{\partial r_1} + \frac{\partial N^{(i)}}{\partial L_2}\frac{\partial L_2}{\partial r_1} + \frac{\partial N^{(i)}}{\partial L_3}\frac{\partial L_3}{\partial r_1} \tag{4.259}$$

$$= \frac{\partial N^{(i)}}{\partial L_1} - \frac{\partial N^{(i)}}{\partial L_3} \tag{4.260}$$

$$\frac{\partial N^{(i)}}{\partial r_2} = \frac{\partial N^{(i)}}{\partial L_1}\frac{\partial L_1}{\partial r_2} + \frac{\partial N^{(i)}}{\partial L_2}\frac{\partial L_2}{\partial r_2} + \frac{\partial N^{(i)}}{\partial L_3}\frac{\partial L_3}{\partial r_2} \tag{4.261}$$

$$= \frac{\partial N^{(i)}}{\partial L_2} - \frac{\partial N^{(i)}}{\partial L_3} \tag{4.262}$$

これらをもとに $\int_V dV \Rightarrow \int_0^1 \int_0^{1-r_1} \det J \, dr_2 \, dr_1$ を行う. 実際には $\iint F \, dr_1 \, dr_2 = \frac{1}{2}\sum w_i F(x_i, y_i)$ の形式で

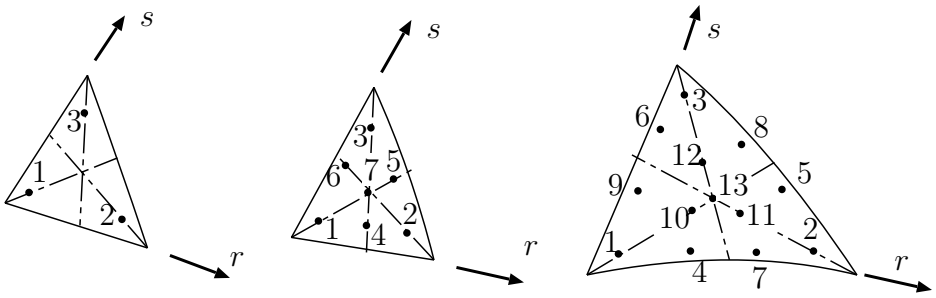


図 4.14: 積分点

Integration order	Degree of precision	r-coordinates	s-coordinates	Weights
3-point	2	$r_1 = 0.16666\ 66666\ 667$	$s_1 = r_1$	$w_1 = 0.33333\ 33333\ 333$
		$r_2 = 0.66666\ 66666\ 667$	$s_2 = r_1$	$w_2 = w_1$
		$r_3 = r_1$	$s_3 = r_2$	$w_3 = w_1$
7-point	5	$r_1 = 0.10128\ 65073\ 235$	$s_1 = r_1$	$w_1 = 0.12593\ 91805\ 448$
		$r_2 = 0.79742\ 69853\ 531$	$s_2 = r_1$	$w_2 = w_1$
		$r_3 = r_1$	$s_3 = r_2$	$w_3 = w_1$
		$r_4 = 0.47014\ 20641\ 051$	$s_4 = r_6$	$w_4 = 0.13239\ 41527\ 885$
		$r_5 = r_4$	$s_5 = r_4$	$w_5 = w_4$
		$r_6 = 0.05971\ 58717\ 898$	$s_6 = r_4$	$w_6 = w_4$
		$r_7 = 0.33333\ 33333\ 333$	$s_7 = r_7$	$w_7 = 0.225$
13-point	7	$r_1 = 0.06513\ 01029\ 002$	$s_1 = r_1$	$w_1 = 0.05334\ 72356\ 008$
		$r_2 = 0.86973\ 97941\ 956$	$s_2 = r_1$	$w_2 = w_1$
		$r_3 = r_1$	$s_3 = r_2$	$w_3 = w_1$
		$r_4 = 0.31286\ 54960\ 049$	$s_4 = r_6$	$w_4 = 0.07711\ 37608\ 903$
		$r_5 = 0.63844\ 41885\ 698$	$s_5 = r_4$	$w_5 = w_4$
		$r_6 = 0.04869\ 03154\ 253$	$s_6 = r_5$	$w_6 = w_4$
		$r_7 = r_5$	$s_7 = r_6$	$w_7 = w_4$
		$r_8 = r_4$	$s_8 = r_5$	$w_8 = w_4$
		$r_9 = r_6$	$s_9 = r_4$	$w_9 = w_4$
		$r_{10} = 0.26034\ 59660\ 790$	$s_{10} = r_{10}$	$w_{10} = 0.17561\ 52574\ 332$
		$r_{11} = 0.47930\ 80678\ 419$	$s_{11} = r_{10}$	$w_{11} = w_{10}$
		$r_{12} = r_{10}$	$s_{12} = r_{11}$	$w_{12} = w_{10}$
		$r_{13} = 0.33333\ 33333\ 333$	$s_{13} = r_{13}$	$w_{13} = -0.14957\ 00444\ 67$

1 次 3 節点

$$N^{(1)} = L_1 \tag{4.263}$$

$$N^{(2)} = L_2 \tag{4.264}$$

$$N^{(3)} = L_3 \tag{4.265}$$

$$\begin{array}{lll} \dfrac{\partial N^{(1)}}{\partial L_1} = 1 & \dfrac{\partial N^{(2)}}{\partial L_1} = 0 & \dfrac{\partial N^{(3)}}{\partial L_1} = 0 \\ \dfrac{\partial N^{(1)}}{\partial L_2} = 0 & \dfrac{\partial N^{(2)}}{\partial L_2} = 1 & \dfrac{\partial N^{(3)}}{\partial L_2} = 0 \\ \dfrac{\partial N^{(1)}}{\partial L_3} = 0 & \dfrac{\partial N^{(2)}}{\partial L_3} = 0 & \dfrac{\partial N^{(3)}}{\partial L_3} = 1 \end{array}$$

2 次 6 節点

$$N^{(1)} = L_1(2L_1 - 1) \tag{4.266}$$

$$N^{(2)} = L_2(2L_2 - 1) \tag{4.267}$$

$$N^{(3)} = L_3(2L_3 - 1) \tag{4.268}$$

$$N^{(4)} = 4L_2L_3 \tag{4.269}$$

$$N^{(5)} = 4L_3L_1 \tag{4.270}$$

$$N^{(6)} = 4L_1L_2 \tag{4.271}$$

$$\begin{array}{llllll} \dfrac{\partial N^{(1)}}{\partial L_1} = 4L_1 - 1 & \dfrac{\partial N^{(2)}}{\partial L_1} = 0 & \dfrac{\partial N^{(3)}}{\partial L_1} = 0 & \dfrac{\partial N^{(4)}}{\partial L_1} = 0 & \dfrac{\partial N^{(5)}}{\partial L_1} = 4L_3 & \dfrac{\partial N^{(6)}}{\partial L_1} = 4L_2 \\ \dfrac{\partial N^{(1)}}{\partial L_2} = 0 & \dfrac{\partial N^{(2)}}{\partial L_2} = 4L_2 - 1 & \dfrac{\partial N^{(3)}}{\partial L_2} = 0 & \dfrac{\partial N^{(4)}}{\partial L_2} = 4L_3 & \dfrac{\partial N^{(5)}}{\partial L_2} = 0 & \dfrac{\partial N^{(6)}}{\partial L_2} = 0 \\ \dfrac{\partial N^{(1)}}{\partial L_3} = 0 & \dfrac{\partial N^{(2)}}{\partial L_3} = 0 & \dfrac{\partial N^{(3)}}{\partial L_3} = 4L_3 - 1 & \dfrac{\partial N^{(4)}}{\partial L_3} = 4L_2 & \dfrac{\partial N^{(5)}}{\partial L_3} = 0 & \dfrac{\partial N^{(6)}}{\partial L_3} = 0 \end{array}$$

6 節点の補間関数は 3 節点から作ることができる.

$$N^{(1)} = L_1 - \frac{1}{2}N^{(5)} - \frac{1}{2}N^{(6)} \tag{4.272}$$

$$= L_1 - \frac{1}{2}\cdot 4L_3L_1 - \frac{1}{2}\cdot 4L_1L_2 \tag{4.273}$$

$$= L_1 - 2L_3L_1 - 2L_1L_2 \tag{4.274}$$

$$= L_1(1 - 2(L_2 + L_3)) \tag{4.275}$$

$$= L_1(2L_1 - 1) \qquad (\because L_1 + L_2 + L_3 = 1) \tag{4.276}$$

以下同様.

これと同じ方法に従って 3 節点, 6 節点に bubble を導入する.

3 節点.

$$N^{(1)} = L_1 \tag{4.277}$$

$$N^{(2)} = L_2 \tag{4.278}$$

$$N^{(3)} = L_3 \tag{4.279}$$

に対し bubble $N^{(4)}$

$$N^{(4)} = 27L_1L_2L_3 \tag{4.280}$$

これより

$$\tilde{N}^{(1)} = N^{(1)} - \frac{1}{3}N^{(4)} \tag{4.281}$$

$$\tilde{N}^{(2)} = N^{(2)} - \frac{1}{3}N^{(4)} \tag{4.282}$$

$$\tilde{N}^{(3)} = N^{(3)} - \frac{1}{3}N^{(4)} \tag{4.283}$$

となる.

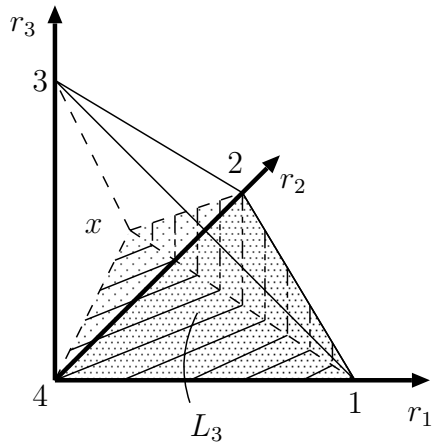


図 4.15: 体積座標系

6 節点.

$$
\begin{aligned}
N^{(1)} &= L_1(2L_1-1) & & \tilde{N^{(1)}} = N^{(1)} + \frac{1}{9}N^{(7)} \\
N^{(2)} &= L_2(2L_2-1) & & \tilde{N^{(2)}} = N^{(2)} + \frac{1}{9}N^{(7)} \\
N^{(3)} &= L_3(2L_3-1) & & \tilde{N^{(3)}} = N^{(3)} + \frac{1}{9}N^{(7)} \\
N^{(4)} &= 4L_2L_3 & \Rightarrow \quad & \tilde{N^{(4)}} = N^{(4)} - \frac{4}{9}N^{(7)} \\
N^{(5)} &= 4L_3L_1 & & \tilde{N^{(5)}} = N^{(5)} - \frac{4}{9}N^{(7)} \\
N^{(6)} &= 4L_1L_2 & & \tilde{N^{(6)}} = N^{(6)} - \frac{4}{9}N^{(7)} \\
N^{(7)} &= 27L_1L_2L_3 \quad \text{(bubble.)}
\end{aligned}
$$

4.6 4 面体要素の補間関数と数値積分

4 面体要素の補間関数は以下のように定義される体積座標系で表される. L_i = 節点 i を含まない三角錐の基準化された体積

したがって

$$L_1 + L_2 + L_3 + L_4 = 1 \tag{4.284}$$

節点 1　(1 0 0 0)

節点 2　(0 1 0 0)

節点 3　(1 0 1 0)

節点 4　(0 0 0 1)

自然座標系 r_1, r_2, r_3 とは, 以下のように対応させる.

$$L_1 = r_1 \tag{4.285}$$

$$L_2 = r_2 \tag{4.286}$$

$$L_3 = r_3 \tag{4.287}$$

$$L_4 = 1 - r_1 - r_2 - r_3 \tag{4.288}$$

物理空間での積分 $\int_V \sim dV$ を自然座標系に変換する.

$$\begin{bmatrix} \dfrac{\partial N^{(i)}}{\partial r_1} \\ \dfrac{\partial N^{(i)}}{\partial r_2} \\ \dfrac{\partial N^{(i)}}{\partial r_3} \end{bmatrix} = \underbrace{\begin{bmatrix} \dfrac{\partial x}{\partial r_1} & \dfrac{\partial y}{\partial r_1} & \dfrac{\partial z}{\partial r_1} \\ \dfrac{\partial x}{\partial r_2} & \dfrac{\partial y}{\partial r_2} & \dfrac{\partial z}{\partial r_2} \\ \dfrac{\partial x}{\partial r_3} & \dfrac{\partial y}{\partial r_3} & \dfrac{\partial z}{\partial r_3} \end{bmatrix}}_{\Downarrow [J]} \begin{bmatrix} \dfrac{\partial N^{(i)}}{\partial x} \\ \dfrac{\partial N^{(i)}}{\partial y} \\ \dfrac{\partial N^{(i)}}{\partial z} \end{bmatrix} \tag{4.289}$$

したがって

$$\begin{bmatrix} \dfrac{\partial N^{(i)}}{\partial x} \\ \dfrac{\partial N^{(i)}}{\partial y} \\ \dfrac{\partial N^{(i)}}{\partial z} \end{bmatrix} = [J^{-1}] \begin{bmatrix} \dfrac{\partial N^{(i)}}{\partial r_1} \\ \dfrac{\partial N^{(i)}}{\partial r_2} \\ \dfrac{\partial N^{(i)}}{\partial r_3} \end{bmatrix} \tag{4.290}$$

それぞれの $\dfrac{\partial x}{\partial r_1}$, $\dfrac{\partial y}{\partial r_1}$, ... はアイソパラメトリック要素なので,

$$\frac{\partial x}{\partial r_1} = \sum \frac{\partial N^{(i)}}{\partial r_1} x_i \ldots \tag{4.291}$$

である. $\dfrac{\partial N^{(i)}}{\partial r_1}$ は, 以下のようにして面積座標系と対応させる.

$$\frac{\partial N^{(i)}}{\partial r_1} = \frac{\partial N^{(i)}}{\partial L_1}\frac{\partial L_1}{\partial r_1} + \frac{\partial N^{(i)}}{\partial L_2}\frac{\partial L_2}{\partial r_1} + \frac{\partial N^{(i)}}{\partial L_3}\frac{\partial L_3}{\partial r_1} + \frac{\partial N^{(i)}}{\partial L_4}\frac{\partial L_4}{\partial r_1} \tag{4.292}$$

$$= \frac{\partial N^{(i)}}{\partial L_1} - \frac{\partial N^{(i)}}{\partial L_4} \tag{4.293}$$

$$\cdots$$

これらをもとに

$$\int_V F(x,y,z)dV = \frac{1}{6}\sum w_i F(x_i,y_i,z_i) \tag{4.294}$$

の形式で数値積分を行う.

積分点, 及び重み

N	M	Weight W	α	β	ν	
2	4	$0.25000\ 00000\ 000000^{+000}$	$0.58541^{01966}249685^{+000}$	$0.13819^{66011}25015^{+000}$	$0.13819^{66011}250105^{+000}$	0.13819^{660}
3	1	$-0.80000^{00000}000000^{+000}$	$0.25000^{00000}000000^{+000}$	$0.25000^{00000}000000^{+000}$	$0.25000^{00000}000000^{+000}$	0.25000^{000}
	4	$0.45000^{00000}000000^{+000}$	$0.50000^{00000}000000^{+000}$	$0.16666^{66666}666667^{+000}$	$0.16666^{66666}666667^{+000}$	0.16666^{666}
4	4	$0.50373\ 79410\ 012282^{-001}$	$0.77164\ 29020\ 672371^{+000}$	$0.76119\ 03264\ 425430^{-001}$	$0.76119\ 03264\ 425430^{-001}$	$0.76119\ 032$
	12	$0.66542\ 06863\ 329239^{-001}$	$0.11970\ 05277\ 978019^{+000}$	$0.71831\ 64526\ 766925^{-001}$	$0.40423\ 39134\ 672644^{+000}$	$0.40423\ 391$
5	1	$0.18841\ 85567\ 365411^{+000}$	$0.25000\ 00000\ 000000^{+000}$	$0.25000\ 00000\ 000000^{+000}$	$0.25000\ 00000\ 000000^{+000}$	$0.25000\ 000$
	4	$0.67038\ 58372\ 604275^{-001}$	$0.73163\ 69079\ 576180^{+000}$	$0.89454\ 36401\ 412733^{-001}$	$0.89454\ 36401\ 412733^{-001}$	$0.89454\ 364$
	12	$0.45285\ 59236\ 327399^{-001}$	$0.13258\ 10999\ 384657^{+000}$	$0.24540\ 03792\ 903000^{-001}$	$0.42143\ 94310\ 662522^{+000}$	$0.42143\ 943$
6	1	$0.90401\ 29046\ 014750^{-001}$	$0.25000\ 00000\ 000000^{+000}$	$0.25000\ 00000\ 000000^{+000}$	$0.25000\ 00000\ 000000^{+000}$	$0.25000\ 000$
	4	$0.19119\ 83427\ 899124^{-001}$	$0.82771\ 92480\ 479295^{+000}$	$0.57426\ 91731\ 735683^{-001}$	$0.57426\ 91731\ 735683^{-001}$	$0.57426\ 917$
	12	$0.43614\ 93840\ 666568^{-001}$	$0.51351\ 88412\ 556341^{-001}$	$0.48605\ 10285\ 706072^{+000}$	$0.23129\ 85436\ 519147^{+000}$	$0.23129\ 854$
	12	$0.25811\ 67596\ 199161^{-001}$	$0.29675\ 38129\ 690260^{+000}$	$0.60810\ 79894\ 015281^{+000}$	$0.47569\ 09881\ 472290^{-001}$	$0.47569\ 098$

4 面体の補間関数

4.6.1 1 4 節点

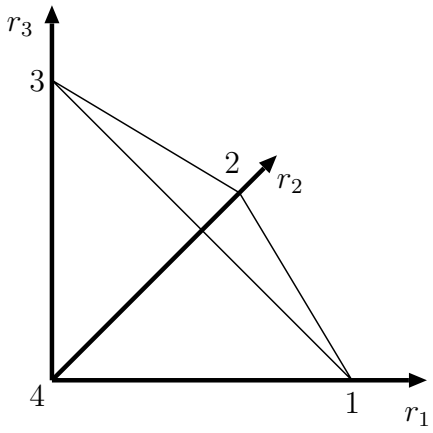


図 4.16: 体積座標系

$$N^{(1)} = L_1 \tag{4.295}$$

$$N^{(2)} = L_2 \tag{4.296}$$

$$N^{(3)} = L_3 \tag{4.297}$$

$$N^{(4)} = L_4 \tag{4.298}$$

$$\frac{\partial N^{(1)}}{\partial L_1} = 1 \qquad \frac{\partial N^{(2)}}{\partial L_1} = 0 \qquad \frac{\partial N^{(3)}}{\partial L_1} = 0 \qquad \frac{\partial N^{(4)}}{\partial L_1} = 0$$

$$\frac{\partial N^{(1)}}{\partial L_2} = 0 \qquad \frac{\partial N^{(2)}}{\partial L_2} = 1 \qquad \frac{\partial N^{(3)}}{\partial L_2} = 0 \qquad \frac{\partial N^{(4)}}{\partial L_2} = 0$$

$$\frac{\partial N^{(1)}}{\partial L_3} = 0 \qquad \frac{\partial N^{(2)}}{\partial L_3} = 0 \qquad \frac{\partial N^{(3)}}{\partial L_3} = 1 \qquad \frac{\partial N^{(4)}}{\partial L_3} = 0$$

$$\frac{\partial N^{(1)}}{\partial L_4} = 0 \qquad \frac{\partial N^{(2)}}{\partial L_4} = 0 \qquad \frac{\partial N^{(3)}}{\partial L_4} = 0 \qquad \frac{\partial N^{(4)}}{\partial L_4} = 1$$

4.6.2 2 4節点 + 1 bubble at center of element

$$N_5 = \mathrm{at}\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right) \tag{4.299}$$

$$N_5 = 256 L_1 L_2 L_3 L_4 \tag{4.300}$$

$\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right)$ で 0 になるように $N^{(1)} \sim N^{(4)}$ を変更する.

$$N^{(1)} = N^{(1)} - \frac{1}{4} N^{(5)} \tag{4.301}$$

$$N^{(2)} = N^{(2)} - \frac{1}{4} N^{(5)} \tag{4.302}$$

$$N^{(3)} = N^{(3)} - \frac{1}{4} N^{(5)} \tag{4.303}$$

$$N^{(4)} = N^{(4)} - \frac{1}{4} N^{(5)} \tag{4.304}$$

$$\frac{\partial N^{(5)}}{\partial L_1} = 256 L_2 L_3 L_4 \tag{4.305}$$

$$\frac{\partial N^{(5)}}{\partial L_2} = 256 L_1 L_3 L_4 \tag{4.306}$$

$$\frac{\partial N^{(5)}}{\partial L_3} = 256 L_1 L_2 L_4 \tag{4.307}$$

$$\frac{\partial N^{(5)}}{\partial L_4} = 256 L_1 L_2 L_3 \tag{4.308}$$

4.6.3 3 10節点 — quadratic

4 節点の各辺上に中間節点を持つ.

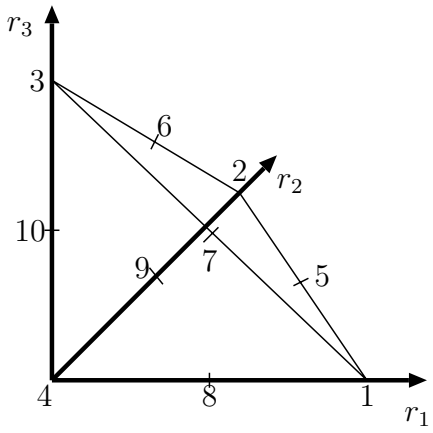


図 4.17: 体積座標系

$1 \leftrightarrow 2$	5
$2 \leftrightarrow 3$	6
$1 \leftrightarrow 3$	7
$1 \leftrightarrow 4$	8
$2 \leftrightarrow 4$	9
$3 \leftrightarrow 4$	10

$$N^{(5)} = 4L_1L_2 \quad \left(\frac{1}{2}, \frac{1}{2}, 0, 0\right) \tag{4.309}$$

$$N^{(6)} = 4L_2L_3 \quad \left(0, \frac{1}{2}, \frac{1}{2}, 0\right) \tag{4.310}$$

$$N^{(7)} = 4L_1L_3 \quad \left(\frac{1}{2}, 0, \frac{1}{2}, 0\right) \tag{4.311}$$

$$N^{(8)} = 4L_1L_4 \quad \left(\frac{1}{2}, 0, 0, \frac{1}{2}\right) \tag{4.312}$$

$$N^{(9)} = 4L_2L_4 \quad \left(0, \frac{1}{2}, 0, \frac{1}{2}\right) \tag{4.313}$$

$$N^{(10)} = 4L_3L_4 \quad \left(0, 0, \frac{1}{2}, \frac{1}{2}\right) \tag{4.314}$$

$$N^{(1)} = N^{(1)} - \frac{1}{2}(N^{(5)} + N^{(7)} + N^{(8)}) \ = L_1(2L_1 - 1) \tag{4.315}$$

$$N^{(2)} = N^{(2)} - \frac{1}{2}(N^{(5)} + N^{(6)} + N^{(9)}) \ = L_2(2L_2 - 1) \tag{4.316}$$

$$N^{(3)} = N^{(3)} - \frac{1}{2}(N^{(6)} + N^{(7)} + N^{(10)}) = L_3(2L_3 - 1) \tag{4.317}$$

$$N^{(4)} = N^{(4)} - \frac{1}{2}(N^{(8)} + N^{(9)} + N^{(10)}) = L_4(2L_4 - 1) \tag{4.318}$$

$$\frac{\partial N^{(1)}}{\partial L_1} = 4L_1 - 1 \tag{4.319}$$

$$\frac{\partial N^{(2)}}{\partial L_2} = 4L_2 - 1 \tag{4.320}$$

$$\frac{\partial N^{(3)}}{\partial L_3} = 4L_3 - 1 \tag{4.321}$$

$$\frac{\partial N^{(4)}}{\partial L_4} = 4L_4 - 1 \tag{4.322}$$

$$\frac{\partial N^{(5)}}{\partial L_1} = 4L_2 \qquad \frac{\partial N^{(5)}}{\partial L_2} = 4L_1$$

$$\frac{\partial N^{(6)}}{\partial L_2} = 4L_3 \qquad \frac{\partial N^{(6)}}{\partial L_3} = 4L_2$$

$$\frac{\partial N^{(6)}}{\partial L_2} = 4L_3 \qquad \frac{\partial N^{(7)}}{\partial L_3} = 4L_1$$

$$\frac{\partial N^{(8)}}{\partial L_1} = 4L_4 \qquad \frac{\partial N^{(8)}}{\partial L_4} = 4L_1$$

$$\frac{\partial N^{(9)}}{\partial L_2} = 4L_4 \qquad \frac{\partial N^{(9)}}{\partial L_4} = 4L_2$$

$$\frac{\partial N^{(10)}}{\partial L_3} = 4L_4 \qquad \frac{\partial N^{(10)}}{\partial L_4} = 4L_3$$

4.6.4　4　10 節点+4 bubble at center of each face and 1 bubble at center of element

10 節点の各面の重心に bubble.

1 の対面	14	$\left(0 \quad \frac{1}{3} \quad \frac{1}{3} \quad \frac{1}{3}\right)$
2 の対面	13	$\left(\frac{1}{3} \quad 0 \quad \frac{1}{3} \quad \frac{1}{3}\right)$
3 の対面	12	$\left(\frac{1}{3} \quad \frac{1}{3} \quad 0 \quad \frac{1}{3}\right)$
4 の対面	11	$\left(\frac{1}{3} \quad \frac{1}{3} \quad \frac{1}{3} \quad 0\right)$

$$N_{11} = 27L_1L_2L_3 \tag{4.323}$$

$$N_{12} = 27L_1L_2L_4 \tag{4.324}$$

$$N_{13} = 27L_1L_3L_4 \tag{4.325}$$

$$N_{14} = 27L_2L_3L_4 \tag{4.326}$$

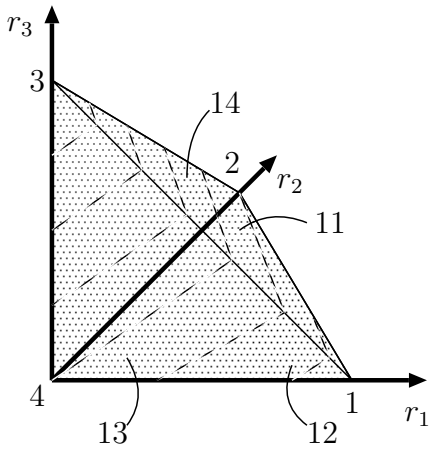


図 4.18: 体積座標系

$N^{(1)} \sim N^{(10)}$ をとりあえず変更する.

$$N^{(1)} = N^{(1)} + \frac{1}{9}(N_{11} + N_{12} + N_{13}) \tag{4.327}$$

$$N^{(2)} = N^{(2)} + \frac{1}{9}(N_{11} + N_{12} + N_{14}) \tag{4.328}$$

$$N^{(3)} = N^{(3)} + \frac{1}{9}(N_{11} + N_{13} + N_{14}) \tag{4.329}$$

$$N^{(4)} = N^{(4)} + \frac{1}{9}(N_{12} + N_{13} + N_{14}) \tag{4.330}$$

$$N^{(5)} = N^{(5)} - \frac{4}{9}(N_{11} + N_{12}) \tag{4.331}$$

$$N^{(6)} = N^{(6)} - \frac{4}{9}(N_{11} + N_{14}) \tag{4.332}$$

$$N^{(7)} = N^{(7)} - \frac{4}{9}(N_{11} + N_{13}) \tag{4.333}$$

$$N^{(8)} = N^{(8)} - \frac{4}{9}(N_{12} + N_{13}) \tag{4.334}$$

$$N^{(9)} = N^{(9)} - \frac{4}{9}(N_{12} + N_{14}) \tag{4.335}$$

$$N^{(10)} = N^{(10)} - \frac{4}{9}(N_{13} + N_{14}) \tag{4.336}$$

要素の重心に bubble
at $\left(\frac{1}{4} \quad \frac{1}{4} \quad \frac{1}{4} \quad \frac{1}{4}\right)$

$$N^{(15)} = 256 L_1 L_2 L_3 L_4 \tag{4.337}$$

$N^{(1)} \sim N^{(14)}$ を変更する

$$N^{(1)} \sim N^{(14)} = N^{(1)} \ \sim N^{(4)} \ - \frac{1}{64} N^{(15)} \tag{4.338}$$

$$N^{(5)} \sim N^{(10)} = N^{(5)} \ \sim N^{(10)} + \frac{1}{8} N^{(15)} \tag{4.339}$$

$$N^{(11)} \sim N^{(14)} = N^{(11)} \sim N^{(14)} - \frac{27}{64} N^{(15)} \tag{4.340}$$