4.2 確率モデルの概要

4.2.1 鎖モデル

このモデルは「最弱連鎖」のモデルとも呼ばれ、確率モデルの中で最も広く用いられているものである。「物体を構成する単位要素のうち最も弱いものが破壊したときにその物体全体の破壊が起こる」との仮定に基づいており、Fig.4.1に示したような直列につながった鎖を構成する環の中で最も弱い環が切断されたときに鎖が使えなくなる現象で説明されるのでこの名で呼ばれる。その考え方から疲労破壊、特に多くの部品で構成される機器寿命のモデル化や脆性材料の破壊のモデル化に適しており、ﾜｲﾌﾞﾙ分布はこの考え方に立脚して考案されたものである[47]。例えば、脆性材料の場合は、材料内にさまざまな大きさの亀裂状の欠陥が空間的に分布していてそのうちの最も弱いものによって強度が決まると考えている。それ以外でも直列系の破壊を起こすもの、例えば繊維の切断などに適用することができ、また、並列系の破壊に対しても構成要素の強度が均一である場合や構成要素が少ない場合には近似的に適用可能である。

また、鎖を構成する要素が全て同じ分布関数に従う場合のみを「鎖モデル」と呼び、それぞれ異なる分布関数に従う場合は「競合リスクモデル」と呼んで区別することもあるが、ここでは区別をしないで全て鎖モデルと呼ぶ。

次に上述した鎖の場合を例にとって鎖モデルに従って破壊する物質の強度の分布関数を求めてみる。

i番目の環が荷重x以下で破壊する事象をE_iとすると、鎖が荷重x以下で破壊する確率、すなわち鎖の強度の確率分布関数$F(x)$は以下のように与えられる。

$$P\{X \le x\} = F(x) = P\{E_1 \cup E_2 \cup \cdots \cup E_n\} \tag{4.1}$$

$X \le x$の排反事象の確率$P\{X > x\}$、すなわち鎖の生存確率（荷重xでまだ破壊していない確率）を考えるとド・モルガンの法則より、(4.1)式より、

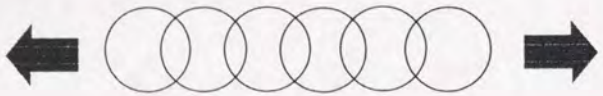

Fig.4.1 Schematic diagram of chain model.

$$P\{X > x\} = 1 - F(x) = P\{E_1^c \cap E_2^c \cap \cdots \cap E_n^c\} \tag{4.2}$$

となる。ここで、E_i^cというのはE_iの排反事象のことである。

一つ一つの環の強度は互いに独立であると考えられるので(4.2)式の右辺は、

$$P\{E_1^c \cap E_2^c \cap \cdots \cap E_n^c\} = P\{E_1^c\}P\{E_2^c\}\cdots P\{E_n^c\} = \prod_{i=1}^{n}\left[P\{E_i^c\}\right] \tag{4.3}$$

となり、よって(4.2)式は、

$$1 - F(x) = \prod_{i=1}^{n}\left[P\{E_i^c\}\right] \tag{4.4}$$

となる。i番目の環の強度x_iが分布関数$F_i(x_i)$に従っているとすると、$F_i(x_i) = P\{E_i\}$だから(4.4)式は、

$$1 - F(x) = \prod_{i=1}^{n}\left[1 - F_i(x)\right] \tag{4.5}$$

すなわち、

$$F(x) = 1 - \prod_{i=1}^{n}\left[1 - F_i(x)\right] \tag{4.6}$$

となり、鎖全体としての強度の分布関数が得られる。

また、すべての環の強度が同じ分布関数$F_0(x)$に従って分布している場合には、鎖全体としての強度の分布関数は、

$$F(x) = 1 - \left[1 - F_0(x)\right]^n \tag{4.7}$$

で与えられる。(4.7)式は、上述した狭義の鎖モデルの分布関数に当たり、(4.6)式が競合リスクモデルの分布関数となる。

また、(4.5)式の両辺にlogをとって変換した形

$$F(x) = 1 - \exp\left[\sum_{i=1}^{n}\log\left\{1 - F_i(x)\right\}\right] \tag{4.8}$$

もよく用いられる。

また、環の強度$F_i(x_i)$がワイブル分布に従う、すなわち

$$F(x) = 1 - \exp\left\{-\left(\frac{x}{\beta_i}\right)^{\alpha_i}\right\} \tag{4.9}$$

とおいて、(4.6)式に代入したときに得られる分布関数

$$F(x) = 1 - \prod_{i=1}^{n}\left[-\exp\left\{-\left(\frac{x}{\beta_i}\right)^{\alpha_i}\right\}\right] \tag{4.10}$$

を多重モードワイブル分布の分布関数と呼ぶことがある[50]。

鎖モデルの確率密度関数$f(x)$は、(4.6)，(4.7)式を微分すれば求められ、i番目の環の強度の密度関数を$f_i(x)$とすれば、

$$f(x)=\sum_{i=1}^{n}\left\{f_i(x)\times\prod_{\substack{j=1\\j\neq i}}^{n}\left(1-F_j(x)\right)\right\} \tag{4.11}$$

もしくは、

$$f(x)=\sum_{i=1}^{n}\left[\frac{f_i(x)}{1-F_i(x)}\times\exp\left\{\sum_{i=1}^{n}\log\left(1-F_i(x)\right)\right\}\right] \tag{4.12}$$

となる。

i番目の環の強度の周辺分布の密度関数 $f_{X_i}(x_i)$は(4.6)式を x_iで偏微分すれば求めることができるが以下のように考えればより簡単に求めることができる。今の場合、$f_{X_i}(x_i)$というのは｛i番目の環が荷重xで破壊し、かつ残りの環が荷重xで破壊しない｝という事象が起こる確率と同値であるから、i番目の環が荷重xで破壊する事象をG_iとすると、$f_{X_i}(x_i)$は、

$$f_{X_i}(x_i)=\mathrm{P}\left\{\mathrm{G_i}\cap\mathrm{E_1^c}\cap\cdots\cap\mathrm{E_{i-1}^c}\cap\mathrm{E_{i+1}^c}\cap\cdots\cap\mathrm{E_n^c}\right\} \tag{4.13}$$

となる。それぞれの環が荷重xもしくはx以下で破壊するという事象は互いに独立であるから(4.13)式は、

$$f_{X_i}(x_i)=\mathrm{P}\left\{\mathrm{G_i}\right\}\mathrm{P}\left\{\mathrm{E_1^c}\right\}\cdots\mathrm{P}\left\{\mathrm{E_{i-1}^c}\right\}\mathrm{P}\left\{\mathrm{E_{i+1}^c}\right\}\cdots\mathrm{P}\left\{\mathrm{E_n^c}\right\} \tag{4.14}$$

となり、これで周辺分布の密度関数が導かれたことになる。

ここで、(4.11)式と(4.14)式を比較してみると鎖の強度の密度関数は、i番目の環の強度の周辺分布の密度関数の和となっている、すなわち、

$$f(x)=\sum_{i=1}^{n}\left\{f_{X_i}(x)\right\} \tag{4.15}$$

であることがわかる。

－　鎖モデルの連鎖則　－

もしFig.4.2に示したように鎖を構成する環がさらに複数の環から構成されているならば、i番目の環の強度X_iが従う分布関数は(4.6)式に従うことになる。したがって、i番目の環が分布関数$F_{ij}(x_{ij})$に従う強度X_{ij}を持つm_i個の小環から構成されている場合には、i番目の鎖の強度の分布関数$F_i(x_i)$は、

$$F_i(x_i)=1-\prod_{j=1}^{m_i}\left\{1-F_{ij}(x_i)\right\} \tag{4.16}$$

となり、鎖全体としての強度の分布関数は(4.16)式から、

$$F(x)=1-\prod_{i=1}^{n}\left[\prod_{j=1}^{m_i}\left\{1-F_{ij}(x)\right\}\right]$$

$$=1-\prod_{i=1}^{N}\left\{1-F_k(x)\right\} \tag{4.17}$$

$$N=\sum m_i$$

$$k=m_i\times(i-1)+j$$

で与えられることになる。すなわち$\sum m_i$個の鎖のうち最も弱いものが破壊したときに全体としての破壊が起こると考えるのと同じことになる。これを逆に考えて、一本の鎖がk個の連（m_i個の一つながりの環）からなると考えると、i番目の連の強度x_i'が分布関数を$F_i'(x_i')$に従うならば、(4.16)式から

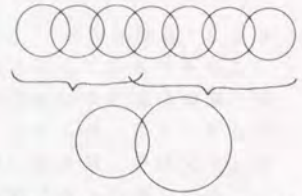

Fig.4.2. Schematic diagram of theory of continuous chain.

$$F_i'(x_i')=1-\prod_{j=1}^{m_i}\left\{1-F_{ij}'(x_i')\right\} \tag{4.18}$$

であり、鎖全体としての強度の分布関数は

$$F(x)=1-\prod_{i=1}^{n}\left\{1-F_i(x)\right\}$$

$$=1-\prod_{i=1}^{k}\left[\prod_{j=1}^{m_j}\left\{1-F_{ij}'(x)\right\}\right]=1-\prod_{i=1}^{k}\left\{1-F_i'(x)\right\} \tag{4.19}$$

となる。以上から、

「鎖モデルに従って破壊する物質では、いくつかの構成要素を一まとめにして一つの構成要素（これを連鎖と呼ぶ）とみなすことができ、全体としての破壊は最も弱い連鎖が破壊したときに起こる。」

と言うことができ、これを鎖モデルの連鎖則と名付ける。これにより、鎖モデルに従う破壊強度の分布関数は個々の構成要素の分布関数がわからなくても連鎖の分布関数がわかっていれば求めることができる。この事実により、鎖モデルを実際の破壊現象に当てはめる際に実験から得られたデータから推定した母集団分布関数をその破壊機構にとらわれずにそのまま用いることができることになり、鎖モデルの適用範囲は著しく増大することになる。

4.2.2 並列モデル

このモデルは、上述した鎖モデルが一番弱い要素が破壊した時に全体の破壊が起こると仮定した直列系の破壊のモデルであるのに対して、Fig.4.3に示したように「すべての要素が破壊したときに破壊が起こる、すなわち、最も強い要素が破壊したときに破壊が起こる」と仮定したものである。その考え方から並列系や待機系の機器の寿命破壊のモデルとして用いられるが[47]、破壊現象には適用しにくいがしかし、亀裂伝播を起こすときの先端部分の破壊等に用いることが可能なのではないかと思われる。後述する「縄モデル」はこのモデルの特殊な場合であるが、ここでは別のものとして扱う。

鎖モデルの時と同様にその強度の分布関数を求めてみる。i番目の要素が荷重x以下で破壊する事象をE_iとすると、全体の強度の確率分布関数$F(x)$は以下のように与えられる。

$$P\{X \le x\} = F(x) = P\{E_1 \cap E_2 \cap \cdots \cap E_n\} \tag{4.20}$$

各事象は互いに独立であるから、

$$F(x) = \prod_{i=1}^{n} P\{E_i\} \tag{4.21}$$

i番目の要素の強度x_iが分布関数$F_i(x_i)$に従っているとすると、

$$F(x) = \prod_{i=1}^{n} F_i(x) \tag{4.22}$$

または、

$$F(x) = \exp\left[\sum_{i=1}^{n} \log\{F_i(x)\}\right] \tag{4.23}$$

となり、全体としての強度の分布関数は、(4.22)，(4.23)式で与えられることになる。

また、すべての要素の強度が同じ分布関数$F_0(x)$に従って分布している場合には、

$$F(x) = F_0(x)^n \tag{4.24}$$

で与えられる。

全体としての強度の密度関数$f(x)$は(4.22)，(4.23)式を微分すれば求められ、

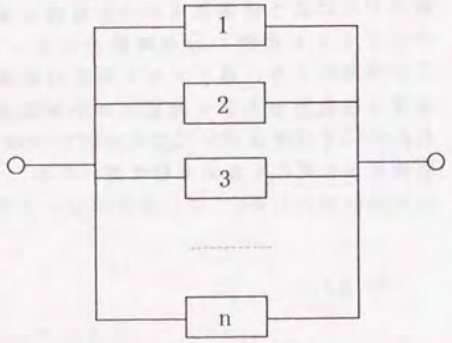

Fig.4.3. Schematic diagram of parallel model.

$$f(x) = \sum_{i=1}^{n}\left\{f_i(x) \times \prod_{\substack{j=1 \\ j \ne i}}^{n} F_j(x)\right\} \tag{4.25}$$

もしくは、

$$f(x) = \sum_{i=1}^{n}\left(\frac{f_i(x)}{F_i(x)}\right) \times \exp\left\{\sum_{i=1}^{n} \log(F_i(x))\right\} \tag{4.26}$$

となる。ただし、$f_i(x)$はi番目の要素の強度の密度関数である。

並列モデルでも鎖モデルの時と同様に連鎖則が成り立つ。

$$F_i'(x_i') = \prod_{j=1}^{m_i}\left\{F_{ij}'(x_i')\right\} \tag{4.27}$$

であり、全体としての強度の分布関数は

$$\begin{aligned} F(x) &= \prod_{i=1}^{n}\{F_i(x)\} \\ &= \prod_{i=1}^{n}\left\{\prod_{j=1}^{m_i}\{F_{ij}'(x)\}\right\} = \prod_{i=1}^{n}\{F_i(x)\} \end{aligned} \tag{4.28}$$

となる。

4.2.3 繩モデル

このモデルは束モデルとも呼ばれ、荷重を加えていくと最初の要素がまず破壊して荷重の再配分が起き、さらに荷重を増していくと２番目に弱い要素が破壊して荷重のまた再配分が起こり、という過程が繰り返されて行き、残存要素の数が次第に減少していくので全体として負担できる荷重は途中で最大となり、以後減少していく、という現象をモデル化したものであり、Fig.4.4に示したような複数の糸から構成される繩に荷重を加えた場合に繩が切れる現象で説明されるのでこの名で呼ばれる。各要素の破壊が独立に起こらないためにその分布関数・密度関数を求めることができず、求められたとしても非常に複雑な形をとるので、モンテカルロシミュレーションによってその結果を見ることが多い。構成される要素数が少ない場合には鎖モデルで近似的に現すことができ、また要素数が多くなると結果として得られる確率分布は正規分布に近づくことが理論的に示されている[46]。

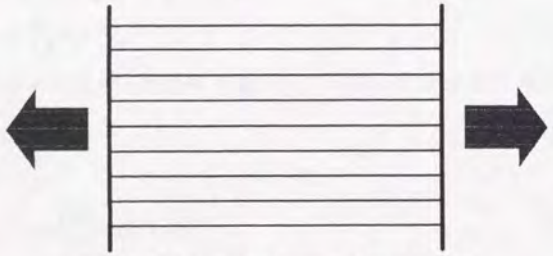

Fig.4.4 Schematic diagram of lope model.

4.2.4 モンテカルロシミュレーションと乱数の発生方法

4.2.4.1 モンテカルロシミュレーション

確率モデルによるモデル化を行った場合、上述してきたような式の形で表せる場合は少なく、たとえ表せたとしても非常に難解な形となってモデル化をしても結果がわからないというような事態に陥ることが良くある。そのような場合に用いられるのがモンテカルロシミュレーション法であり、微分方程式が解けないときに使うオイラー法、ルンゲ・クッタ法や、解けない積分の値を求めるのに使う数値積分と同じ様な使い方をされるものと考えれば良い。その定義は、「統計学の標本調査の技術を応用し、ランダムな変動を入れながら、多数回試行を反復して計算して、問題を近似的に解く帰納的な方法」[46]とされるが、今日では乱数を用いた統計の実証的実験は全てモンテカルロ法による実験、すなわちモンテカルロシミュレーションと呼ばれる。従って、モンテカルロシミュレーションを行うに当たっては、優れた乱数を用いることが大切になりそのためにいろいろな研究が行われている。以下では、この論文で用いた乱数の発生方法について簡単に述べる[62]。

4.2.4.2 一様乱数の発生方法

本論文で行っているモンテカルロシミュレーションで用いた一様乱数は、M系列乱数というものである。M系列というのは、最大周期列（Maximum-length linearly recurring sequence）の略で、疑似ランダム系列（真にランダムな系列の持ついろいろな性質の内のいくつかを近似的に満たすもの）の１つであり、ガロア体GF(2)の原始多項式 $f(x)$ の係数 $(1, C_1, C_2, \ldots, C_p)$ を用いたp次の線形漸化式

$$a_t = C_1 a_{t-1} + C_2 a_{t-2} + \cdots + C_p a_{t-p} \pmod 2$$

によって生成されるGF(2)上の数列$<a_t>$のことをいう。

M系列自体は、0と1からなる系列であり、$<a_t>$の位相を適当にずらしたものを各ビットに配置してつくられたnビットの２進数の系列$<x_t>$が一様乱数となる。

4.2.4.3 正規乱数の発生法

正規乱数の発生法はいろいろ考案されているのであるが、ここでは、本論文で用いた極座標法について述べる。

$$X_1 = \sqrt{-2\log U_1}\cos(2\pi U_1)$$

$$X_2 = \sqrt{-2\log U_2}\sin(2\pi U_2)$$

U_1, U_2：一様乱数

とすると、X_1とX_2は互いに独立で標準正規分布N(0,1)に従う乱数となる。

この方法は乱数発生速度が比較的遅いので、高速化するためにcos, sinの計算をしなくてすむように改良したものが次の方法で、本論文でもこの方法を用いている。そのフローは以下のようになる。

1. U_1, U_2を発生し、$V_1 = 2U_1 - 1, V_2 = 2U_2 - 1$とする。
2. $S = V_1^2 + V_2^2$
3. $S \geq 1$ なら1.にもどる。
4. $X_1 = V_1\sqrt{-2\log S} / S$
   $X_2 = V_2\sqrt{-2\log S} / S$

また、平均μ分散σ^2の正規乱数(Y)は、標準正規乱数(X)から、

$$Y = \mu + X \times \sigma^2$$

として生成できる。

また、平均μ_1, μ_2、分散がσ_1^2, σ_2^2、相関係数がρの2次元正規分布に従う乱数Y_1, Y_2は、標準正規分布に従う独立な乱数X_1, X_2を用いて、

$$Y_1 = \mu_1 + \sigma_1 X_1$$
$$Y_2 = \mu_2 + \sigma_2\left(\rho X_1 + \sqrt{1-\rho^2} X_2\right)$$

によって発生させることができる。

4.2.4.4 ワイブル乱数（2ﾊﾟﾗﾒｰﾀ）

ワイブル分布のように分布関数が閉じた形で表せるものの乱数は逆関数法によって簡単に発生させることができる。逆関数法によれば、ある分布関数$F(x)$の逆関数を$F^{-1}(x)$で定義すると、$F(x)$に従う乱数Xは、

$$X = F^{-1}(U)$$

から得ることができる。2ﾊﾟﾗﾒｰﾀのワイブル分布の場合、分布関数は

$$F(x) = 1 - \exp\left\{-\left(\frac{x}{\beta}\right)^{\alpha}\right\}$$

で与えられるから、

$$X = \beta\{-\log U\}^{1/\alpha}$$

によってワイブル分布に従う乱数を得ることができる。

4.3 3プライ単板積層材引張り破壊への確率モデルの適用

この節では、3プライの単板積層材の引張り破壊を取り上げ、その破壊のモデル化とそれによる強度予測を行い、実測値との比較、検討をおこなった。

4.3.1 破壊のモデル化

3プライの単板積層材の破壊形態は、エレメントそのものの破壊と、ジョイント部近くの接着層・隣接する被着材の破壊の2種類に分類することができる。どちらの破壊がどの程度の応力で生じるのかは、エレメントの材質、接着剤の物性、積層方法等によって異なるが、いずれにしても全く確率的な現象である。このように一つのシステムがいくつかの破壊要因によって破壊する場合のモデル化には、鎖モデルの一種である競合リスクモデルを用いることができる。

4.3.2 競合リスクモデル

競合リスクモデルは、「破壊因子が複数あり、各因子に依存する破壊が互いに独立に生じる」という直列系の事象をモデル化したものである[47]。競合リスクモデルでは、物体の強度をXとすると、確率変数Xの分布関数$F(x)$は、

$$F(x) = 1 - \prod_{i=1}^{n}\{1 - F_i(x)\} \quad (4.29)$$

で与えられる。ここで$F_i(x)$というのは「統計的」に互いに独立な各破壊因子の確率分布関数であり、一つの破壊因子は複数の異なる破壊因子の集合でも構わない。競合リスク分布をワイブル確率紙上に示すと、Fig.4.5に示したような下に凸の折れ線になる。

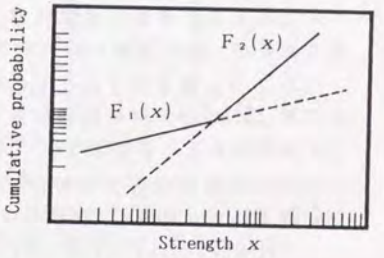

Fig.4.5. Schematic diagram of competing risk distribution on Weibull probability paper.

4.3.3 モデル化と強度分布関数の定義

Fig.4.6に示したような3プライの単板積層材に引張力を作用させた時の破壊を競合リスクモデルによってモデル化する。この時の試験体の破壊形態は、

1）接着層、及び隣接した被着材部分の破壊
（Fig.4.6の斜線部分。以降、接着部破壊と表記）

2）上記以外の単板部分の破壊（引張り破壊と表記）

の２つに大きく分けることができる。

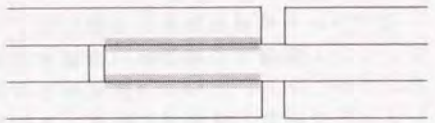

Fig.4.6. Schematic diagram of parts of failure of 3-ply LVL.

接着部破壊は接着層端部に作用するせん断応力、垂直応力のいずれかにより接着層、被着材、接着界面のいずれかで起こると考えられ[63,64]、その結果、様々な破壊形態をとる。しかし、破壊形態を分類して得られるそれぞれの強度分布は、接着部の強度分布に対する周辺分布であって母集団分布ではないので、そのままでは確率モデルに用いることはできない。そこで、各種破壊形態の集合である接着部破壊を統計上の一つの破壊因子として扱い、上記単板積層材の引張り試験から得られた強度分布がその母集団分布であるとする。単板の破壊形態も同様にいくつかに分類することが可能であるが、同じ理由から一まとめにして一つの破壊因子として扱う。

この２つの破壊因子はともに構成単板の物性の影響を受けるのでその強度にはある程度の相関があると思われるが、ここでは、互いに独立な因子であると仮定する。

この２つの破壊因子に対して式（4.29）を適用すると、接着層の重なり長さ（ラップ長）がLの時の試験体の強度の分布関数$F_L(x)$は、

$$F_L(x)=1-\left\{1-F_{A_L}(x)\right\}\left\{1-F_T(x)\right\} \qquad (4.30)$$

$F_{A_L}(x)$：ラップ長Lの時の接着部の強度（A_L）の分布関数

$F_T(x)$：引張り強度（T）の分布関数

で与えられる。

4.3.4 分布関数の推定実験の概要

前節で定めた分布関数$F_{A_L}(x)$、$F_T(x)$を実際に求めるための実験を行なった。この節ではその概要を述べる。

Fig4.7 (1)に示した二重重ね継手状の試験体（TYPE I）、およびFig4.7 (2)に示した形状の試験体（TYPE II）を作製して引張り試験を行った。単純重ね継手の試験体を用いなかった理由は、後者ではモーメントが作用して純粋なせん断強度を得ることが困難であるということと、対

象としている単板積層材の形状が前者と同じものであり、母集団分布を得るためには同じ形状のものから得られた値の方が適切であると考えられるからである。

使用した単板は厚さ2.8mmのベイマツのロータリー単板で、その物性、及び接着剤圧締条件などはTable4.1、4.2に示した通りである。単板のヤング係数は単板の表面にひずみゲージを貼って引張り試験を行って測定した結果から得られた値で、また、単板は直径1cm程度の節を含んでいたが、特にこれを除去しないで使用した。製造方法は、まず、幅40cm（長さは試験体の重なり長さに依存）のシートを作製し、養生後、幅12.5mmに切り揃えて実験を行なった。

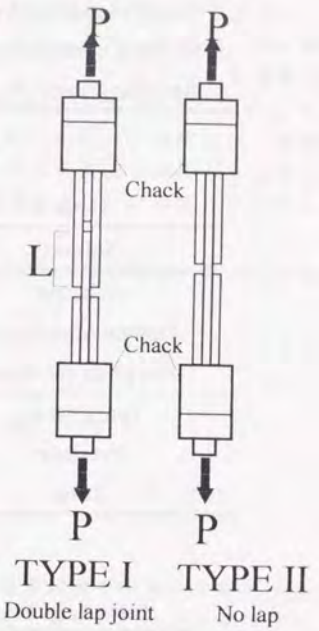

Fig.4.7. Schematic diagram of test specimens.
Notes : L: Overlap length (5, 10, 15, 20, 25, 30, 40, 50, 60mm), P: Load(kgf).

TYPE Iは、破壊強度の母集団分布を得ること、及びモデルに従って得られた強度と比較するための実験値を得ることを目的にしたものであり、そのためにラップ長を変えたものを複数作成した。その寸法はTable4.3に示した通りである。

TYPE IIは、引張り破壊強度の母集団分布を得ることを目的としたもので、このような形状にすることによって想定した単板積層材と同様の単板部破壊が起こり、正しい母集団分布が得られると考えた。

実験に当たっては、万能材料試験機（ミネベア製 TCM5000S）を用い、標点距離10cm、クロスヘッドスピード5mm/min.で行った。

Table 4.1. Properties of lamina.

Subjects	Properties
Young's modulus(Avg.)	11.0×10^4kgf/cm^2
Moisture content(Avg.)	11.9(%)
Specific gravity(Avg.)	0.43

Table 4.2. Gluing condition.

Subject	Condition
Adhesive	Resorcinol resin
Amount of Adhesive	300g/m^2(one side)
Hot press condition	
Temperature	80°C
Pressure	10kg/cm^2
Time	9 minutes

Table 4.3. Dimension of test species.

Width(mm)	12.5
Thickness(mm)	8.4
Overlap length(mm)	Samples
2.5	18
5.0	74
10.0	79
15.0	81
20.0	84
25.0	77
30.0	61
40.0	63
50.0	66
60.0	61

4.3.5 TYPE I 引張り試験

4.3.5.1 実験結果

Table4.4、Fig.4.8に実験結果を示した。文献[65,66]などで報告されているように、ラップ長が長くなるほど見かけのせん断強度（最大荷重／（接着面積×２））は小さくなっていることがわかる。単純重ね継手の試験体では、接着長さが長くなるとその見かけのせん断強度$\tau=P/A$が減少することはよく知られている。その理由は、接着面に均一に応力が作用しておらず端部に大きな応力が作用しているためと考えられ、その応力集中を理論的に求めようという試みはかなり以前からなされている。

Szepe[67]は、重ね部分に応力が作用していない場合の応力集中係数として、

$$\eta=1+\frac{GL^2}{6Edt} \qquad (4.31)$$

τ：　応力集中係数（$=\tau_0/\tau_{avg}$）
G：　接着剤のせん断剛性率
E：　被着材のヤング係数
d：　接着層の厚さ
L：　接着長さ
t：　被着材の厚さ

という式を与えている。これは、二重重ね継手試験体の解としても知られている。また、Volkersen[68]は、継手部にモーメントが作用している場合の近似解として

$$\eta=\sqrt{\frac{GL^2}{2Edt}}\coth\left(\frac{GL^2}{2Edt}\right) \qquad (4.32)$$

という式を与えている。

そのほかにもGolandとReissnerの解[69]などがあるがいずれも決定的なものではなく[70]、現在でも有限要素法による応力解析が行われている[64,65]。

そこで本論文では、任意のラップ長における接着部強度の分布関数$F_{A_L}(x)$を求めるために、引張り破壊の割合が小さいL=5～20mmの試験体の接着部破壊強度分布から、確率モデルを用いて$F_{A_L}(x)$の推定を行なうことにした。ここでA_L, xはそれぞれ、試験体幅1cm当たりの最大荷重（P_{max}/w(=1.25mm)）、荷重（P/w）であるとし、これ以後強度という表現はA_L（$=P_{max}/w$）を指すものとする。

Table 4.4. Results of tensile test for TYPE I .

L (mm)	Samples	P_{max}/w Mean (kgf/cm)	P_{max}/w COV (%)	Ratio of adhesive region fracture (%)	τ_{avg} Mean (kgf/cm^2)
2.5	18	70.5	10.2	100.0	140.5
5	74	109.7	12.8	100.0	105.1
10	79	137.1	12.7	100.0	69.0
15	81	170.0	11.1	97.5	58.1
20	84	169.8	12.6	90.5	43.2
25	77	190.5	15.3	87.0	39.3
30	61	218.8	12.0	77.0	37.2
40	63	236.2	22.0	39.7	31.7
50	66	244.3	17.6	18.2	26.2
60	61	250.0	19.1	18.0	23.8

Notes: L: Overlap length, P_{max}: Maximum load(kgf), w: Width of specimen(cm),
τ_{avg}: Shear strength ($=P_{max}/(2\times w\times L)$)

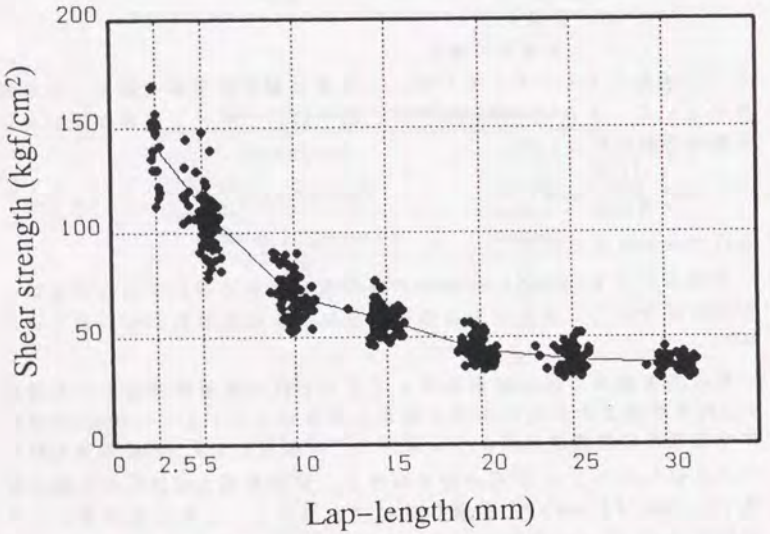

Fig.4.8. Relationship between lap-length and shear strength.

4.3.5.2 確率モデルによる $F_{A_L}(x)$ の推定

TYPE Iのような二重重ね継手状の試験体に引張荷重を作用させた場合、被着材に作用する内部モーメントと、被着材の変形のずれの影響によって、接着層及びその上下の被着材には垂直・せん断応力が不均一に作用する。その大きさは端部付近で最大になり、接着部の破壊を引き起こすが、その応力集中度がラップ長の影響を受けて変動するためにラップ長が長くなるほど接着部の見かけのせん断強度が小さくなる[63]。すなわち、ラップ長Lの時の接着部の強度A_Lは、ラップ長に依存しない試験体固有の接着部強度A_0と、Lによって変化する作用応力の大きさによって決まるので、A_0とLの関数

$$A_L = g(A_0, L) \tag{4.33}$$

の形で表される確率変数であると考えられる。ここで、A_0が形状母数α_0、尺度母数β_0の2母数ワイブル確率変数であるとすると、その分布関数は、確率変数A_0がx以下になる確率は$P\{A_0 \le x\}$であるから、

$$F_{A_0}(x) = 1 - \exp\left\{-\left(\frac{x}{\beta_0}\right)^{\alpha_0}\right\} \tag{4.34}$$

で与えられる。gの逆関数g^{-1}を用いれば、式(4.33)、(4.34)からA_Lの分布関数$F_{A_L}(x)$を、以下のように導き出すことができる。$F_{A_L}(x)$は確率変数A_Lがx以下になる確率であるから、

$$\begin{aligned} F_{A_L}(x) &= P\{A_L \le x\} \\ &= P\{g(A_0, L) \le x\} \\ &= P\{A_0 \le g^{-1}(x, L)\} = 1 - \exp\left\{-\left(\frac{g^{-1}(x,L)}{\beta_0}\right)^{\alpha_0}\right\} \end{aligned} \tag{4.35}$$

さらに式を簡略化するために、関数gがLの関数hを使って$g(A,L) = A \times h(L)$という形で表せるとすれば、式(4.35)は、

$$\begin{aligned} F_{A_L}(x) &= 1 - \exp\left\{-\left(\frac{g^{-1}(x,L)}{\beta_0}\right)^{\alpha_0}\right\} \\ &\equiv 1 - \exp\left\{-\left(\frac{x}{\beta_L}\right)^{\alpha_0}\right\} \qquad \left(\beta_L = \frac{\beta_0}{h^{-1}(L)}\right) \end{aligned} \tag{4.36}$$

となり、A_Lは形状母数α_0、尺度母数β_Lの2母数ワイブル確率変数になる。

式(4.36)からわかるように、α_0はLによらず変化しない。Table4.5に、L= 5mm～20mmの強度分布にミーンランク法[71]を用いて2母数のワイブル分布をあてはめて母数を推定した結果を示した。形状母数の値を

Table 4.5. Estimated parameter of Weibull distribution for the tensile strength of TYPE I specimens.

L (mm)	α_0	β_0
5	9.17	115.6
10	9.15	144.5
15	11.2	179.5
20	9.97	206.8

Notes : L : Overlap length of double lap joint, α_0: Shape parameter, β_0: Scale parameters.

見ると、L=15mmの時の値はやや他と異なるものの、それ以外はほぼ同じ値をとり、式(4.36)と矛盾していないことがわかる。従って、実験結果からLとβ_Lの関係及びα_0の値を推定して、式(4.36)に代入することで、$F_{A_L}(x)$を求めることができる。

a) β_Lの推定

Lとβ_Lの関係を理論的に求めることは困難であるので、Lとβ_Lとは直線関係にあると仮定してLを横軸に、β_Lを縦軸にとってプロットしてみたものがFig.4.9である。これを見るとLとβ_Lとはほぼ直線関係にあるとみて良いと思われる。よって、β_Lは最小二乗法による回帰を行った結果、

$$\beta_L = 4.72L + 96.5 \tag{4.37}$$

で与えられるとした。

b) α_0の推定

式(4.36)で、

$$X = \frac{A_L}{\beta_L} \tag{4.38}$$

とおくと、確率変数Xの分布関数$F_X(x)$は、

$$\begin{aligned} F_X(x) &= \mathrm{P}\left\{\frac{A_L}{\beta_L} \le x\right\} \\ &= \mathrm{P}\{A_L \le x \times \beta_L\} \\ &= F_{A_L}(x \times \beta_L) = 1 - \exp\left\{-x^{\alpha_0}\right\} \end{aligned} \tag{4.39}$$

となり、尺度母数1、形状母数α_0のワイブル分布になる。すなわち、式(4.39)を使ってデータを標準化した後、2母数のワイブル分布を当てはめることによってα_0を推定することができる。Fig.4.10に標準化し

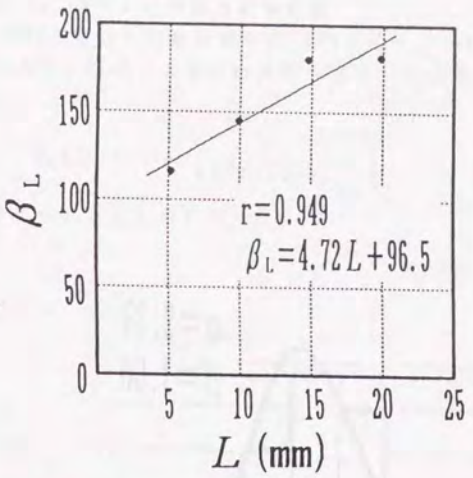

Fig.4. 9. Relationships between L and β_L.

Notes : L: Overlap length(mm), β_L: Scale parameter when overlap length is L, r: correlation efficient.

たデータに対して母数推定を行った結果を示す。この結果得られた形状母数の推定値はα_0= 9.39であった。

以上より、ラップ長Lの時の接着部強度の分布関数は、

$$F_{A_L}(x) = 1 - \exp\left\{-\left(\frac{x}{4.72L + 96.5}\right)^{9.39}\right\} \tag{4.40}$$

で与えられる。

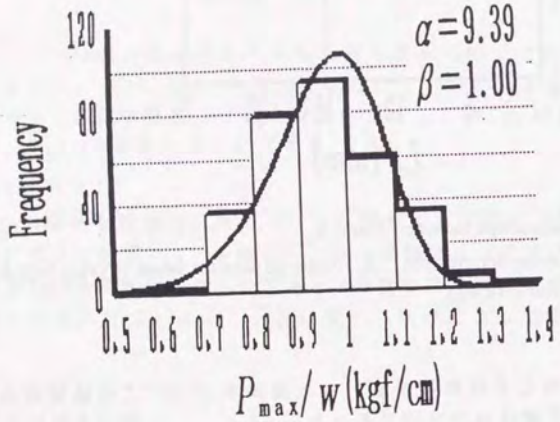

Fig.4. 10. Histogram and Weibull probability density function with two parameters for P_{max} / w values of TYPE I.

Notes : P_{max}: Maximum load(kgf), w: Width of specimen(cm), α : Shape parameter, β : Scale parameter.

4.3.6 TYPEⅡの引張り試験結果

TYPEⅡ引張り試験結果をFig.4.11に示す。2母数のワイブル分布をあてはめて、引張り強度の母集団を推定した結果、確率変数Tの分布関数は、

$$F_t(x)=1-\exp\left\{-\left(\frac{x}{4.72L+96.5}\right)^{9.39}\right\} \qquad (4.41)$$

で与えられることがわかった。

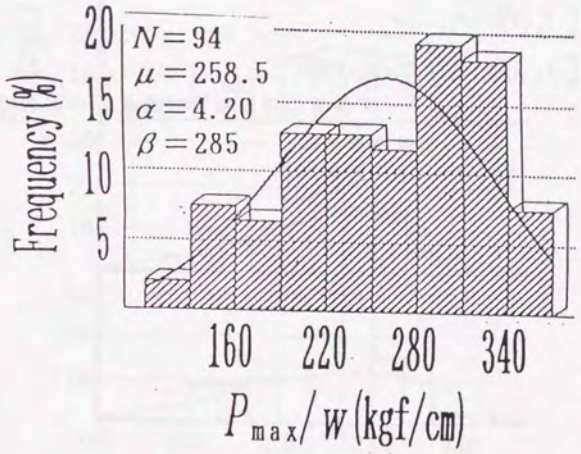

Fig.4. 11. Histogram and Weibull probability density function with two parameters for P_{max} / w values of TYPE II.

Notes : P_{max}: Maximum load(kgf), w: Width of specimen(cm), N: Data number, μ : Mean(kgf/cm), α : Shape parameter, β : Scale parameter.

4.3.7 実験結果と計算結果の比較

前二節の推定で得られた分布関数を用いて、TYPE Iの試験体の引張り試験による破壊強度の予測を行い、実際の結果と比較検討を行った。

式(4.30)、(4.40),(4.41)から、TYPE Iの接着積層材が幅1cmあたりx以下の荷重で破壊する確率は、

$$F_L(x)=1-\exp\left\{-\left(\frac{x}{4.72L+96.5}\right)^{9.39}-\left(\frac{x}{285.0}\right)^{4.2}\right\} \qquad (4.42)$$

で与えられる。式(4.42)から得られる分布曲線と、実験結果とを重ねたものをFig.4.12に示す。また、Fig.4.13、Table4.6に、平均値、変動係数、及び接着部破壊の割合をそれぞれ実験によって得られた値と併せて示した。なお、平均値、分散、接着部破壊の割合の計算値はそれぞれ以下の式から求めることができる。

$$E(x)=\int_0^\infty x f(x)dx \qquad (4.43)$$

$$\mathrm{var}(x)=E\left[\left(X-E(x)^2\right)\right] \qquad (4.44)$$

$$R_L=\int_0^\infty f_{A_L}'(x)\left(1-F_L(x)\right)dx = \int_0^\infty f_{A_L}(x)\left(1-F_T(x)\right)\left(1-F_L(x)\right)dx \qquad (4.45)$$

$f_{A_L}'(x)$ ：ラップ長 L の時の接着部強度( )の周辺分布の密度関数

$f_{A_L}(x)$ ： A_L の密度関数

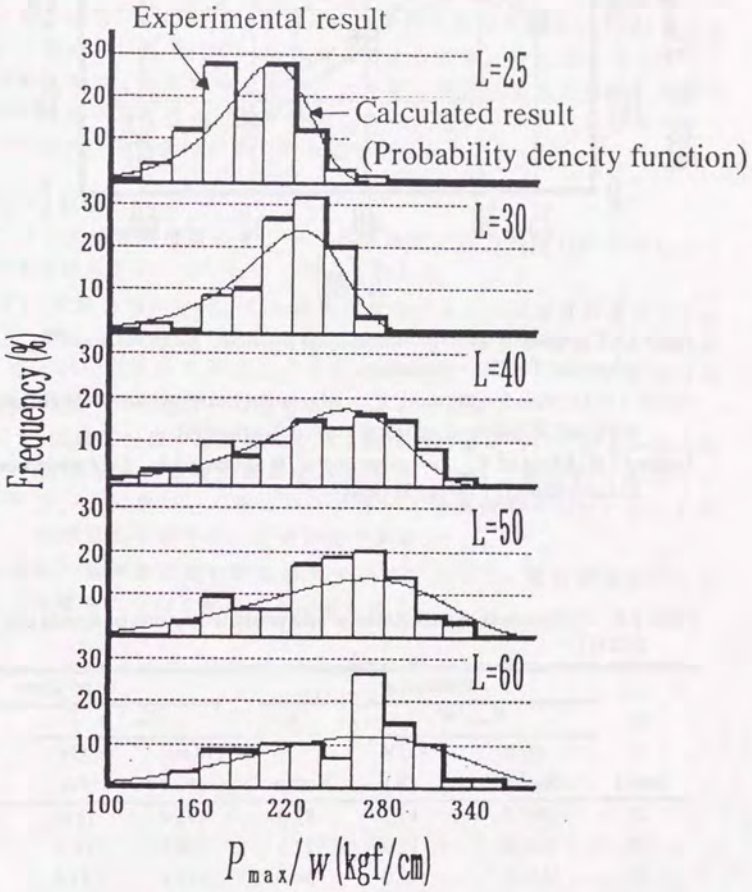

Fig.4.12. Histograms of P_{max} / w values and predicted probability density function for TYPE I specimens.

Legend : P_{max}: Maximum load(kgf), w: Width of specimen(cm), L: Overlap length(mm).

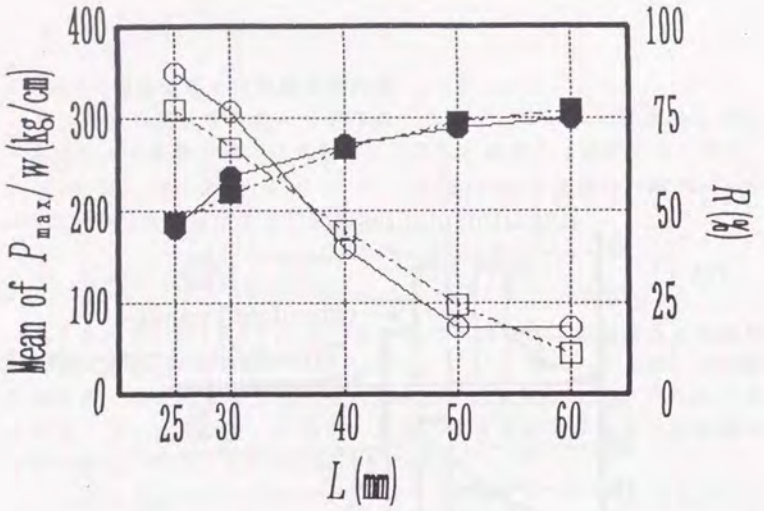

Fig.4.13. Comparison of experimental and predicted mean values of P_{max} / w and values for TYPE I specimens.

Notes : L: Overlap length(mm), P_{max}: Maximum load(kgf), w: Width of specimen(cm), R: Ratio of adhesive region fracture(%).

Legend : ●: Mean of P_{max} / w (experiment), ■: Mean of P_{max} / w (prediction), ○: R(experiment),□; R(prediction).

Table 4.6. Comparison of experimental and predicted results of tensile test for TYPE I.

L	Estimation			Experiment		
	P_{max} / w		R	P_{max} / w		R
	Mean	COV		Mean	COV	
(mm)	(kgf/cm)	(%)	(%)	(kgf/cm)	(%)	(%)
25	190.5	15.3	87.0	193.4	16.6	77.1
30	218.8	12.0	77.1	209.2	18.2	67.2
40	236.2	22.0	39.7	233.4	21.3	44.6
50	244.3	17.6	18.2	247.7	23.8	24.4
60	250.0	19.1	18.0	254.5	25.5	11.2

Notes : L: Overlap length of double lap joint, P_{max}: Maximum tensile load, w: Width of specimen, COV: Coefficient of variation, R: Ratio of adhesive region fracture.

4.3.8 考察

Table 4.6を見ると、変動係数はLが小さい試験体では比較的よく一致しているが、大きくなると予測値の方が実測値よりも大きくなっている傾向がうかがえる。これは、破壊因子の分布関数として2母数ワイブル分布を用いたために、推定によって得られた分布関数$F_L(x)$が低強度側に裾の長い形状になったためと考えられる。しかし、平均値と接着部破壊を起こす試験体の割合、および、現実に破壊が起きた範囲の強度の頻度分布形状は、実験結果とよく一致しており、競合リスクモデルによる強度予測が有効であることが示された。

4.3.9 結論

3プライの単板積層材でラップ長が限定された範囲内にあるという条件下ではあるが、以下のことが示された。

1) 実験結果から得られた接着層重なり長さと接着部強度分布との関係を用いて、重なり長さがLの時の接着部強度の分布関数$F_L(x)$を推定することにより、競合リスクモデルによる接着積層材の強度予測が可能であった。

2) 比較的少数の破壊因子を持つ物体の破壊についてはその個々の破壊因子の破壊機構が明らかでなくても、統計的に処理したデータに基づいて競合リスクモデルを適用することにより、その強度を予測することが可能である。

今後は、積層数と接合部を増やすことによって、更に破壊因子が多くなった場合について検討する必要がある。

4.4 5プライ、9プライ単板積層材引張り破壊への確率モデルの適用

この節では、プライ数を増やした単板積層材に二つの確率モデルを適用し、その適合性について検討を行った。

4.4.1 鎖モデルによるモデル化

Fig.4.14,15に示したような5プライ、9プライの単板積層材の引張り破壊を鎖モデルを用いてモデル化するために、以下のような仮定を行なった。

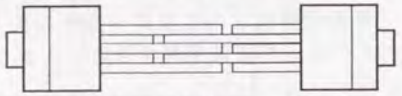

Fig.4.14. Schematic diagram of 5-ply LVL.

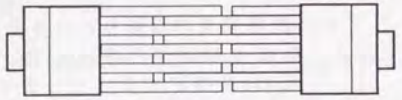

Fig.4.15. Schematic diagram of 9-ply LVL.

1） 破壊は、接着層とそれ以外の位置で起こるものとし、その破壊源のうちのどれか1つが破壊した時点で、全体としての破壊が起こるものとする。

2） 各破壊源の強度は互いに独立である（相関がない）ものとする。

3） プライ数が増えると材の剛性に変化が生じるため前節で求めた3プライの単板積層材に対する、接着長さと分布関数の変化の関係を用いることができなくなる。しかし、剛性が高くなると接着長さの変化に伴う強度の変動は少なくなるので、ここでは荷重は各接着層に均一に作用し、その見かけのせん断応力（τ）は

$$\tau = P/(n-1)A$$

P ： 荷重（kg）
A ： 接着面積（cm^2）
n ： プライ数

で与えられるものとした。

4） L≧25mmでは見かけのせん断強度は接着長さによらず一定であるとし、L<25mmでは、見かけのせん断強度は3プライの時と同じになるとする。

5） 見かけのせん断強度をτ_{max}、みかけの作用せん断応力をτとすると、$\tau_{max}>\tau$となった時点で接着層はせん断破壊を起こす。

以上の仮定に基づいて接着長さLの時のnプライの単板積層材の単位幅あたりの最大荷重の分布関数$F_{nL}(x)$を求める。接着長さがLの時の単位幅あたりの接着層のせん断強度が分布関数$F_{\tau L}(x)$、それ以外の破壊源の単位幅あたりの強度が分布関数$F'_{0n}(x)$に従うとすると、式(4.5)より

$$F_{nL}(x) = 1 - \{1 - F_{\tau L}(x)\}^{n-1} \times \{1 - F_{0n}(x)\} \qquad (4.46)$$

n ： プライ数

で全体としての強度の分布関数を与えることができる。さらに鎖モデルの連鎖則に従い、n-1個の接着層を一つの連と考えてその強度分布が分布関数$F'_{L_{n-1}}(x)$に従うとすると、式(4.46)は、

$$F_{nL}(x) = 1 - \{1 - F'_{L_{n-1}}(x)\} \times \{1 - F_{0n}(x)\} \qquad (4.47)$$

n ： プライ数

と書き換えられることになる。

n-1個の接着層の単位幅あたりの強度が従う分布関数$F'_{L_{n-1}}(x)$は、4.3.5節で行なった引張りせん断試験の結果から推定して与えることができる。ただし、この実験は二重重ね継手の形状の試験体に対して行ったものだから、この時の分布関数は$F'_{L_2}(x)$に相当するものであり、したがって接着長さLの時実験データから推定された2パラメータワイブル分布の形状母数をα_L、尺度母数をβ_Lとすると、

$$F'_{L_2}(x) = 1 - \exp\left\{-\left(\frac{x}{\beta_L \times L \times 2}\right)^{\alpha_L}\right\} \qquad (4.48)$$

ただし、$L \geqq 25$mmでは、$\alpha_L = 41.6$
$\beta_L = 8.60$

で与えられることになる。これをnプライの試験体に拡張すると、

$$F'_{L_{n-1}}(x) = 1 - \exp\left\{-\frac{n-1}{2}\left(\frac{x}{\beta_L \times L \times (n-1)/2}\right)^{\alpha_L}\right\} \qquad (4.49)$$

ただし、$L \geqq 25$mmでは、$\alpha_L = 41.6$
$\beta_L = 8.60$

で与えられることになる。

接着せん断以外の破壊源の単位幅あたりの強度の分布関数は、4.3.6節で述べた引張り試験の結果から推定して与える。その実験から得られた母集団分布関数は複数の破壊源を一つの連とみなしたときの分布関数であり、プライ数を増やした場合に個々の破壊源の数が同じ割合で増加するとはいえないのであるが、ここでは便宜的に、

「仮定1で定義した破壊源は、プライ数がnの時には(n-1)/2個存在するとし、またその負担荷重は、外力がPの時2P/(n-1)になるとする。」

という仮定が成り立つとする。したがって分布関数 $F_{0n}(x)$ は、

$$F_{0n}(x)=1-\left\{1-F_3\left(\frac{2x}{n-1}\right)\right\}^{(n-1)/2}$$
$$=1-\exp\left\{-\frac{n-1}{2}\left(\frac{2x}{285\times(n-1)}\right)^{42}\right\} \quad (4.50)$$

で与えられる。

式（4.49），（4.50）を式（4.47）に代入すれば分布関数 $F_{nL}(x)$ を得ることができ、

$$F_{nL}(x)=1-\exp\left\{-\frac{n-1}{2}\left(\frac{x}{\beta_L L(n-1)/2}\right)^{\alpha_L}-\frac{n-1}{2}\left(\frac{2x}{285(n-1)}\right)^{42}\right\}$$

ただし、$L\geqq 25$mmでは、$\alpha_L=41.6$　　(4.51)

$\beta_L=8.60$

となる。

Table 4.7に3プライ単板積層材の実験結果を2パラメータワイブル分布にあてはめた時の、各接着長さにおける形状母数と尺度母数の値を示した。$L<25$mmの場合にはこの値を使って計算を行った。

Table 4.7. Weibull parameter in each lap-length of shear tests of 3-ply LVL.

L (mm)	α	β
5	8.91	111.0
10	8.44	73.2
15	11.2	60.8
20	9.69	45.6
25	8.60	41.6

Notes: L: Lap length, α: Shape parameters of Weibull distribution, β: Scale parameters of Weibull distribution.

4.4.2 縄モデルによるモデル化

プライ数が増え破壊源が多くなると一番弱い要素が破壊したときに全体の破壊が起こるという鎖モデルの仮定が成り立たなくなってくる可能性がある。そこで5プライ、9プライの試験体に対しては、その破壊に縄モデルを当てはめてモデル化してみた。縄モデルによるモデル化を行うために以下に示したような仮定をおこなった。

1）破壊は、接着層とジョイント部分の単板で起こるものとする。

2）その破壊源のうちのどれか1つが破壊すると荷重の最配分が起こる。これが繰り返されて全ての破壊源が破壊した場合に破壊が起こるとする。破壊荷重としては破壊が進行する過程における荷重の最大値をとる。

3）各破壊源の強度は互いに独立である（相関がない）ものとする。

4）荷重は各接着層に均一に作用し、その見かけの作用せん断応力（τ）は

$$\tau=P/nA$$

P：荷重（kg）
A：接着面積（cm^2）
n：残存している接着層の数

で与えられるものとする。

5）$L\geqq 25$mmでは見かけのせん断強度は接着長さによらず一定であるとし、$L<25$mmでは、見かけのせん断強度は3プライの時と同じになるとする。

6）見かけのせん断強度（τ_{max}）＞見かけの作用せん断応力（τ）

となった時点で接着層はせん断破壊を起こす。

7）ジョイント部分の単板に作用する引張り応力（σ）は

$$\sigma=P/nS$$

P：荷重（kg）
A：単板の断面積（cm^2）
n：ジョイント部分の残存している単板の数

で与えられるものとし、単板の破壊は、

作用引張り応力（σ）＞引張り強度（σ_{max}）

となった時点で起こるものとする。

さてこの仮定で問題となるのは仮定1であり、前章で述べたように、ジョイント部分の単板が破壊を起こすとは限らないのである。鎖モデルでは連鎖則が成り立つので引張り破壊とその他の破壊を含めて一つ

の破壊とみなすことができたが縄モデルではそのように扱うことはできず、個々の強度の分布関数を求めなければならないことになる。しかしその破壊形態は、

(1)左右ジョイントの内側での単板の破壊

(2)左右ジョイントの外側の破壊

とに分けることが可能であり、(2)の破壊は左右ジョイントの内側の破壊の進行に影響を受けないし、また逆に影響を及ぼすこともないことは明らかである。以上の事実から左右ジョイントの内側の破壊に対しては縄モデルを適用し、これを一つの連とみなして、全体としての破壊はジョイントの内側の破壊と、ジョイントの外側の破壊どちらかが起こった時点で起こると考えて鎖モデルを適用するのが適当であると思われる。したがって、接着層のせん断強度の分布関数と、(1)の破壊強度の分布関数及び(2)の破壊強度分布をそれぞれ求めれば良いことになる。

プライ数がnの時の単位幅あたりの接着層の強度の分布関数は4.3節で用いたTYPE Iの試験体から求められ、接着長さLの時の実験データから推定された2パラメータワイブル分布の形状母数をα_L、尺度母数をβ_Lとすると、

$$F_{u_n}(x)=1-\exp\left\{-\frac{1}{2}\left(\frac{x}{\beta_L L(n-1)/2}\right)^{\alpha_L}\right\} \qquad (4.52)$$

ただし、$L\geqq 25$mmでは、$\alpha_L=41.6$

$\beta_L=8.60$

で与えられる。

プライ数がnの時の単位幅あたりのジョイント部分の単板の破壊強度の分布関数は4.3節で用いたTYPE II引張り試験から求められ、

$$F_{o_n}(x)=1-\exp\left\{-\left(\frac{x}{260\times(n-1)/2}\right)^{3.9}\right\} \qquad (4.53)$$

で与えられる。

プライ数がnの時の単位幅あたりのジョイントの外側部分の破壊強度の分布関数も4.3節で用いたTYPE IIの引張り試験から求められ、

$$F_{o_n}(x)=1-\exp\left\{-\left(\frac{x}{319\times(n-1)/2}\right)^{7.69}\right\} \qquad (4.54)$$

で与えられる。

縄モデルでは、その分布関数を式の形で求めることはできないのでモンテカルロシミュレーションを行ってその結果を見た。試行は各接着長さについて1000回繰り返して、その分布型と平均値、分散および破壊様式を求めた。

4.4.3 5プライ、9プライ単板積層材の引張り試験

次に、実際に5プライ、9プライの単板積層材を作成して引張り試験を行った。使用した単板及び接着条件は3プライの時と同じである。試験体形状もFig.4.14,15に示したように、3プライの試験体を積み重ねた形状である。荷重速度は、今までの試験と同じくクロスヘッドスピード5mm/min.であった。5プライの試験体の実験結果をTable.4.8, Fig.4.16に示した。9プライの試験体の実験結果はTable 4.9 , Fig.4.17に示した。

5プライの場合には3プライの時と同様にL=30から40mmあたりで頭打ちになる傾向がみてとれる。9プライの場合は、強度の平均値については、特にはっきりした傾向を見いだすことはできないが変動係数の値を見てみると5プライの試験結果が3プライの結果とほとんど変わらないのに対して全ての接着長さで小さくなっていることがわかる。これは、9プライでは3プライの時よりも破壊源が多くなり、破壊の進行に伴う応力の最分配が行われていることを示していると思われる。

Table 4.8. Results of tensile tests of 5-ply test specimens.

Lap length(mm)		Number of Specimens	Ratio	P_{max} / w(kgf/mm)	
Avg.	S.D.		(%)	Avg.	S.D.
5.38	0.43	66		205.4	40.4
5.74	0.14	3	4.55	229.1	61.5
5.36	0.44	63	95.45	204.3	39.5
9.68	0.88	65		294.4	36.9
10.40	0.71	3	4.62	304.8	72.8
9.65	0.88	62	95.38	293.9	35.4
19.19	0.50	62		404.2	54.2
19.08	0.20	16	25.81	348.0	56.9
19.23	0.56	46	74.19	423.7	37.3
30.21	0.58	63		493.1	78.2
30.16	0.66	16	25.40	423.7	90.6
30.22	0.55	47	74.60	516.7	57.7
40.09	0.40	60		512.0	65.5
40.03	0.43	43	71.67	509.4	73.8
40.24	0.29	17	28.33	518.5	37.9

Notes: P_{max}: Maximum load(kgf), w: Width of specimens.
Notes: 1st line shows all of specimens, 2nd line shows specimens failed except for bonded parts, 3rd line shows specimens failed in bonded parts.

Table 4.9. Results of tensile tests of 9-ply test specimens.

Lap length(mm)		Number of Specimens	Ratio	P_{max} / w(kgf/mm)	
Avg.	S.D.		(%)	Avg.	S.D.
5.38	0.43	66		205.4	40.4
-	-	0	0.00	-	-
-	-	66	100.0	205.4	40.4
9.33	0.33	38		528.8	51.5
-	-	0	0.00	-	-
-	-	38	100.0	528.8	51.5
19.90	0.30	42		826.9	78.3
19.80	0.35	12	28.57	814.4	75.5
19.90	0.28	30	71.43	831.9	80.1
39.82	0.74	41		1021.0	88.7
39.92	0.79	27	65.85	1022.0	100.0
39.62	0.61	14	34.15	1017.0	64.3

Notes: P_{max}: Maximum load(kgf), w: Width of specimens.
Notes: 1st line shows all of specimens, 2nd line shows specimens failed except for bonded parts, 3rd line shows specimens failed in bonded parts.

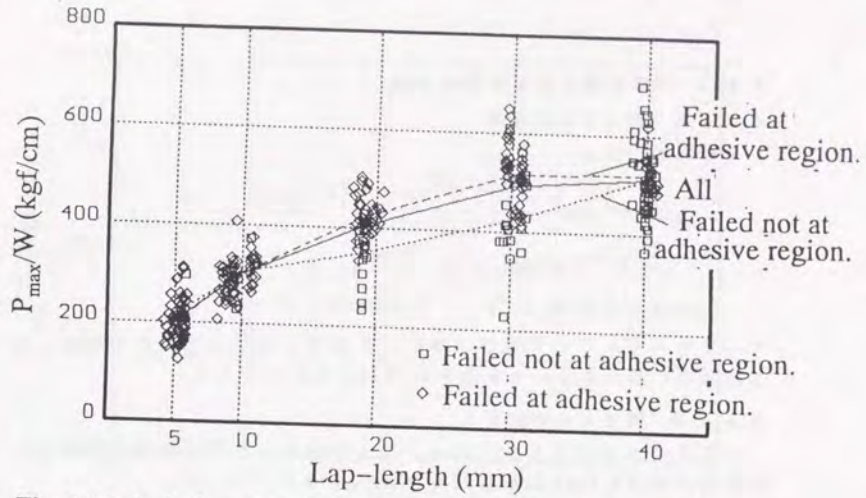

Fig.4.16. Relationships between lap-length and P_{max}/W (5-ply).
Legend : P_{max} : Maximum load (kgf). W : Width of specimens (mm).

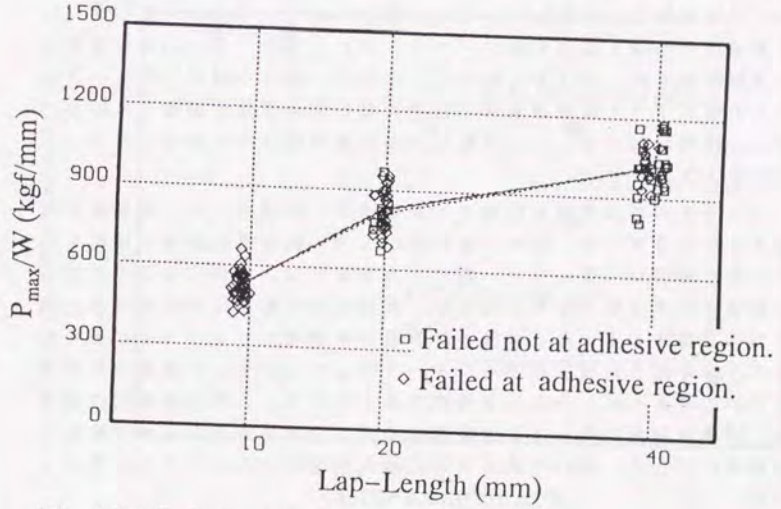

Fig.4.17. Relationships between lap-length and P_{max}/W (9-ply).
Legend : P_{max} : Maximum load (kgf). W : Width of specimens (mm).

4.4.4 実験結果と計算結果の比較

4.4.4.1 鎖モデルの結果

分布関数の式は、

$$F_{nL}(x)=1-\exp\left\{-\frac{n-1}{2}\left(\frac{x}{\beta_L L(n-1)/2}\right)^{\alpha_L}-\frac{n-1}{2}\left(\frac{2x}{285\times(n-1)}\right)^{42}\right\} \qquad (4.55)$$

ただし、L≧25mmでは、$\alpha_L=41.6$

$\beta_L=8.60$

で与えられる。この式及び、周辺分布関数の式から求めた平均値、せん断破壊の割合を示したものをFig.4.18から21に示した。

4.4.4.2 縄モデルの結果

モンテカルロシミュレーションにより得られた平均値及び接着せん断破壊の割合をFig.4.22から25及びTable 4.10 ,11に示した。

4.4.4.3 考察

5プライの試験体について鎖モデルの結果を見ると、どの接着長さでも計算値は実測値を大きく下回っている。しかし、せん断破壊の割合では実測値とほぼ同じ傾向を示している。また、縄モデルによるシミュレーション結果を見ると、この場合は強度、せん断破壊の割合両方とも実測値とは一致していない。鎖モデルで、傾向は一致しながら計算値の方が低くなるのは、パラメータとして3プライの単板積層材の実験結果を用いたためであると考えられ、適正な初期パラメータを与えれば5プライ単板積層材の破壊は最も弱い部分が破壊した時点で全体の破壊が起こる、という鎖モデルの適用範囲内であると言うことができよう。

9プライの単板積層材に鎖モデルを適用した場合、その計算値は実測値をかなり下回る。また、接着長さとせん断破壊の割合の関係を見ても余り傾向は一致しない。縄モデルを適用した場合にもその強度の計算値は実測値をかなり下回るが、破壊形態の変化は実際の場合と同じ傾向で推移している。強度の計算値が実測値よりも小さくなるのは5プライの時と同様に初期パラメータとして3プライの実験値を用いたためと考えられ、その点を考慮すると9プライの単板積層材の破壊は、破壊の進行に伴って各破壊要素の応力分担が変化しながら最終的な破壊にいたる、という縄モデルの適用範囲内に入ってくると考えられる。

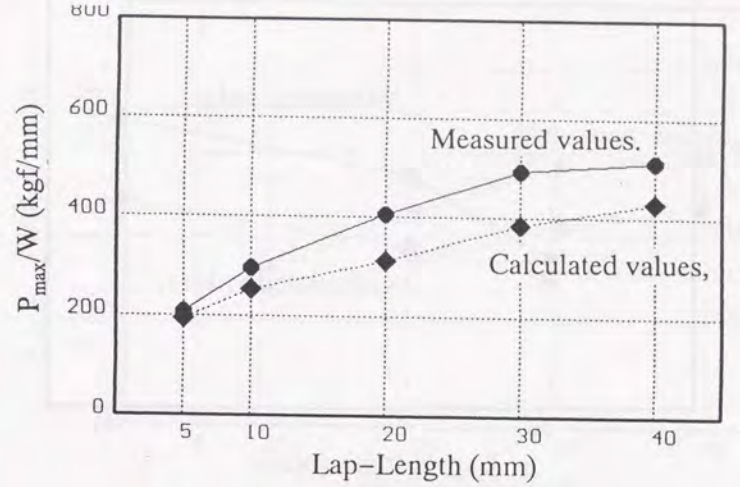

Fig.4.18. Relationship between lap-length and P_{max} / w (5-ply).
Calculated by chain model and measured values of all specimens.
Notes: P_{max}: Maximum load(kgf), w: Width of specimens(mm).

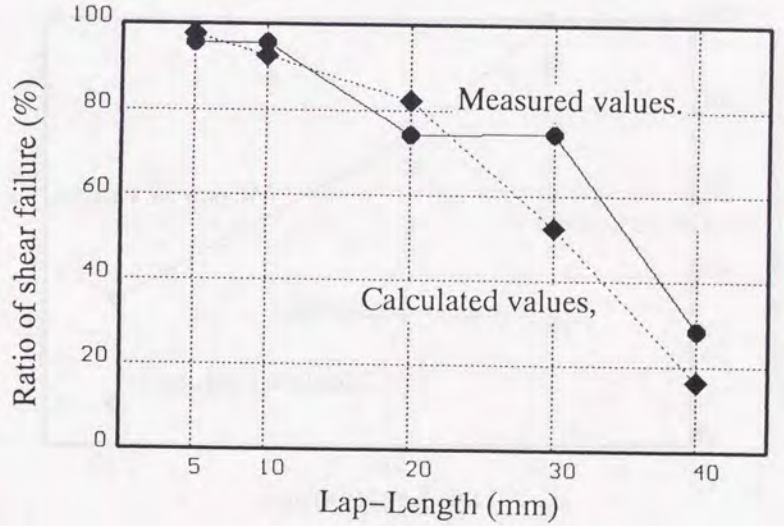

Fig.4.19. Relationship between lap-length and ratio of shear failure (5-ply).
Calculated by chain model and measured values of all specimens.

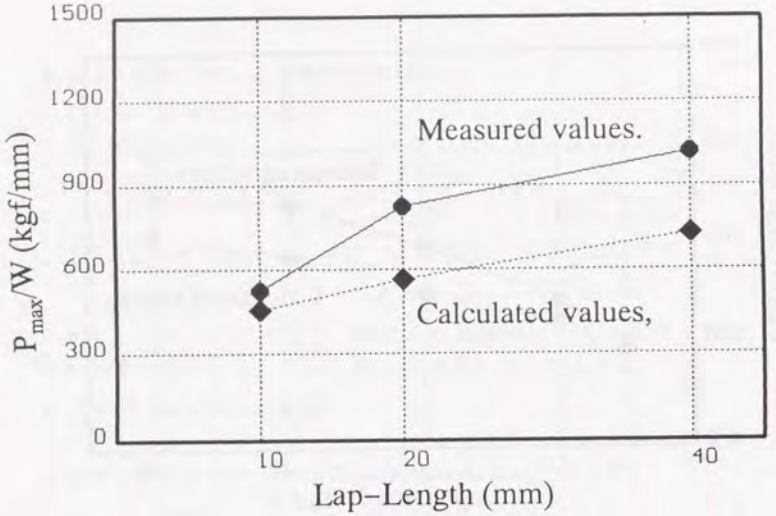

Fig.4.20. Relationship between lap-length and P_{max} / w (9-ply).
Calculated by chain model and measured values of all specimens.
Notes: P_{max}: Maximum load(kgf), w: Width of specimens(mm).

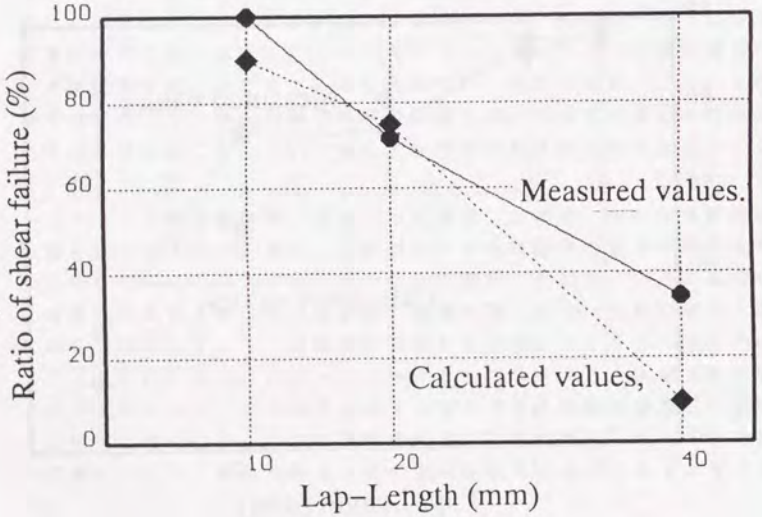

Fig.4.21. Relationship between lap-length and ratio of shear failure (9-ply).
Calculated by chain model and measured values of all specimens.

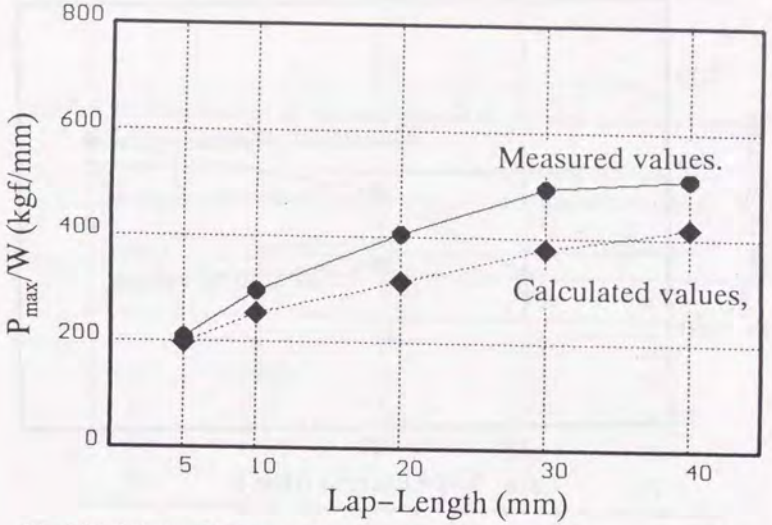

Fig.4.22. Relationship between lap-length and P_{max} / w (5-ply).
Calculated by rope model and measured values of all specimens.
Notes: P_{max}: Maximum load(kgf), w: Width of specimens(mm).

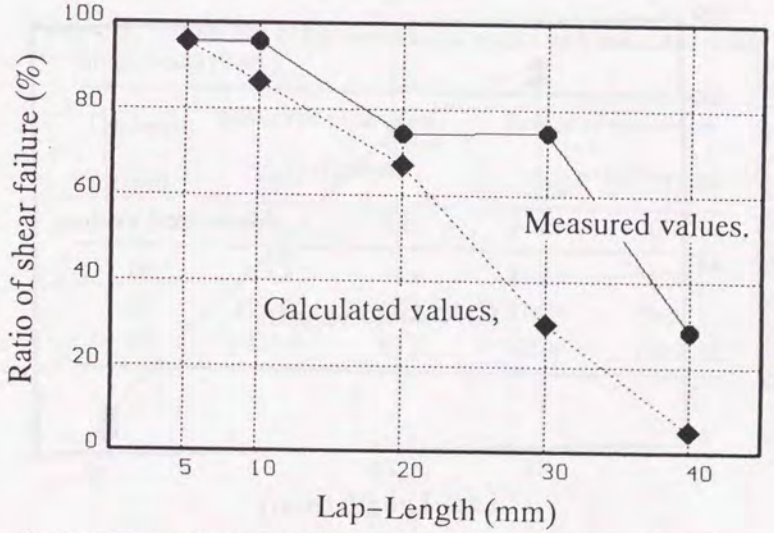

Fig.4.23. Relationship between lap-length and ratio of shear failure (5-ply).
Calculated by rope model and measured values of all specimens.

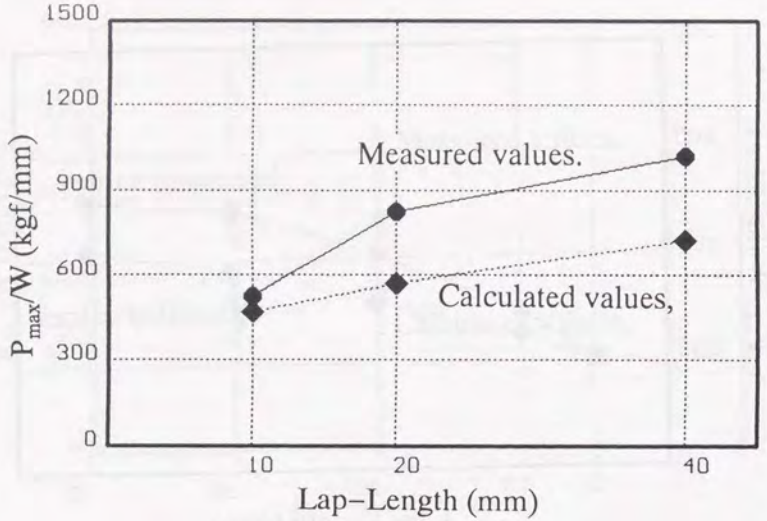

Fig.4.24. Relationship between lap-length and P_{max} / w (9-ply).
Calculated by rope model and measured values of all specimens.
Notes: P_{max}: Maximum load(kgf), w: Width of specimens(mm).

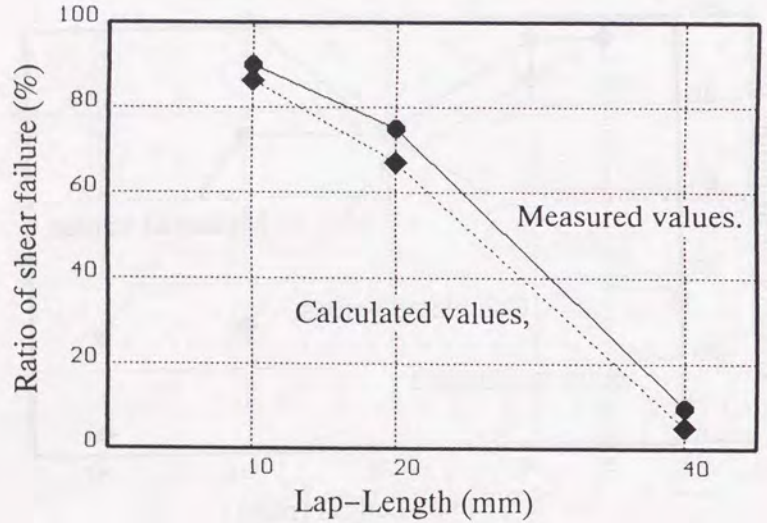

Fig.4.25. Relationship between lap-length and ratio of shear failure (9-ply).
Calculated by rope model and measured values of all specimens.

Table 4.10. Comparison of the experimental results with calculated results of the model (5-ply).

Lap length (mm)	Results of experiment P_{max} / w(kgf/mm)		Results of simulation P_{max} / w(kgf/mm)	
	Avg.	S.D.	Avg.	S.D.
5	205.4	195.0	195.0	24.3
10	294.4	251.7	251.7	34.1
20	404.2	312.7	312.7	46.7
30	493.1	378.4	378.4	75.5
40	512.0	418.3	418.3	97.2

Table 4.11. Comparison of the experimental results with calculated results of the model (9-ply).

Lap length (mm)	Results of experiment P_{max} / w(kgf/mm)		Results of simulation P_{max} / w(kgf/mm)	
	Avg.	S.D.	Avg.	S.D.
10	528.8	51.5	473.5	56.28
20	826.9	78.3	574.9	79.33
40	1021.0	88.7	725.9	139.2

4.5 結論

本章で得られた結果をまとめると以下のようになる。

1) 3プライでは、接着長さの範囲が広くなると接着層に作用する応力を解析することが必要になってくる。しかし、実験結果により得られた接着長さと接着強度の分布を回帰した式を適用することにより、強度予測を行うことが可能となった。

2) 5プライ、9プライとプライ数を増やしていくと、破壊源の数が増加するために3プライの実験データを基にした鎖モデルでは説明しきれなくなり、縄モデルの適用範囲内に入ってくる。

以上より、複数であるが小数の破壊源を持つ物体の破壊については、その個々の破壊源の破壊機構がわからなくても統計的に処理したデータに基づいて鎖モデルを適用することによりその強度を予測することが可能であることが示された。しかし、破壊源が増えていき、鎖モデルではモデル化しきれず縄モデルを導入しなければならなくなると、個々の破壊源の強度についての精密な実験データもしくは理論的に導かれた値というものが必要となってくる。もちろん、破壊源の多い試験体に対しては、例えば9プライの単板積層材ならば9プライの試験体の接着せん断強度、引張り強度を測定してこれをそれぞれ一つずつの破壊源とみなして鎖モデルを適用するというように、何個かの破壊源を一まとめにした一つの破壊源とみなして実験から強度分布を求めて鎖モデルを適用することも可能なのであるが、それではいちいち破壊試験を行わねばならずモデル化を行う意味がなくなってしまうことになる。今後は、破壊源の応力解析等の破壊機構の解析と確率的手法を組み合わせたモデル化が必要となろう。

第5章 シミュレーションによる木材の破壊強度の予測

5.1 研究の目的と概要

第3章で木理の数値化と、その数値を用いて木材の異方性を有する物性の中の代表的なものである、収縮、応力波の伝播速度、ヤング係数に対する木理の影響についての検討を行なった。これらの物性に加えて木理の影響を強く受ける木材の物性として強度を挙げることができる。木材は縦引張り強度は非常に強いが、それに対して横引張り強度、せん断強度は非常に弱い。そのため目切れや節のによる繊維走向の乱れの影響により外力の作用方向に対して平行でない面が存在するとその部分で破壊を起こす。しかし、それと同時に材内の物性のばら付きが存在するために、ある部分が破壊を起こしても他の強い部分がその分の外力を負担してなかなか最終的な破壊までいたらない、という延性材料的な面も併せ持っている。このような性質を持つ材料の強度を評価するには、4章で扱った強度確率モデルが適している。前者の破壊現象は、鎖モデルや競合リスクモデルに相当し、後者は並列モデルに相当している。

この章では、3章で考案した木理の数値化法を用いて目切れを含む任意の木取りを持つ木材をモデル化し、これに対して確率モデルを適用してシミュレーションを行なうことによってその強度を推定をおこない、木取りが木材の強度に及ぼす効果について検討を行なった。

5.2 繊維モデルによる木材のモデル化

この節では、繊維傾斜を考慮に入れた木材の破壊モデルである「繊維モデル」を以下に示した仮定に従うモデルとして定義する。

（仮定１） 木材をFig.5.1に示したような正方形の断面を持つ複数の繊維要素の集合体であると考える。その断面積の大きさが比重を表わすものとする。

（仮定２） 木材中の任意の点におけるL軸方向（3.3節で定義）のヤング係数は比重に比例するとする。

（仮定３） TLR座標系（3.3節で定義）は木材中における位置に依存するため、一本一本の繊維要素の軸方向がL方向となす角度（以降、この角度のことをω_{Li}と表記する。添字の第i番目の繊維要素であることを表わす）は要素ごとに異なる。この角度ω_{Li}の大きさで繊維要素のヤング係数が決まるとする。ω_{Li}が０の時のヤング係数は材のL軸方向のヤング係数と一致し、ω_{Li}が大きくなるほどヤング係数は低下するとする。

（仮定４） i番目の繊維要素はFig.5.2に示したように、その内部にL軸方向となす角度ω_{Li}だけ傾いた平面を持つとする。これを繊維要素の目切れ面と呼ぶ。ω_{Li}が０の場合にはこの平面は存在しない。

（仮定５） i番目の繊維要素の木口面に対して外力σ_{ti}が作用した時、目切れ平面に垂直方向の主応力$\sigma_{\perp i}$と平行方向、すなわちL軸方向の主応力σ_{Li}、及びせん断応力τ_iが作用する。その大きさはそれぞれ、

$$\begin{aligned}\sigma_{Li} &= \sigma_{ti}\cos^2\omega_{Li}\\ \sigma_{\perp i} &= \sigma_{ti}\sin^2\omega_{Li}\\ \tau_i &= \sigma_{ti}\sin\omega_{Li}\cos\omega_{Li}\end{aligned} \qquad (5.1)$$

で与えられる。繊維要素の破壊強度はこの３つの応力とそれに対する強度の関係で決まる。すなわち、それぞれの方向の強度を$\sigma_{L\max i}$, $\sigma_{\perp\max i}$, $\tau_{\max i}$とすると、破壊の条件式はそれぞれ

$$\begin{aligned}\sigma_{L\max i} \le \sigma_{Li} = \sigma_{ti}\cos^2\omega_{Li} &\Leftrightarrow \sigma_{ti} \ge \frac{\sigma_{L\max i}}{\cos^2\omega_{Li}}\\ \sigma_{\perp\max i} \le \sigma_{\perp i} = \sigma_{ti}\sin^2\omega_{Li} &\Leftrightarrow \sigma_{ti} \ge \frac{\sigma_{\perp\max i}}{\sin^2\omega_{Li}}\\ \tau_{\max i} \le \tau_i = \sigma_{ti}\sin\omega_{Li}\cos\omega_{Li} &\Leftrightarrow \sigma_{ti} \ge \frac{\tau_{\max i}}{\sin\omega_{Li}\cos\omega_{Li}}\end{aligned} \qquad (5.2)$$

で与えられる。この３つの条件の内いずれか一つの条件が満足されれば繊維要素が破壊するとする。繊維要素の強度もヤング係数と同じく比重に比例すると考える。

（仮定６） 繊維要素が一本破壊すると応力の再分配が起こり他の要素が受け持つ応力が増大する。

このモデルでは、（仮定５）の部分が脆性的な破壊を、（仮定６）の部分が延性的な破壊をモデル化していることになる。

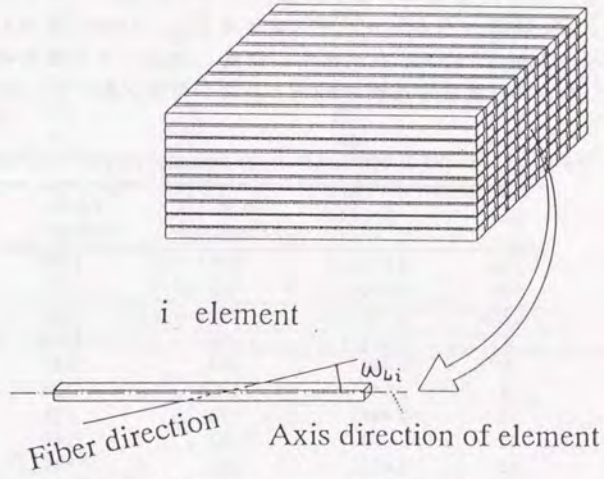

Fig.5.1. Schematic diagram of fiber model.

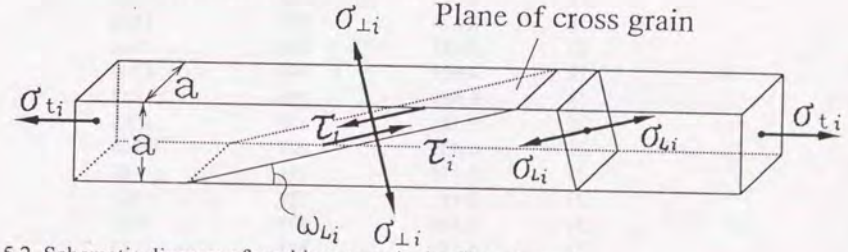

Fig.5.2. Schematic diagram of working stress in the fiber-element.

Notes: σ_{ti}: Tensile stress which work on the cross section of fiber element, $\sigma_{\perp i}$: Normal stress which work on the sloping plane, σ_{Li}: Normal stress which work on the perpendicular plane to the sloping plane (along L-direction) , τ_i: Shear stress which work on the sloping grain. ω_{Li}: Angle between L-axis and direction of fiber element.

5.3 繊維傾斜のみを考慮した破壊のモデル化

この節では、繊維モデルの最も単純な形態である繊維傾斜のみを考慮した木材のモデルを用いて木材の引張り、曲げ強度の予測を行なった。

5.3.1 カラマツラミナの曲げ破壊試験

実験結果と比較検討を行うために、曲げ破壊試験を行なった。使用した材は、3章でヤング係数、応力波の評価に用いたカラマツのラミナである。曲げ試験は中央集中荷重でスパンは60cm、クロスヘッドスピード10mm/minで行なった。その結果得られたMORの値をMOE、比重と併せてTable.5.1に示した。また、比重、MOEとの相関関係をFig.5.3,4に示した。曲げ強度は比重、MOEと非常に相関が高いことがわかる。

Table 5.1. Results of bending Rupture tests and properties of lumbers.

No.	ρ	MOE ($x10^3kgf/cm^2$)	MOR (kgf/cm^2)
1	0.625	208	1317
2	0.590	162	1077
4	0.546	192	1184
5	0.531	166	1100
6	0.489	149	816
8	0.584	181	1120
10	0.464	136	779
12	0.592	182	1260
13	0.458	120	594
14	0.512	171	976
15	0.611	212	1370
16	0.574	204	1138
17	0.442	110	542
18	0.607	191	1309
19	0.609	196	1357
20	0.625	218	1359
21	0.605	207	1346
22	0.572	191	986
23	0.451	129	752
24	0.546	197	1257
25	0.557	182	1123
26	0.584	186	1253
27	0.504	146	583
28	0.572	174	1129
29	0.595	191	1260
30	0.653	221	1530
31	0.482	157	873
32	0.533	171	933
33	0.631	212	1171
34	0.637	226	1507
35	0.564	185	1258
36	0.539	161	1071
37	0.516	150	754
38	0.498	164	674
39	0.613	211	1400
40	0.618	215	1390
41	0.518	176	1086
42	0.623	171	1211
43	0.663	235	1410
44	0.582	184	1140
45	0.583	195	1235
46	0.594	196	1254
47	0.509	147	645
48	0.611	158	1166
49	0.487	155	916
50	0.564	176	973
51	0.532	149	804
52	0.505	149	995
Avg.	0.560	179	1091

Notes: ρ: Specific gravity, MOE : Modulus of Elasticity, MOR: Modulus of rapture.

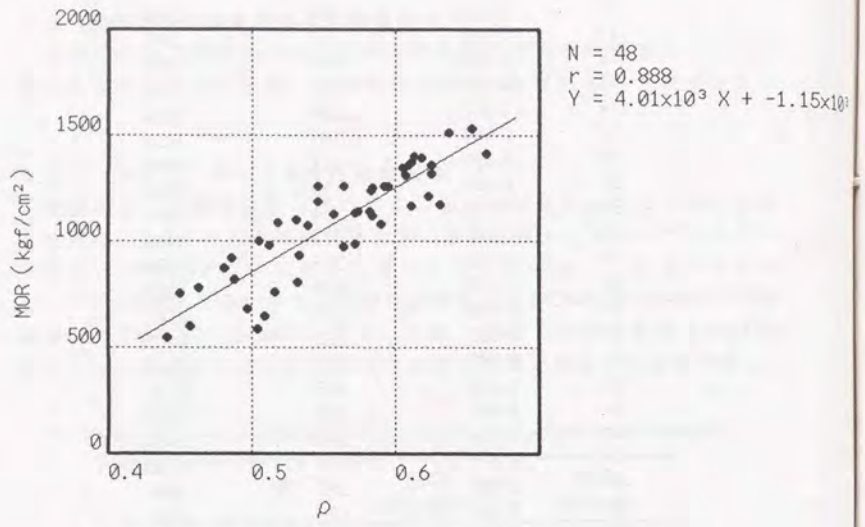

Fig.5.3 . Relationship between ρ and MOR.

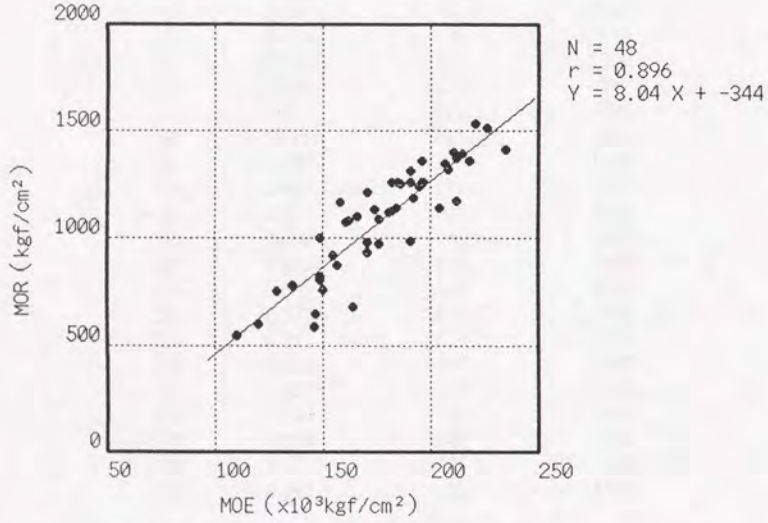

Fig.5.4 . Relationship between MOE and MOR.

5.3.2 引張り破壊モデル

5.3.2.1 モデル化

この節では、繊維モデルを用いて実際に木取りを測定した木材のモデル化を行い、その引張り強度の推定を行なった。対象とした材は3章でヤング係数、応力波の評価を行ったカラマツのラミナである。以下に前章で定義した仮定に基づくカラマツラミナのモデル化の過程を述べる。

（仮定１） 木口面を一辺が1mmの正方形に分割し、繊維要素はこれを断面として持つ直方体であるとする。ここでは比重のばら付きを考慮に入れていないので、すべての繊維要素の断面積は同じ、すなわち、一辺の長さがすべて同じで繊維要素が互いに密着しているような形態でモデル化できる。

（仮定２） L方向のヤング係数の比重に対するバラ付きは考慮に入れず、また比重の大きさも考慮に入れていないので、L方向のヤング係数を全て1であるとすることができる。したがってシミュレーションの結果得られるヤング係数はL方向のヤング係数に対するヤング係数の低減率に相当する。

（仮定３） 繊維要素の軸方向のヤング係数のL方向に対する低減率であるが、ここでは、3.7節で求めた応力波伝播速度と、L軸と伝播方向ベクトルとのなす角度β_Lの間の回帰式を用いて推定を行なう。Fig.3.50に示したように$\tan\beta_L$と伝播速度vの間には、

$$v = -6090\tan\beta_L + 5820 \qquad (5.3)$$

という関係が成り立つ。また、ヤング係数Eと伝播速度vの間には、比重ρを用いると、$E=\rho v^2$という関係が成り立つことが知られている。この節のモデルではすべての繊維要素の比重は同一としたのでEとv^2の間に比例関係が成り立つと考えて良い。そこで、第i番の繊維要素の軸方向のヤング係数は、繊維要素の軸方向とL方向がなす角度ω_{Li}を用いて、

$$E = \left(-1.033\omega_{Li} + 1\right)^2 \qquad (5.4)$$

で与えられるとする。

（仮定４） 繊維要素はFig.5.2に示したように、その内部にL方向となす角度ω_{Li}だけ傾いた平面を持つとする。T方向、R方向の強度異方性は考えないので、その平面は繊維要素の下（上）の面に対してω_{Li}だけ傾いた平面であるとして良い。

（仮定５） 比重に対する繊維要素の強度のバラ付きは考慮に入れず、そのL方向（目切れ面の平行方向）の強度は全て1であるとする。したがってシミュレーションの結果得られる強度は、L方向の強度に対する強度低減率に相当することになる。また、目切れ面に対して垂直方向の強度、及びせん断強度も繊維要素によって変わらないとし、その大きさをそれぞれ$\sigma_{\perp max}$,τ_{max}とする。しかし、それぞれの大きさのL方向の強度に対する比率は不明なので、$\sigma_{\perp max}$,τ_{max}の値を適当に変動させてシミュレーションを実行し、その値に依存する破壊強度、破壊形態の変化を検討することにする。

各繊維要素の破壊の条件は、以下のように考えて定める。繊維要素は塑性変形域を持たないとし、そのヤング係数がE_iすると、そのひずみε_iと応力σ_{ii}の間には、

$$\sigma_{ii} = E_i \varepsilon_i \qquad (5.5)$$

という関係が成り立つので破壊条件式(5.2)は、ε_{ij}については、

$$\varepsilon_i \geq \frac{\sigma_{L\max}}{E_i \cos^2 \beta_{Li}}$$
$$\varepsilon_i \geq \frac{\sigma_{\perp \max}}{E_i \sin^2 \beta_{Li}} \qquad (5.6)$$
$$\varepsilon_i \geq \frac{\tau_{\max}}{E_i \sin\beta_{Li} \cos\beta_{Li}}$$

となる。後述するように、引張り破壊では応力よりもひずみを破壊条件として用いた方が簡便なモデル化が可能なので、式(5.6)を繊維要素の破壊条件式として用いる。

（仮定６） 繊維要素の破壊による応力の再配分と、そのくり返しによる破壊の進行は、Fig.5.5に示したように模式的に表わすことができる。材全体としてのヤング係数をEとし、材のみかけの断面積をAとする。材の断面に応力σが作用し、その時のひずみがεであったとすると、みかけのヤング係数Eは、フックの並列バネの法則から、

$$A\sigma = \sum A_i \sigma_i$$
$$AE\varepsilon = \sum A_i E_i \varepsilon \qquad (5.7)$$
$$E = \frac{\sum A_i E_i}{A}$$

で与えられる。ただし、A_iは第i繊維要素の断面積である。

ここでは、繊維要素の断面積はすべて同一で、繊維の間に隙間が無いとしたので、式(5.7)は繊維要素の総数nを用いて、

$$E = \frac{\sum E_i}{n} \qquad (5.8)$$

と書き換えることができる。このEがFig.5.5の初期の応力－ひずみ直線に相当する。

材全体のひずみが増大すると、破壊条件式(5.6)の条件を満たす繊維要素が現れ、破壊を起こす。繊維要素の内の一本が破壊を起こすと、その時のみかけのヤング係数E'は式(5.8)より、

$$E' = \frac{\sum_{i}^{n-1} E_i}{n} \qquad (5.9)$$

と変化する。その直線関係がFig.5.5の破線の部分に相当し、その時の材に作用するみかけの応力はσ'になる。繊維要素の側からみれば引張り力を受け持っていた繊維要素が一本消滅したことによりその応力分配が変化し、前よりも大きな力を負担しなければならなくなったことになる。この過程が繰り返されると次第に応力－ひずみ直線の傾きが小さくなりある時点でピークを迎えることになる。その時の応力が材の最大応力σ_{max}、破壊ひずみε_{max}に相当する。

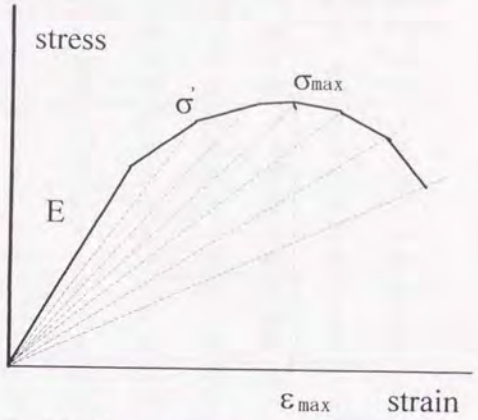

Fig.5.5. Schematic diagram of tensile failure process in Fiber model.

以上の手順にしたがってコンピュータシミュレーションにより応力のピークを求め、そこに至るまでの繊維要素の破壊形態を記憶しておけば、任意の木取りを持つ材の引張り強度と破壊形態を推定することができる。

5.3.2.2 強度推定結果

Table 5.2に示したような$\sigma_{\perp max}$, τ_{max}の値の組み合わせを用いて引張り破壊シミュレーションを行い、各試験体の強度推定を行なった。

Fig.5.6にシミュレーションで得られた引張りヤング係数低減率で曲げヤング係数の実測値を割った値と比重の関係を示した。また、Table 5.2, 3に強度の結果を示す。Table 5.2は、破壊形態ごとに、その条件下

でその破壊形態で破壊した試験体の総数を示している。また、Table 5.3は、実験で得られた曲げ破壊強度を、各条件下で推定された強度低減率で割った値と比重との相関分析を行なった結果を示している。また、Fig.5.7, 8には、シミュレーション結果から得られた応力ーひずみ曲線の例を示した。

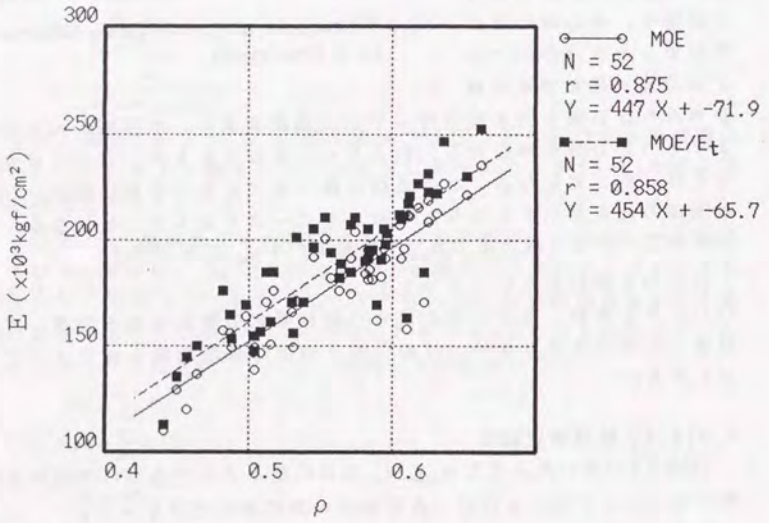

Fig.5.6. Relationship between ρ and MOE/E_t.

Notes : ρ: Specific gravity, MOE: Modulus of elasticity in bending test, E_t: Ratio of E-reduction estimated by simulation.

Table 5.2. Strength condition of fiber element and totals of tensile simulated species fractured by each failure mode.

$\sigma_{\perp max}$	τ_{max} 1	0.1	0.08	0.07	0.06	0.05	0.04	0.03	0.02	0.01
1	52	52	50	49	45	42	34	24	9	1
	0	0	0	0	0	0	0	0	0	0
	0	0	2	3	7	10	18	28	43	51
0.01	51	51	50	49	45	42	34	24	9	1
	1	1	0	0	0	0	0	0	0	0
	0	0	2	3	7	10	18	28	43	51
0.005	49	49	49	49	45	42	34	24	9	1
	3	3	3	3	2	1	0	0	0	0
	0	0	0	0	5	9	18	28	43	51
0.004	45	45	45	45	45	42	34	24	9	1
	7	7	7	7	7	2	1	0	0	0
	0	0	0	0	0	8	17	28	43	51
0.003	43	43	43	43	43	42	34	24	9	1
	9	9	9	9	9	9	3	1	0	0
	0	0	0	0	0	1	15	27	43	51
0.002	38	38	38	38	38	38	34	24	9	1
	14	14	14	14	14	14	10	7	1	0
	0	0	0	0	0	0	8	21	42	51
0.001	25	25	25	25	25	25	25	24	9	1
	27	27	27	27	27	27	27	27	10	1
	0	0	0	0	0	0	0	1	33	50

Notes : First column : Number of fiber elements fractured by σ_L, Second column : Number of fiber elements fractured by $\sigma_\perp$, Third column : Number of fiber elements fractured by τ, $\sigma_\perp$: Normal stress which work on the sloping plane, σ_L: Normal stress which work on the perpencdicular plane to the sloping plane, τ: Shear stress which work on the sloping grain, $\sigma_{\perp max}$: Strength against normal stress which work on the sloping plane, τ_{max}: Strength against shear stress which work on the sloping grain.

Table 5.3. Results of simple regression analysis between specific gravity and MOR/σ_t.

$\sigma_{\perp max}$	τ_{max} 1	0.1	0.08	0.07	0.06	0.05	0.04	0.03	0.02	0.01
1	4071	4071	3999	3938	3918	3947	4119	4670	6687	13470
	-1152	-1152	-1108	-1066	-1034	-1001	-998.9	-1085	-1629	-3372
	0.883	0.883	0.879	0.864	0.81	0.682	0.528	0.394	0.336	0.33
0.01	4046	4046	3999	3938	3918	3947	4119	4670	6687	13470
	-1137	-1137	-1108	-1066	-1034	-1001	-998.9	-1085	-1629	-3372
	0.881	0.881	0.879	0.864	0.81	0.682	0.528	0.394	0.336	0.33
0.005	3544	3544	3544	3544	3627	3809	4119	4670	6687	13470
	-822.9	-822.9	-822.9	-822.9	-858.8	-919.9	-998.9	-1085	-1629	-3372
	0.743	0.743	0.743	0.743	0.73	0.654	0.528	0.394	0.336	0.33
0.004	3395	3395	3395	3395	3395	3466	3947	4670	6687	13470
	-702.3	-702.3	-702.3	-702.3	-702.3	-718.9	-897.4	-1085	-1629	-3372
	0.603	0.603	0.603	0.603	0.603	0.579	0.5	0.394	0.336	0.33
0.003	3143	3143	3143	3143	3143	3153	3286	4441	6687	13470
	-479	-479	-479	-479	-479	-483.4	-516.6	-949.7	-1629	-3372
	0.395	0.395	0.395	0.395	0.395	0.396	0.402	0.371	0.336	0.33
0.002	2611	2611	2611	2611	2611	2611	2712	3338	6344	13470
	5.56	5.56	5.56	5.56	5.56	5.56	-40.8	-281.3	-1426	-3372
	0.189	0.189	0.189	0.189	0.189	0.189	0.197	0.24	0.315	0.33
0.001	2443	2443	2443	2443	2443	2443	2443	2450	3818	12780
	849	849	849	849	849	849	849	846.6	314.9	-2966
	0.074	0.074	0.074	0.074	0.074	0.074	0.074	0.0743	0.12	0.31

Notes : First column : Slope coefficient, Second column : Intercept, Third column : Correlation coefficient, σ_t: Strength reduction ratio estimated by simulation, MOR: Modulus of rupture in bending test, $\sigma_{\perp max}$: Strength against normal stress which work on the sloping plane, τ_{max}: Strength against shear stress which work on the sloping grain.

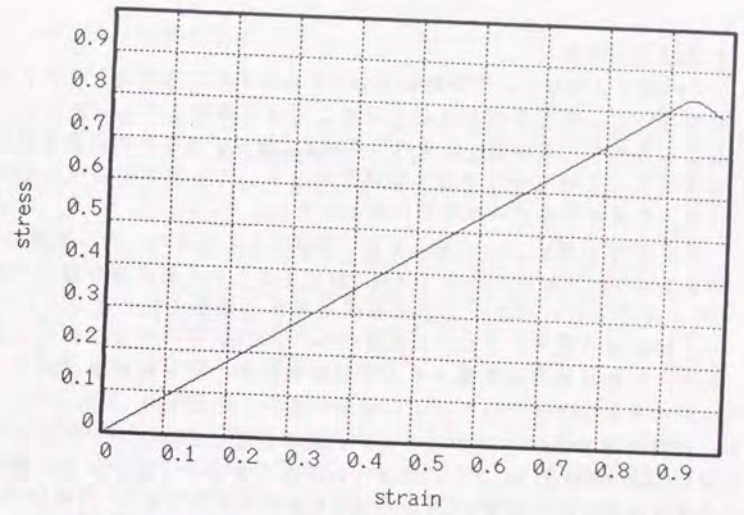

Fig.5. 7. Simulated stress-strain diagram. (No.31, condition $\sigma_\perp = 1, \tau = 0.06$)

Fig.5. 8. Simulated stress-strain diagram. (No.32, condition $\sigma_\perp = 0.005, \tau = 0.06$)

5.3.2.3 考察

3.5節で、曲げヤング係数は材軸とL軸のずれの角度の大きさとは関係が無いことが明らかにされている。引張り破壊シミュレーションで推定されたヤング係数は断面内の材軸とL軸のずれの角度の平均値に相当するものであるからやはり相関性はなく、この値で曲げヤング係数を割った値と比重との相関性は低下してしまう。

次に破壊形態について見てみる。Table 5.2を見ると、せん断強度τ_{max}の値をL方向の強度の8%から6%に設定するとせん断破壊が起こり始め、3%に設定すると半数以上の試験体が専断で破壊を起こすことになる。せん断強度の値としては、L方向の8% から6%というのは妥当な値であり、実際の引張り試験でもこの程度の割合でせん断破壊が起こっていると考えて良いだろう。

目切れ面の垂直方向の応力による破壊が起こるのは、その強度$\sigma_{\perp max}$をL方向の強度の1%以下に設定した場合である。一般に木材の横引張り強度は縦引張り強度の5%から10%程度の大きさであり、1%というのは小さすぎる。したがって、今回想定したような特に大きな繊維走向の乱れが存在しないような材の破壊では目切れ面に対する垂直応力による破壊は生じないと考えられる。

実際にシミュレーションから得られた引張り強度低減率の推定値σ_tであるが、Table 5.3を見るとわかるように、MOR/σ_tと比重の相関が最大になるのは全試験体がL方向の引張り応力で破壊した場合であるが、その時の相関係数も実験で得られた比重とMORの相関係数よりも低く、繊維傾斜の存在による強度の低減を表わしているとは言い難い結果であった。

5.3.3 曲げ破壊モデル

5.3.3.1 モデル化

この節では、繊維モデルを用いて実際に木取りを測定した木材のモデル化を行い、その曲げ強度の推定を行なった。対象とした材は前節と同じカラマツのラミナである。以下に仮定条件とモデル化の方法を示す。

（仮定１） この仮定は引張り破壊モデル時と同じである。

（仮定２） この仮定は引張り破壊モデル時と同じである。

（仮定３） この仮定は引張り破壊モデルと同じである。

（仮定４） この仮定は引張り破壊モデルと同じである。

（仮定５） 繊維要素の引張り破壊については、その破壊条件式は式(5.6)で引張り破壊モデルの時と同じである。しかし繊維要素に圧縮力が作用した場合には別の考えをする必要がある。第i繊維要素の目切れ面に作用するせん断力τ_iについては引張り破壊の時と同じであるが、σ_{Li}に対する強度は引張りと圧縮とでは違ってくる。また、破壊後の挙動も引張り破壊の場合には、応力負担が0になるという仮定で構わないが、圧縮の場合には塑性変形を無視することはできない。そこで、目切れ面平行方向及び垂直方向の圧縮強度は、その引張り強度の1/2であり、せん断強度は引張りと同じと仮定し、破壊後の応力負担はFig.5.9のようになるとする。

６） 繊維要素の破壊による応力の再配分と、そのくり返しにより破壊が進行する点は引張り破壊モデルと同じであるが、曲げの場合には応力が断面に一様に作用するわけではないのでその点を考慮に入れる必要がある。材にモーメントMが作用した時のひずみεの分布はFig.5.10に示したように、引張り側から圧縮側まで直線的に変化し、

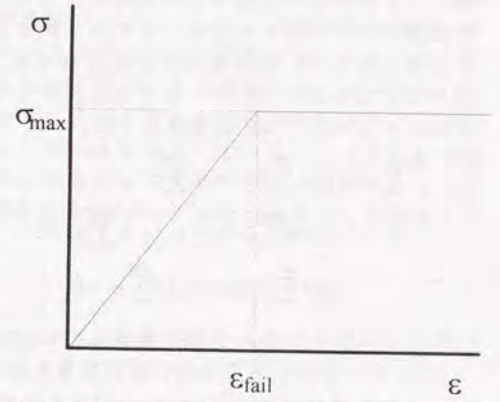

Fig.5.9. Stress-strain diagram of Fiber element when compression stress is acted.

中立軸で0になるとする。この時の傾きをkとすると、第i番目の繊維要素のひずみε_iは、

$$\varepsilon_i = k(w_i - w_n) \tag{5.10}$$

で与えられる。$k>0$であり、この時引張り側のひずみは負の値になる。

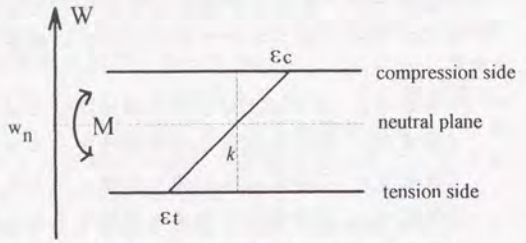

Fig.5.10. Schematic diagram of strain variation in cross section at moment acted.

曲げモーメントが作用した時に断面に作用する力の和は0になるので、式(5.10)及び、繊維要素のW座標w_iとヤング係数E_i、繊維要素の断面積A_0を用いると、

$$\sum^{n} A_0\sigma_i = A_0\sum^{n} E_i\varepsilon_i = kA_0\sum^{n} E_i(w_i - w_n) = 0 \tag{5.11}$$

n：繊維要素の総数

で与えられる。一般にはこの式から中立軸の位置w_nを求めることができ、そうすれば中立軸からの距離によって作用応力が決まってくるので繊維要素に作用する応力、ひずみを求めることができる。しかし5)で定めたように、繊維要素が圧縮破壊を起こした場合にはその応力負担が0にならないために、曲げ破壊過程では式(5.11)は成り立たなくなる。すなわち、曲げ破壊過程では、

$$\sum^{n} A_0\sigma_i = A_0\sum^{live} E_i\varepsilon_i + A_0\sum^{cFail}\sigma_i = kA_0\sum^{live} E_i(w_i - w_n) + A_0\sum^{cFail}\sigma_i = 0 \tag{5.12}$$
$$\Leftrightarrow k\sum^{live} E_i(w_i - w_n) + \sum^{cFail}\sigma_i = 0$$

live：破壊していない繊維要素の数

cfail：圧縮破壊を起こした繊維要素の数

という関係が成り立つことになる。したがってこの式からだけでは中立軸の位置を求めることはできない。

材に作用する曲げモーメントMと繊維要素断面に作用する応力との間には、

$$M = \sum A_0\sigma_i(w_i - w_n) \tag{5.13}$$

という関係が成り立つ。式(5.12)と同様に考えると、

$$M = A_0\sum^{live}\sigma_i(w_i - w_n) + A_0\sum^{cfail}\sigma_{ij}(w_i - w_n) = kA_0\sum^{live} E_i(w_i - w_n)^2 + A_0\sum^{cfail}\sigma_i(w_i - w_n) \tag{5.14}$$

となる。したがって式(5.12),(5.14)よりw_nは、

$$w_n = \frac{\sum^{live} E_iw_i \times \left(A_0\sum^{cfail}\sigma_iw_i - M\right) - A_0\sum^{live} E_iw_i^2 \times \sum^{cfail}\sigma_i}{\sum^{live} E_i \times \left(A_0\sum^{cfail}\sigma_iw_i - M\right) - A_0\sum^{live} E_iw_i \times \sum^{cfail}\sigma_i} \tag{5.15}$$

から求めることができ、この値を式(5.14)の代入すればkを求めることができる。したがって、式(5.10)を用いてモーメントMが作用した時の各繊維要素のひずみを求めることができる。

圧縮強度の方が引張り強度よりも小さいので、破壊の進行は、まず圧縮側で繊維要素が破壊し、その影響で中立軸が引張り側にずれていき、最後に引張り側が破断して最終的な破壊が起こる。

以上の仮定に従い、シミュレーションを実行してその強度の推定を行なった。ただし、引張り破壊シミュレーションの場合には弱い繊維を順番に追っていきその時のみかけのヤング係数を求めて応力のピークを求める形で強度推定を行なったが、曲げ破壊シミュレーションでは中立軸が破壊の進行に伴って移動するため同じ方法を取ることができないので、作用モーメントMの値を増やしていき、すべての繊維要素が破壊した時のモーメントを破壊モーメントとし、材のみかけの断面係数を用いて計算して得られたMORをその材の強度として扱うことにした。

5.3.3.2 強度推定結果

Table 5.4に示したような$\sigma_{\perp max}$, τ_{max}の値の組み合わせを用いて引張り破壊シミュレーションを行い、各試験体の曲げ強度推定を行なった。

シミュレーションの結果得られたヤング係数の低減率で、MOEを割った値と比重との相関をFig.5.11, 12に示した。Fig.5.11は、作用応力の向きがL軸と平行であるとした場合、すなわち繊維要素の配向方向がL

軸と平行である場合のヤング係数の低減率を、Fig.5.12は作用応力の向きが材の木口断面に対して垂直であるとした場合の結果である。

強度については Table 5.4, 5 に示した。Table 5.4 は破壊形態ごとに、その条件下でその破壊形態で破壊した試験体の総数を示している。また、実験で得られた曲げ破壊強度を、各条件下で推定された強度低減率で割った値と比重との相関分析を行なった結果を示している。なお、ここで得られた強度低減率は、応力の作用方向がL軸に対して平行になるように繊維要素を配向させた場合の値である。

5.3.3.3 考察

Fig.5.11,12を見ると、3.5節で得られた結果と同様に、応力の作用方向がL軸と平行と仮定した場合のヤング率低減率を用いた方が、比重との相関性は高くなっているが、曲げヤング係数の実験結果と比重の関係をそのままy切片周りに回転させたような結果になっており、繊維傾斜と曲げヤング係数の関係という観点からすると、明確な関係はない。

次に破壊形態について見てみる。Table 5.4 を見ると、せん断強度 τ_{max} の値がL方向の強度の2%あたりからせん断破壊が起こり始めている。目切れ面の垂直方向の応力による破壊は、$\sigma_{\perp max}$ をL方向の強度の0.05%以下に設定しないと発生しない。なぜこのようになってしまったかというと、繊維要素をL軸と平行になるように配向したので、結果としてL軸と繊維要素軸がなす角度が小さくなり引張り破壊時に比べてL方向の応力以外の破壊が起こりにくくなってしまったからである。しかし実際に破壊試験を行った際には、明らかに目切れ部分から破壊を生じた試験体が複数観察された。モデルでは破壊条件を単純にするために曲げモーメントを繊維要素に作用する垂直応力に置き換えて考えたが、実際には目切れの部分には応力集中が起こるため破壊条件式で与えられる応力よりもはるかに大きな応力が作用する。その大きさを計算で求めることは困難なので、強度の方を通常より低めに設定して応力集中の効果を取り入れるべきであると考えられる。

実際にシミュレーションから得られた引張り強度低減率の推定値 σ_b であるが、Table 5.5 を見るとわかるように、MOR/σ_b と比重の相関が最大になるのは全試験体がL方向の引張り応力で破壊した場合であるが、その時の相関係数も実験で得られた比重とMORの相関係数と同程度であり、繊維傾斜の存在による強度の低減を表わしているとは言い難い結果であった。

Table 5.4. Strength combination of fiber element and totals of bending simulated species fractured by each failure mode.

$\sigma_{\perp max}$ \ τ_{max}	1.0	0.03	0.025	0.02	0.015	0.01	0.005
1.0	52	52	51	46	38	26	10
	0	0	0	0	0	0	0
	0	0	1	6	14	26	42
0.001	52	52	51	46	38	26	10
	0	0	0	0	0	0	0
	0	0	1	6	14	26	42
0.00075	51	51	52	46	38	26	10
	1	1	0	0	0	0	0
	0	0	0	6	14	26	42
0.0005	50	50	50	48	38	26	10
	2	2	2	2	0	0	0
	0	0	0	2	14	26	42
0.00025	39	39	39	39	39	28	10
	13	13	13	13	13	1	0
	0	0	0	0	0	23	42
	27	27	27	27	27	27	27	10	1
	0	0	0	0	0	0	1	33	50

Notes : First column : Number of fiber elements fractured by σ_L, Second column : Number of fiber elements fractured by $\sigma_\perp$, Third column : Number of fiber elements fractured by τ, $\sigma_\perp$: Normal stress which work on the sloping plane, σ_L: Normal stress which work on the perpendicular plane to the sloping plane, τ: Shear stress which work on the sloping grain, $\sigma_{\perp max}$: Strength against normal stress which work on the sloping plane, τ_{max}: Strength against shear stress which work on the sloping grain.

Table 5.5. Results of simple regression analysis between specific gravity and MOR/σ_t.

$\sigma_{\perp max}$	τ_{max} 1.0	0.03	0.025	0.02	0.015	0.01	0.005
1.0	4604	4624	4670	4832	5154	5885	9175
	-1319	-1323	-1336	-1395	-1474	-1603	-2301
	0.886	0.883	0.866	0.811	0.684	0.477	0.327
0.001	4533	4553	4592	4791	5134	5886	9175
	-1267	-1277	-1289	-1365	-1461	-1603	-2301
	0.877	0.876	0.862	0.812	0.683	0.477	0.327
0.00075	4552	4552	4559	4723	5069	5908	9175
	-1262	-1262	-1263	-1321	-1420	-1616	-2301
	0.852	0.852	0.847	0.804	0.677	0.479	0.327
0.0005	4671	4671	4671	4659	4937	5802	9194
	-1274	-1274	-1274	-1260	-1333	-1553	-2312
	0.747	0.747	0.747	0.736	0.654	0.471	0.328
0.00025	5069	5069	5069	5069	5045	5531	9041
	-1241	-1241	-1241	-1241	-1224	-1325	-2219
	0.461	0.461	0.461	0.461	0.458	0.41	0.323

Notes : First column : Slope coefficient, Second column : Intercept, Third column : Correlation coefficient, σ_t: Strength reduction ratio estimated by simulation, MOR: Modulus of rupture in bending test, $\sigma_{\perp max}$: Strength against normal stress which work on the sloping plane, τ_{max}: Strength against shear stress which work on the sloping grain.

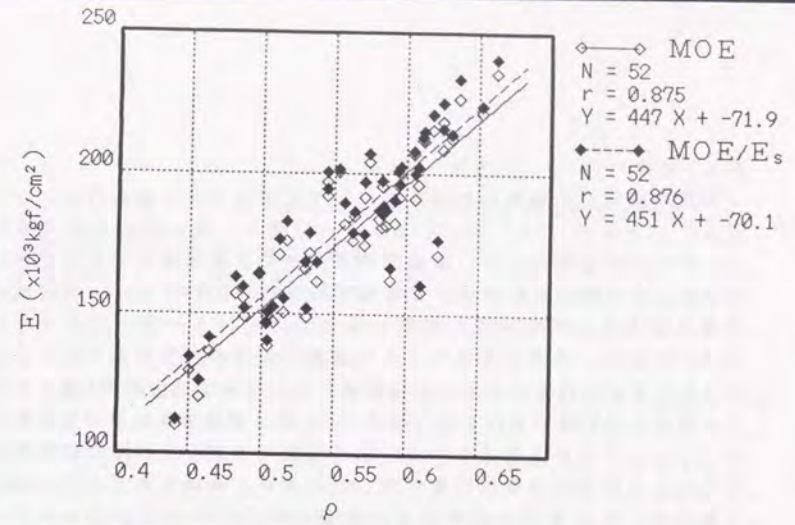

Fig.5.11. Relationship between ρ and MOE/E_S on the assumption that stress works parallel with L-direction.

Notes : ρ: Specific gravity, MOE: Modulus of elasticity in bending test, E_S: Ratio of E-reduction estimated by simulation .

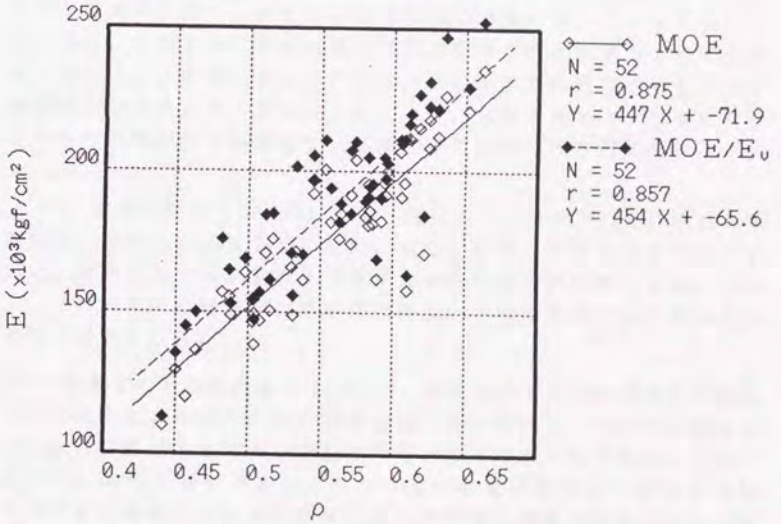

Fig.5.12. Relationship between ρ and MOE/E_V on the assumption that stress works perpendicular to cross section.

Notes : ρ: Specific gravity, MOE: Modulus of elasticity in bending test, E_V: Ratio of E-reduction estimated by simulation.

5.4 結論

実際に測定した繊維傾斜をもとにして木材のモデル化を行い、コンピュータシミュレーションによりその引張り、曲げ強度の推定を行なった。その結果からは、繊維傾斜と強度の低減を導くことはできなかった。その理由としては、繊維要素の物性を決めるのはその繊維傾斜だけであり、その結果、材内の強度のばら付きがほとんど生じなかったために、強度は最初の一本の繊維の破壊強度で決まってしまったことが挙げられる。もう一つの理由としてはヤング係数のL軸とのずれに依存する低減を表わす式の問題で、L軸と繊維要素の軸がなす角度をパラメータとして用いたために曲げ破壊のモデル化の際には繊維要素のヤング係数がほぼ同じ値になってしまい、そのためさらに、強度は最初の一本の繊維の破壊強度の影響を強く受けることになってしまった。今後の課題としては、平均年輪幅、早晩材の配置、繊維要素の軸方向強度のばらつき等を考慮に入れたモデル化を行なうことと、繊維傾斜と弾性率の関係を表わす適切な数式を導入することが必要になるであろう。

第6章　結言

本論文で得られた結果から以下のような結論が導きだされる。

木理の数値化と木材の物性の評価

1）ヤング係数と木理の関係については明確な相関関係を見出すことはできなかったが、曲げモーメント作用時に断面に作用する応力の方向についての知見が得られた。

2）乾燥変形については、厚さ収縮率については明確な関係は見出せなかったが、幅反りについては単純なパラメータで評価することが可能であった。弓反り、縦反りについても明確な関係は見出せなかったものの、元の樹幹の状態(完満度)に依存している可能性が示された。今後、木理パラメータを用いて乾燥による狂いを、材の3次元的な変形挙動の2次元断面として捉えれば、さらに精度の高い狂いの予測が可能になると考えられる。

3）応力波については、繊維方向とのなす角度が大きくなるに従って、速度が低下する傾向が明確に認められた。また、伝播速度が最大になる角度の0度からのずれが旋回木理の角度を示していると考えられ、また、回帰式から求めた繊維走向の伝播速度と比重との相関性が高くなった。これらの例に示されたように、木理の数値化と応力波伝播速度を組み合わせて用いることにより、伝播速度をヤング率に代わる木材の材質評価の指標値として用いることができる可能性が示された。

4）木理の数値化法については、現在のところ両木口面に現れた年輪の違いだけしか利用していない。板目、柾目に現れた木理を利用すれば、材の曲がり等の影響を考慮に入れた数値化が可能になる。しかし、そのためには簡便で比較的精度の高い木理の測定方法を開発する必要が認められる。

5）本論文の木理の数値化法により、材全体の平均的な物性の評価は可能であった。しかし、局所的な物性、特に節によって生じる繊維走向の乱れに起因する材質の評価はこの方法では不可能である。しかしながら、節の影響を考慮に入れない形で材質評価を行うことにより、その材質の説明がつかない部分は節、その他の傷害の影響であると限定することができる。そのような形で用いることにより有効に活用することができると考えられる。

確率モデルによる単板積層材の強度予測

1） 37ﾌﾟﾗｲで接着長さがある範囲内にあるという限定された条件下では、接着せん断強度が接着長さによって変わらないという単純な仮定に基づいて、比較的簡単な実験の結果に統計処理を加えて鎖モデルを適用することによってほぼ完全にその破壊強度を予測することができた。

2） 接着長さの範囲が広くなると接着層に作用する応力を解析することが必要になってくる。しかし、実験結果により得られた接着長さと接着強度の分布を回帰した式を適用することにより、強度予測を行うことが可能となった。

3） 57ﾌﾟﾗｲ、97ﾌﾟﾗｲとﾌﾟﾗｲ数を増やしていくと、破壊源の数が増加するために37ﾌﾟﾗｲの実験データを基にした鎖モデルでは説明しきれなくなる。

以上より、複数であるが小数の破壊源を持つ物体の破壊については、その個々の破壊源の破壊機構がわからなくても統計的に処理したデータに基づいて鎖モデルを適用することによりその強度を予測することが可能であることが示された。

シミュレーションによる木材の破壊強度の予測

実際に測定した繊維傾斜をもとにして木材のモデル化を行い、コンピュータシミュレーションによりその引張り、曲げ強度の推定を行なった。その結果からは、繊維傾斜と強度の低減を導くことはできなかった。今後の課題としては、平均年輪幅、早晩材の配置、繊維要素の軸方向強度のばらつき等を考慮に入れたモデル化を行なうことと、繊維傾斜と弾性率の関係を表わす適切な数式を導入することが必要になるであろう。

結語

木理の数値化を行うことによって、今までは異方性の評価を行なうために特別に作成された試験体でしか確認されていなかった狂い、応力波の伝播速度に対する異方性の影響を、任意の木取りの材で確認することが可能になった。また、確率モデルを用いれば複数の破壊機構が不明な破壊源を持つ材料の強度を推定することが可能であることが示された。この二つを組み合わせた木材の強度推定を行ったが、その結果は明確な予測を行なうことはできなかったが、木理、もしくは木理と他の物性を組み合わせてを木材の材質指標のパラメータとして使用できる可能性を示唆する結果が得られた。

謝辞

本論文をまとめるに当たり多大なるご指導を賜った大熊幹章教授と有馬孝禮助教授に心から感謝の意を表します。大熊幹章教授には学部学生の頃より教わるところが多く、本論文の作成に際しても多数の貴重な御助言を戴きました。有馬孝禮助教授には本研究を遂行するにあたり直接ご指導を賜り、学部学生の頃より研究活動を通し幅広い御教示を賜りました。ここで改めて大熊・有馬両博士に感謝し、御礼申し上げたいと思います。信田聡助手には本論文の作成のみならず大学院において研究を進めるにあたって一方ならないお世話になりました。ここに厚く御礼申し上げます。

また、現東大農学部付属演習林助手の中村昇前助手には直接様々な御指導、御教示を賜りました。ここに厚く御礼申し上げます。

また、木質材料学教室の卒業生で、現在建設省建築研究所の中島史朗氏、三井ホームの名波直道両氏、及び博士課程3年生の渋澤龍也氏には、学部学生の頃より具体的な実験の遂行の方法、論文のまとめ方等様々な点に渡って示唆を受けることが多く、現在のテーマを手懸けるようになったのも三氏の影響による所が多く、ここで改めて厚く御礼申し上げます。

さらに、日頃の研究と論文作成にあたりいろいろと手助けをして下さった木質材料学教室の樋本敬大氏、佐久間博文氏、小田祐二氏、高林雅人氏、小池真理氏、西村拓也氏を始めとする大学院生の皆様と、卒業生で現木下工務店勤務の園田里見氏、横浜ゴム勤務の菊地輝彦氏、そして、コンピュータプログラミングについて多くの示唆を与えてくれたNEC勤務の四宮潔氏に心より感謝し、御礼申し上げたいと思います。

最後に、6年間楽しく、気ままに、厳しい研究生活を送ることができる環境を与えて下さった大熊・有馬両博士に、改めて厚く御礼申し上げます。

1993年12月　　岡崎泰男

参考文献

1) Rudolf E.Booker : "A method for recording annual ring orientation in boards", FPJ, **37**(6), 31-33, (1987).

2) Joseph G. Massery; James E. Reeb : "A method for estimating juvenile wood content in boards", FPJ, **39**(2), 30-32, (1989).

3) James R. Olson : Measurement of growth ring orientation in lumber, FPJ, **36**(3), 23-24, (1986).

4) Francis G.Wagner; Philip H. Steele; Lalit Kumar; Dorde Butkovic: Computer grading of southern pine lumber, FPJ, **41**(2), 27-29, (1991).

5) 栃木紀郎:「木材の加工面における木理パターンの予測」, 木材工業, **39**(6), 261-264, (1981).

6) Greenhill, W.L., Austr. Coun. Sci. Ind. Res., Div. of For. Prod., Tech. Paper **35**, Pamphlet 97, (1940).

7) Keylwerth, R. : Holz als Roh-u. Werkstoff, **8**, 253, (1950).

8) 山田正,梶田茂 : 木材研究, **9**, 42, (1952).

9) 井坂三郎 : 木材の狂いについて、材料、**12**、19-22,(1963).

10) Jenkin, C.F. : Report on Mathematical Used in the Construction of AirCraft and Air Engines, H.M. Stationery Office, London, (1920).

11) Keylwerth, R. : VDI-Forschungsheft 430. Deutscher Ingenieur-Verlag, (1951).

12) 澤田　稔 : 材料, **32**(8), 838-847 (1983).

13) Kollman, F., : ForstWiss. Cbl., **56**(6), 181, (1934).

14) Kollman, F., & Cote, W.A. : "Principles of Wood Science and Technology I", 321-419, Spriger-Verlag, (1968).

15) 矢沢亀吉 : 岐阜農学報告, **19**, (1958).

16) Baumann, R., Die bisherigen Ergebnisse der Holzprufugen in der Materialprufungsanstalt an der Tech. Hochshule Stuttgart, Forsh. Gebiete Ingenicufw., H.231, Berlin, (1922).

17) Wilson, T.R.C., J.Forestry, **19**, 747, (1921).

18) 畑山嶬男: 有節材の強度推定に関する研究, 林試研究報告, **326**, 69-197, (1983).

19) Elvery, R.H.; and Nwokoye, D.N.: "Strength assessment of timber for glued laminated beams. Paper Ⅱ, Symp. Non-destructive Testing of Concrete and Timber", Organized by the Institution of Civil Engineering and the British Commission for Nonrestrictive Testing. June 11-12, 1969. Inst. of Civil Eng., London, 105-110. (1970).

20) Galligan, W.L.; and Pellerin, R.F. : "Dynamic tests for nondestructive analysis of lumber quality", 5th Internal. Conf. on Nondestructive Testing. May 21-26. Montreal, Quebec, Canada. 15-20. (1967).

21) Lee, I.D.G.: "A nondestructive method for measuring the elastic anisotropy of wood using ultrasonic pulse technique", J. Inst. Wood Sci. 1, 43-57. (1958).

22) 名波直道:「応力波による立木の材質測定」,東京大学学位論文, 39-50,(1992).

23) Gerhards, C.C.: "Effect of cross grain on stress waves in lumber", USDA Forest Serv. Res. Pap. FPL368, Forest Prod. Lab. , Madison, Wis. 1981.

24) 鈴木弘志,佐々木栄一 :「木材の超音波伝播速度に及ぼす繊維傾斜角の影響」,木材学会誌, **36**(2), 103-107 (1990).

25) 大熊幹章他:「木材の非破壊検査方法の検討と強度等級区分システムの確率に関する研究」, 昭和63年、平成元年度科学研究費補助金（総合研究A）研究成果報告書, (1990).

26) S.E.Taylor, D.A.Bender : "Simulating Correlated Lumber Properties Using a Modified Multivariate Normal Approach", ASAE,**31**(1),P182-186 (1988).

27) S.E.Taylor, D.A.Bender : "A method for simulating multiple correlated lumber properties", FPJ, **39**(7,8), 71-74 (1989).

28) Y.H.Chui, I.Smith : "The use of bending and shear moduli for predicting bending strength of wood", FPJ, **41**(4), 49-52 (1991).

29) 有馬孝禮, 丸山則義他３名: 材料, **42**(473), 141-146 (1993).

30) 荒武志朗, 有馬孝禮, 迫田忠芳, 中村徳孫 : 木材学会誌, **38**(11), 995-1001 (1992).

31) 古沢　信, 平野　茂 : 日本加工技術協会第10回記念年次大会講演要旨集, 東京, 49-50 (1992).

32) F.Pellerin: "A vibration approach to nondestructive testing of structural lumber", For. Prod. J., **15**(3), 93-100 (1965).

33) G.G.Marra, R.F.Pellerin, W.L.Galligan: "Nondestructive determination of wood strength and elasticity by vibration", Holz als Roh-Werkst., **24**(10), 460-466 (1966).

34) D.G.Miller: "Nondestructive testing of joists by a vibrational technique", For. Prod, J., **18**(2), 25-28 (1968).

35) 祖父江信夫: "木材の打音のFFT分析による弾性定数の瞬間測定、梁の曲げ振動への適用", 木材学会誌, **32**(4), 274-279 (1986).

36) 児玉泰義 : "音速による変断面形状を有する木材のヤング係数推定方法（第１報）、丸太材への適用性", 木材学会誌, **36**(11), 997-1003 (1990).

37) 小玉泰義, 秋鹿為之 : "超音波のパルス反射法を用いた木材の探傷（第１報）、隠れ節の計測" : 木材学会誌, **39**(1), 7-12 (1993).

38) Sato K. Fusitani M.: "Application of Acoustic Emission ot Stress Grading of Timber", Progress in Acoustic Emission IV, Japan Society for Non-Destructive Inspection, Kobe, 657-663, (1988).

39) 中井　孝 : 木材工業, **41**(10), 455-459 (1986).

40) 祖父江信夫: "パソコンによる実大木材のヤング率の自動測定", 木材工業, **42**(9), 415-417 (1987).

41) 神谷文夫 : 木材工業, **40**(12), 581-585 (1985).

42) 増田　稔, 小池寿典 :「有限小領域非線形破壊クライテリオンの提案」, 日本木材学会大会（盛岡）, 研究発表要旨 P309 (1993).

43) Steven M. Cramer, James R. Coodman : "Failure Modeling: A Basis for Strength Prediction of Lumber", Wood and Fiber Science, **18**(3), 446-459 (1986).
44) F.T.Perice : J.Tex. Inst., **17**, 355, (1926).
45) W.Weibull : Ing. Vetenkaps Akad. Handl., **151** (1939).
46) 林知行：「モンテカルロシミュレーションと強度研究」,木材工業, **45**(8), 353-358 (1990).
47) 市川昌弘："信頼性工学", 裳華房, 3-4, (1990).
48) M.L.Moeschberger and H.A.David, Biometrics, **27**, 909 (1971).
49) 横堀武夫："材料強度学", 岩波書店 (1974).
50) 松尾陽太郎ほか：「多重モードワイブル分布のパラメータ推定における多段相関係数法と多段最尤法との比較」, 材料, **34**, P1466 (1985).
51) 松尾陽太郎ほか:「アルミナ曲げ強度に及ぼす研削加工の影響と多重モードワイブル分布」, 材料, **36**, P166 (1987).
52) 松尾陽太郎ほか：「複数の破壊原因を考慮した非線形弾性対における破壊位置の分布関数の導出とその応用」, 材料, **39**, P138 (1990).
53) 金原勇：「一方向ハイブリッド繊維強化材料の引張り破壊過程のシミュレーション」, 材料, **34**, P280 (1985).
54) 鈴木寛;関根英樹:「一方向ＦＲＰ複合材料中の繊維の剥離及び引き抜けに関する確率論的研究」,材料, **38**, P106 (1989).
55) 酒井達雄；菊池俊郎；藤沢泰成：「十字形引張り溶接継手の疲労破壊に関する統計的一研究」, 材料, **38**, P287 (1989).
56) 朴鐘宝；又木義博：「パーティクルボードの確率疲労特性」, 木材学会誌, **35**, P609 (1989).
57) 林　知行：「構造用ＬＶＬの疲労特性」, 木材学会誌, **35**, P616 (1989).
58) 林　知行：「確率モデルによる修正加工材料の性能予測（第１報）」,木材学会誌, **35**, P1048 (1989).
59) 小松幸平：「鋼板添板釘打ち接合の変形と耐力（第２報）ﾓﾝﾃｶﾙﾛ法による許容せん断耐力の評価」, 木材学会誌, **36**, P1042 (1990).
60) 川合慧：" 基礎グラッフィックス"、昭晃堂、68-72、(1985)
61) 伏谷賢美他："木材の物理", 文永堂, 61-62, (1985).
62) 伏見正則："乱数", 東京大学出版会 (1989).
63) Adams,R.D. : J.Adhesion, 30, 219-242 (1989).
64) 山口幸三郎監修："接着・粘着の辞典", 朝倉書店, 1986, p.290-291.
65) Adams,R.D.; Atkins,R.W.; Harris,J.A; Kinloch,A.J.: J.Adhesion, "Stress Analysis and Failure Properties of Carbon-Fiber-Reinforced-Plastic/Steel Double-Lap Joints", 20(1),29-53(1986).
66) 杉林俊雄,池上皓三：日本接着協会誌, 18(3), 102-109(1982).
67) F.Szepe : Exp.Mech., 6, 282 (1966)
68) O.Volkersen: Luftfahrforshung, 15-1/2, 41 (1939).

69) M.Coland and Reissner: Journal and Applied Mechanics Trans. ASME, A-17 (1944).
70) 山口章三郎,古川光二,天野晋武ら：「せん断接着強さの測定法について」, 日本接着協会誌, 15(6), 225-232 (1979).
71) 牧野鉄治・野中保雄："信頼性工学", 日科技連 (1983)

Appendix

1.木理の数値化用プログラムのソースリスト

(1) GRAIN.H 木理数値化に必要なクラスの定義

```
# ifndef __GRAIN__
    # define __GRAIN__

# include   "trans.h"
# include   <lib¥matrix.h>
# include   <lib¥mystd.h>
# include   <math.h>

class   CornClass;

class   CrossSectionParamClass {
/*
      対象板材中の任意の木口面の樹軸に対する位置を示すパラメータ、それを基にして木口面の数値化を行なう関数をメンバとして持つクラス。
      ﾒﾝﾊﾞ関数定義  trunk.cpp
*/
public :
    double  ring_width;                 // その断面の平均年輪幅
    double  radius_of_1;    // その断面に一番目にあらわれる年輪の半径
    double  centroid_radius;// 原点から材中心までの距離
    double  angle_with_respect_to_Xaxis;
                                        // 原点と材中心を結んだ直線が X軸となす角度
    double  xCentroid;
    double  yCentroid;
    double  zCentroid;          // 樹幹原点から見た材中心座標

    double  tangentialRatio( double lumber_width ); //  板目材率を返す。
    voidoperator>>( char *string )
    {
        sprintf( string, "%8.3lg%8.3lg%8.3lg%8.3lg%8.4lg", xCentroid, yCentroid
                    , zCentroid, centroid_radius, angle_with_respect_to_Xaxis );
    }
    voidoperator>>( FILE* fp )
    {
        fprintf( fp, "%8.3lg%8.3lg%8.3lg%8.3lg%8.4lg", xCentroid, yCentroid
                    , zCentroid, centroid_radius, angle_with_respect_to_Xaxis );
    }
};

class   LumberParamClass : public CrossSectionParamClass {
/*
      対象板材（そりゃ柱でもいいんだが）の寸法と、樹軸に対する角度と、それらを
    基にして木理の数値化を行なう関数をメンバとして持つクラス。
      ﾒﾝﾊﾞ関数定義  trunk.cpp
*/
protected :
    MatrixClass *transMat_to_XYZ;

public :
    double  width;
    double  height;
    double  length;
    double  rotation_angle_to_X;
    double  rotation_angle_to_W;
    double  yrange;

    LumberParamClass(){
        transMat_to_XYZ = new MatrixClass( 4,4 );
    }
    ~LumberParamClass(){
        delete  transMat_to_XYZ;
    }
};

class   TrunkParamClass : public LumberParamClass {
/*
```



```
      対象材から推定された樹幹のパラメータを持つクラス。
    メンバ関数定義  trunk.cpp
*/
protected :
    MatrixClass *transMat_to_UVW;

public :
    int     piece_number;               // 試験体番号、ただの
    double  ring_width_gradient;// yの変化に伴う、年輪幅の変化
    double  rCentroid_gradient; // yの変化に伴う、基準面の中心位置が所属する
                                        // 樹幹径の変化。その位置のtan(γ)に相当する

    TrunkParamClass(){
        transMat_to_UVW = new MatrixClass( 4,4 );
    }
    ~TrunkParamClass(){
        delete  transMat_to_UVW;
    }

    BOOLoperator<<( FILE* fp_GRN ); // *.GRN ファイルからTrunkParamCla

    // ssのメンバの値を読み込む
    BOOLoperator>>( FILE* fp_GRN );
    voidprint( FILE *fp )
    //
    // 得られたパラメータをASCII形式でファイルに出力する
    {
        fprintf( fp, "試験体番号%3ld¥n", piece_number );
        fprintf( fp, "寸法%37.11f×%5.11f×%5.11f¥n", width, height, length );
        fprintf( fp, "基準面中心位置r%25.3lg¥n", centroid_radius );
        fprintf( fp, "基準面中心位置α%25.4lg¥n", angle_with_respect_to_Xaxis);
        fprintf( fp, "基準面平均年輪幅(mm)          %10.3lg¥n", ring_width );
        fprintf( fp, "基準面の中心位置が通る樹幹のtanγ%9.4lg¥n"
                    , rCentroid_gradient );
        fprintf( fp, "yの変化に伴う平均年輪幅変化   %10.4lg¥n¥n"
                    , ring_width_gradient );

        fprintf( fp, "tanθx%35.4lg¥n", tan( rotation_angle_to_X ) );
        fprintf( fp, "tanθN%35.4lg¥n¥n¥n", tan( rotation_angle_to_W ) );
    }
    void getCrossSectionXYZ( double y, double *right_x
                    , double* left_x, double* back_z, double* face_z );
                // Y==yの木口面のＸＹＺ系の直線の方程式を返す。
    double  ringWidth( double y=0.0 ) const
    // Y==yの木口面の平均年輪幅を返す
    {
        return  ring_width + ring_width_gradient * y;
    }
    double  rCentroid( double y ) const
    // 基準面の中心位置を通る樹幹のY==yの位置での半径を返す。
    {
    return  centroid_radius + rCentroid_gradient * y;
    }
    double  radiusOffirstRing( double y );

    double  thinAngle( double r, double y = 0.0 ) const;

// UVW系への座標変換
    HCVectorClass   trans_to_UVW( const HCVectorClass& pointXYZ ) const
    {
        return  (HCVectorClass)( pointXYZ * (*transMat_to_UVW) );
    }
    HCVectorClass   trans_to_UVW( const CornClass& pointRAY ) const
    {
        return
    (HCVectorClass)( trans_to_XYZ( pointRAY )

    * (*transMat_to_UVW) );
    }
```

```
// XYZ系、さらに円柱座標系への変換
    HCVectorClass    trans_to_XYZ( const
HCVectorClass& pointUVW ) const
    {
        return  (HCVectorClass)( pointUVW *
(*transMat_to_XYZ) );
    }

    HCPointClass trans_to_XYZ( const
CornClass& pointRAY ) const;
    CornClass        trans_to_Corn( const
HCPointClass& pointXYZ ) const;

// TLR 系への座標変換行列の定義
    MatrixClass
    makeTransMat_to_TLR( const
HCPointClass& pointXYZ );
    MatrixClass
    makeTransMatTLR_to_XYZ( const
HCPointClass& pointXYZ );

// XYZ系からTLR系への座標変換の実行
// 始点とXYZ系のベクトルを引数として与える。
    HCVectorClass    transXYZ_to_TLR( const
HCPointClass& startXYZ
                                    ,
const HCVectorClass& vectXYZ )
    {
        MatrixClass      trans_mat =
makeTransMat_to_TLR( startXYZ );
        return  ( HCVectorClass)( vectXYZ *
trans_mat );
    }

// 始点と終点を引数として与え、その２点間
を結ぶベクトルのTLR表現を返す
    HCVectorClass    transXYZ_to_TLR2( const
HCPointClass& startXYZ
                                    ,
const HCPointClass& endXYZ )
    {
        HCVectorClass    vectXYZ = endXYZ -
startXYZ;
        return  transXYZ_to_TLR( startXYZ,
vectXYZ );
    }

// UVW系からTLR系への座標変換の実行
// 始点とUVW系のベクトルを引数として与える。
    HCVectorClass    transUVW_to_TLR( const
HCPointClass& startUVW
                                    ,
const HCVectorClass& vectUVW )
    {
        HCPointClass startXYZ =
trans_to_XYZ( startUVW );
        HCVectorClass    vectXYZ =
trans_to_XYZ( vectUVW );
        return  transXYZ_to_TLR( startXYZ,
vectXYZ );
    }

// 始点と終点を引数として与え、その２点間
を結ぶベクトルのTLR表現を返す
    HCVectorClass    transUVW_to_TLR2( const
HCPointClass& startUVW
                                    ,
const HCVectorClass& endUVW )
    {
        HCVectorClass    vectUVW = endUVW -
startUVW;
        return  transUVW_to_TLR( startUVW,
vectUVW );
    }

// TLR系からUVW系への座標変換の実行
// 始点（∈UVW）とTLR系のベクトルを引数と
して与える。
    HCVectorClass    transTLR_to_UVW( const
HCPointClass& startUVW
                                    ,
const HCVectorClass& vectTLR )
    {
        HCPointClass startXYZ =
trans_to_XYZ( startUVW );
        MatrixClass      trans_mat_toXYZ =
makeTransMatTLR_to_XYZ( startXYZ );
        HCVectorClass    vectXYZ
=( HCVectorClass)( vectTLR *
trans_mat_toXYZ );
        return  trans_to_UVW( vectXYZ );
    }

    CrossSectionParamClass
    getCrossSection( double m );
};
```



```
inline BOOL
TrunkParamClass::operator<<( FILE*
fp_GRN )
                    // *.GRN ファイルから
TrunkParamClassのメンバの値を読み込む
{
// TrunkParamClassの publicメンバ
    if ( fread( &piece_number,
sizeof( int ), 1, fp_GRN ) < 1 )
        return  FALSE;

    fread( &rCentroid_gradient,
sizeof( double ), 1, fp_GRN );
    fread( &ring_width,
sizeof( double ), 1, fp_GRN );
    fread( &ring_width_gradient,
sizeof( double ), 1, fp_GRN );

// LumberParamClassの publicメンバ
    fread( &width, sizeof( double ), 1,
fp_GRN );
    fread( &height, sizeof( double ), 1,
fp_GRN );
    fread( &length, sizeof( double ), 1,
fp_GRN );
    fread( &yrange, sizeof( double ), 1,
fp_GRN );
    fread( &rotation_angle_to_X,
sizeof( double ), 1, fp_GRN );
    fread( &rotation_angle_to_W,
sizeof( double ), 1, fp_GRN );

    *transMat_to_XYZ << fp_GRN;
    *transMat_to_UVW << fp_GRN;

// CrossSectionParamClassの publicメン
バ
    fread( &centroid_radius,
sizeof( double ), 1, fp_GRN );
    fread( &angle_with_respect_to_Xaxis,
sizeof( double ), 1, fp_GRN );
    fread( &radius_of_1,
sizeof( double ), 1, fp_GRN );
    fread( &xCentroid, sizeof( double ),
1, fp_GRN );
    fread( &yCentroid, sizeof( double ),
1, fp_GRN );
    fread( &zCentroid, sizeof( double ),
1, fp_GRN );

    return  TRUE;
}

inline BOOL
TrunkParamClass::operator>>( FILE
*fp_GRN )
{
// *.GRN ファイルへTrunkParamClassのメ
ンバの値を書き込む

// TrunkParamClassの publicメンバ
    fwrite( &piece_number,
sizeof( int ), 1, fp_GRN );
    fwrite( &rCentroid_gradient,
sizeof( double ), 1, fp_GRN );
    fwrite( &ring_width,
sizeof( double ), 1, fp_GRN );
    fwrite( &ring_width_gradient,
sizeof( double ), 1, fp_GRN );

// LumberParamClassの publicメンバ
    fwrite( &width, sizeof( double ), 1,
fp_GRN );
    fwrite( &height, sizeof( double ),
1, fp_GRN );
    fwrite( &length, sizeof( double ),
1, fp_GRN );
    fwrite( &yrange, sizeof( double ),
1, fp_GRN );
    fwrite( &rotation_angle_to_X,
sizeof( double ), 1, fp_GRN );
    fwrite( &rotation_angle_to_W,
sizeof( double ), 1, fp_GRN );

    *transMat_to_XYZ >> fp_GRN;
    *transMat_to_UVW >> fp_GRN;

// CrossSectionParamClassの publicメン
バ
    fwrite( &centroid_radius,
sizeof( double ), 1, fp_GRN );

    fwrite( &angle_with_respect_to_Xaxis,
sizeof( double ), 1, fp_GRN );
    fwrite( &radius_of_1,
sizeof( double), 1, fp_GRN );
    fwrite( &xCentroid,
sizeof( double ), 1, fp_GRN );
    fwrite( &yCentroid,
sizeof( double ), 1, fp_GRN );
```

```
        if ( fwrite( &zCentroid,
sizeof( double ), 1, fp_GRN ) < 1 )
            return  FALSE;

        return  TRUE;
    }

    inline double
    TrunkParamClass::thinAngle ( double
radius, double y ) const
    //
    {
    double  n = ( radius - rCentroid( y ) )
/ ringWidth( y );
        return  atan( rCentroid_gradient +
n * ring_width_gradient );
    }

    inline void
    TrunkParamClass::getCrossSectionXYZ( do
uble y, double *right_x
                        , double* left_x,
double* back_z, double* face_z )
    //  Y==yの木口面のＸＹＺ系の直線の方程
式を返す。厳密に言えば、Ｘ軸とＵ軸
    //  は完全に平行ではないのだから、直線
には傾きがあるはずなんだが。
    {
        HCPointClass pointUVW( width/2.0,
length/2.0, 0.0 );
        HCPointClass pointXYZ =
trans_to_XYZ( pointUVW );
        *right_x = pointXYZ.x1();

        pointUVW.assign( -width/2.0,
length/2.0, 0.0 );
        pointXYZ =
(HCPointClass)trans_to_XYZ( pointUVW );
        *left_x = pointXYZ.x1();

        pointUVW.assign( 0.0, length/2.0, -
height/2.0 );
        pointXYZ =
(HCPointClass)trans_to_XYZ( pointUVW );
        *back_z = pointXYZ.x3();

        pointUVW.assign( 0.0, length/2.0,
height/2.0 );
        pointXYZ =
(HCPointClass)trans_to_XYZ( pointUVW );
        *face_z = pointXYZ.x3();
    }


class   CornClass : public CylinderClass {
/*
    樹幹を円錐台として捉えた時の座標表現
を定義するクラス。
    円筒座標系に加えて、年輪番号とその位
置の細り角をパラメータとして持つ。

    末口と元口の推定が正しかった場合には、
細り比は負の値をとる。すなわち、軸
    に対して右回りの方向が正になるからθx、
θzと同じになって評価しやすい、かも
    しれない。

        tan γ(n)＝rc_gradient + n×
ring_width_gradient

                                r－rc(y)
        n（r,y）＝
                            ⊿r(y)

        rc（y）＝rCentroid(0)＋
r_gradient×y

        ⊿r（y）＝ring_width(0)＋
ring_width_gradient×y

    より、任意断面（y）の樹軸からｒだけ
離れた位置の細り角γが求まる
*/

public :
    double  thin_angle;             // 所
属樹幹の細り

    CornClass(){}
    CornClass( const TrunkParamClass& trunk,
double _r, double _alpha, double _y )
    {
        radius = _r;
        alpha = _alpha;
        y = _y;
        thin_angle = trunk.thinAngle( y );
    }
    voidassign( const TrunkParamClass
&trunk,double _r
```



```
        ,double _alpha, double _y )
    {
        radius = _r;
        alpha = _alpha;
        y = _y;
        double ring_number = ( radius -
trunk.rCentroid( y ) )

            / trunk.ringWidth(y);
        thin_angle = trunk.thinAngle( y );
    }
};

    inline HCPointClass
    TrunkParamClass::trans_to_XYZ( const
CornClass& pointRAY )  const
    {
        HCPointClass temp;
        temp.x1() = pointRAY.radius *
cos( pointRAY.alpha );
        temp.x2() = pointRAY.y;
        temp.x3() = pointRAY.radius *
sin( pointRAY.alpha );
        return  temp;
    }

    inline CornClass
    TrunkParamClass::trans_to_Corn(

    const HCPointClass& pointXYZ ) const
    {
        CornClass   temp;

        temp.radius = sqrt( pointXYZ.x1() *
pointXYZ.x1()

                                        +
pointXYZ.x3() * pointXYZ.x3() );
        temp.alpha = acos( pointXYZ.x1() /
temp.radius );
        temp.y = pointXYZ.x2();
        double  ring_number
                = ( temp.radius -
rCentroid( temp.y ) ) / ringWidth();
        temp.thin_angle =
atan( rCentroid_gradient

                                        +
ring_number * ring_width_gradient );
        return  temp;
    }
```

```
# endif
```

(3)GRAIN.CPP 年輪データを読み込んで木理パラメータを求める。

```
# include   "grain.h"
# include   <lib¥mydir.h>
# include   <lib¥mystd.h>
# include   <string.h>
# include   <stdio.h>
# include   <math.h>
# include   <limits.h>

class   InputSectionClass : public
CrossSectionParamClass {
/*
    .AROファイルから入力された木口面の測定
値を持つクラス。CrossSectionParamClas
    sの導出クラス
*/
private :
    double  total_anual_rings;

public :
    double  radius_of_n;
    void    operator<<( FILE *fp_ARO )
                        // 木口面のデータ
をAROファイルから読み込む
    {
        myfscanf( fp_ARO, "%lg",
&centroid_radius );
        myfscanf( fp_ARO, "%lg",
&angle_with_respect_to_Xaxis );
        angle_with_respect_to_Xaxis *=
M_PI/180.0;
        myfscanf( fp_ARO, "%lg",
&radius_of_1 );
        myfscanf( fp_ARO, "%lg",
&radius_of_n );
        myfscanf( fp_ARO, "%lg",
&total_anual_rings );
    };
    friend classTrunkAnalysisClass;
    voidanalyze();              // 木
口面と材の関係の解析を行なう
};
```

```
    voidInputSectionClass::analyze() //  木口面と材の関係の解析を行なう
    {
    //  木口面の原点のｘ座標、ｚ座標を求める。
        xCentroid = centroid_radius * cos( angle_with_respect_to_Xaxis );
        zCentroid = centroid_radius * sin( angle_with_respect_to_Xaxis );
    //  平均年輪幅を求める
    //      ここのところで半径はこのままでいいのであろうか？
        ring_width = ( radius_of_n - radius_of_1 ) / total_anual_rings;
    }


class   TrunkAnalysisClass : public TrunkParamClass {
/*
     *.AROファイルからデータを入力、解析して基底クラスTrunkParamClassにデータ
    を落とす。したがって、樹幹解析を行なう関数は全てこのクラスで定義される。
*/
private :
    int     dn_between_12;      //  末口、元口面に現れた年輪の番号の差
    InputSectionClass    firstSection;
    InputSectionClass    secondSection;

    double  cos_W;      //  あまり意味はないんだが、少しでも三角関数の演算を
    double  sin_W;      //  減らすために定義した。
    double  cos_X;
    double  sin_X;

public :
    TrunkAnalysisClass(){   dn_between_12 = 0; };
    int     operator<<( FILE *fp_ARO );
    BOOLoperator>>( FILE *fp_GRN )
    {
        return TrunkParamClass::operator>>( fp_GRN );
    }
    voidanalyze();
    voidmakeTransMat_to_XYZ();
    voidmakeTransMat_to_UVW();
};

    int
    TrunkAnalysisClass::operator<<( FILE *fp_ARO )
    /*
        .AROﾌｧｲﾙの内容を読み込んで、LumberClass型のﾃﾞｰﾀ lumberに格納する。（& or ﾌｧｲﾙ先頭）～（& or EOF）までを一つのグループとみなし、そのグループ単
        位で読み込みを行う。

        戻り値      EOFに達した      0
        ﾌｧｲﾙがない、もしくは読み込み中に何らかのｴﾗｰが発生した  -1
        正常終了          1
    */
    {
        myfscanf( fp_ARO, "%d", &piece_number );
        myfscanf( fp_ARO, "%lg", &width );
        double  temp[3];
        myfscanf( fp_ARO, "%lg %lg %lg", &temp[0], &temp[1], &temp[2] );
        height = temp[0];
        for ( int i = 1; i < 3; i++ )
            height += temp[i];
        height /= 3.0;

        myfscanf( fp_ARO, "%lg", &length );
        firstSection << fp_ARO;
        secondSection << fp_ARO;

    secondSection.angle_with_respect_to_Xaxis
                                = M_PI - secondSection.angle_with_respect_to_Xaxis;
                    /*  M_PI - としているのは、末口面と元口面を併せて解析するために
                                はどちらかを裏からみなくてはならないから。 */

    //  dn_between_12 が与えられておらず、かつ、EOFに達した場合
        charline_string[82];
        if ( myfgets( line_string, 81, fp_ARO ) == NULL ){
            return  0;

    //  dn_between_12 が与えられている場合
        } else if ( *line_string != '&' ){
            dn_between_12 = atoi( line_string );

        //  先読みを行う。その結果 EOFに到達したら、0を返す
            if ( myfgets( line_string, 81, fp_ARO ) == NULL )
                return 0;
        }

    //  dn_between_12 が与えられておらず、EOFに達していない場合。
    //  値として0 を与える

        return 1;
    }

    inline  void
    TrunkAnalysisClass::analyze()
    /*
        入力ﾃﾞｰﾀを解析して、TrunkParamClassのメンバを求める
    */
    {
    //  両木口面の解析を行なう
        firstSection.analyze();
        secondSection.analyze();

    //  firstSectionの解析結果を、CrossSectionParamClassのメンバに代入
        centroid_radius = firstSection.centroid_radius;
        angle_with_respect_to_Xaxis = firstSection.angle_with_respect_to_Xaxis;
        radius_of_1 = firstSection.radius_of_1;
        xCentroid = firstSection.xCentroid;
        yCentroid = firstSection.yCentroid;
        zCentroid = firstSection.zCentroid;

    //  切削角度の解析
    //      X軸、W軸回りの回転角を求める。軸に対して右回りが正
    //                                  Z2 - Z1
    //              sinθ(sin_X) = ------------
    //  cosφ                            L
    //                                  X1 - X2
    //              sinφ(sin_W) = ------------
    //                                  L
    //                          L : XYZ.length

        sin_W = ( firstSection.xCentroid - secondSection.xCentroid ) / length;
        rotation_angle_to_W = asin( sin_W );
        cos_W = cos( rotation_angle_to_W );

        sin_X = ( secondSection.zCentroid - firstSection.zCentroid ) * cos_W
                                / length;
        rotation_angle_to_X = asin( sin_X );
        cos_X = cos( rotation_angle_to_X );

    //  yの範囲
        yrange = length / ( cos_W * cos_X );

    //  基準面の平均年輪幅と、ｙの変化に伴う平均年輪幅の変化率を求める。
        ring_width = firstSection.ring_width * cos_X;

//  誤差だよ、誤差！
        ring_width_gradient = ( secondSection.ring_width
                                -
firstSection.ring_width ) / yrange;
            /*  だいたい推定する時にあいうことをやっているのだから、今更
                        回転角度を考慮するのはどおかとも思うんだが.... */

    //  基準面の中心を通る年輪半径の、ｙの変化に伴う変化率（＝細り）を求める。

    //  まず、第一年輪の勾配を求める。
        double  r1_gradient = ( secondSection.radius_of_1 + dn_between_12
                        *
secondSection.ring_width * cos_X
```

```
firstSection.radius_of_1 ) / yrange;
    // ついで、基準面中心を通る樹幹の番号を求める。
    double  n =
( firstSection.centroid_radius -
firstSection.radius_of_1 )

    / firstSection.ring_width;
    //  中心の樹幹の細り（tanγ）は、次式で与えられる。
    // .... r1,r1_gradientをメンバとして持つ形にした方がすっきりするとは思う
    //      のだが、メンバが増えると嫌なので中心を係数として持つようにした
        rCentroid_gradient = r1_gradient +
n * ring_width_gradient;

    //  座標変換行列の生成
        makeTransMat_to_XYZ();
        makeTransMat_to_UVW();
    }

    inline void
    TrunkAnalysisClass::makeTransMat_to_UVW
()
    /*
            ＸＹＺ系の座標をＵＶＷ系に変換する座標変換行列を生成する
    */
    {
        transMat_to_UVW->element( 1, 1 ) =
cos_W;
        transMat_to_UVW->element( 1, 2 ) =
-sin_W;
        transMat_to_UVW->element( 1, 3 ) =
0.0;
        transMat_to_UVW->element( 1, 4 ) =
0.0;
        transMat_to_UVW->element( 2, 1 ) =
cos_X * sin_W;
        transMat_to_UVW->element( 2, 2 ) =
cos_X * cos_W;
        transMat_to_UVW->element( 2, 3 ) =
-sin_X;
        transMat_to_UVW->element( 2, 4 ) =
0.0;
        transMat_to_UVW->element( 3, 1 ) =
sin_X * sin_W;
        transMat_to_UVW->element( 3, 2 ) =
sin_X * cos_W;
        transMat_to_UVW->element( 3, 3 ) =
cos_X;
        transMat_to_UVW->element( 3, 4 ) =
0.0;
        transMat_to_UVW->element( 4, 1 )
                        = -xCentroid *
cos_W - zCentroid * sin_X * sin_W;
        transMat_to_UVW->element( 4, 2 )
                        = xCentroid *
sin_W - zCentroid * sin_X * cos_W;
        transMat_to_UVW->element( 4, 3 ) =
-zCentroid * cos_X;
        transMat_to_UVW->element( 4, 4 ) =
1.0;
    }

    inline void
    TrunkAnalysisClass::makeTransMat_to_XYZ
()
    /*
            ＵＶＷ系の座標をＸＹＺ系に変換する座標変換行列を生成する
    */
    {
        transMat_to_XYZ->element( 1, 1 ) =
cos_W;
        transMat_to_XYZ->element( 1, 2 ) =
cos_X * sin_W;
        transMat_to_XYZ->element( 1, 3 ) =
sin_X * sin_W;
        transMat_to_XYZ->element( 1, 4 ) =
0.0;
        transMat_to_XYZ->element( 2, 1 ) =
-sin_W;
        transMat_to_XYZ->element( 2, 2 ) =
cos_X * cos_W;
        transMat_to_XYZ->element( 2, 3 ) =
sin_X * cos_W;
        transMat_to_XYZ->element( 2, 4 ) =
0.0;
        transMat_to_XYZ->element( 3, 1 ) =
0.0;
        transMat_to_XYZ->element( 3, 2 ) =
-sin_X;
        transMat_to_XYZ->element( 3, 3 ) =
cos_X;
        transMat_to_XYZ->element( 3, 4 ) =
0.0;
        transMat_to_XYZ->element( 4, 1 ) =
xCentroid;
        transMat_to_XYZ->element( 4, 2 ) =
0.0;
        transMat_to_XYZ->element( 4, 3 ) =
zCentroid;
        transMat_to_XYZ->element( 4, 4 ) =
1.0;
    }

class    InputGrainClass {
/*
    GrainClass に加えて、*.AROファイルから読み込んだ測定データをメンバとして
    持つ関数。操作関数は、*.AROファイルからの入力と、*.GRNファイルへの出力に関
    連するもののみである。
*/
private :
    FILE*fp_ARO;
    FILE*fp_GRN;
    int      can_read;
    TrunkAnalysisClass   lumber;

public  :
    InputGrainClass( char *fname );
    ~InputGrainClass(){ fclose( fp_ARO );
fclose( fp_GRN ); }

    BOOLcannotOpen() {
        return ( fp_ARO == NULL || fp_GRN
== NULL ) ? TRUE : FALSE;
    }

    int      canRead(){  return  can_read;
    }

    voidInputGrainClass::read_from_ARO(){
                                    //   .ARO
ファイルからデータを読み込む
        can_read = lumber << fp_ARO;
    }
    voidInputGrainClass::write_to_GRN(){
                                    //   .ARO
ファイルからデータを読み込む
        lumber >> fp_GRN;
    }

    voidanalyze()        //  年輪、樹幹の解析を行う
    {
        lumber.analyze();
    }

};

    InputGrainClass::InputGrainClass( char
*ARO_fname ) : can_read(1)
    /*
            ﾃﾞｰﾀ入力用の .AROﾌｧｲﾙ及び出力用の .GRNﾌｧｲﾙを開く。.GRNﾌｧｲﾙ名は .AROﾌｧ
        ｲﾙの拡張子を .GRNに変えた名前とする。
    */
    {
        static char
    GRN_fname[PATH_LENGTH];

        if ( ( fp_ARO = fopen( ARO_fname,
"rt" ) ) == NULL )
            fopenFatal( ARO_fname );
        else {
        //  .ARN を .GRNに変え、GRN_fnameに与える
            strcpy( GRN_fname, ARO_fname );
            strtok( GRN_fname, "." );
            strcat( GRN_fname, ".GRN" );
            fp_GRN = fopen( GRN_fname,
"wb" );
            if ( fp_GRN == NULL )
                fopenFatal( GRN_fname );
        }
    }

main( int argc, char **argv )
{
    FILE*   fin;
    if ( argc > 1 )
        fin = fopen( argv[1], "rt" );
    else
        fin = stdin;

    charfname[CHAR_MAX];
    printf( "input ARO file name : " );
    while( myfscanf( fin, "%s", fname ) !=
EOF ){
        static InputGrainClass
    Grain( fname );
        if ( !Grain.cannotOpen() ) {
            while ( Grain.canRead() ){
                Grain.read_from_ARO();
```

```
            if ( Grain.canRead() == -
1 ) {
                break;      // 読み込
みｴﾗｰの発生。つぎの入力ﾌｧｲﾙ、もしなけれ
                            // ばつぎ
のﾌｧｲﾙ名指定へ制御を移す
            }
            Grain.analyze();
            Grain.write_to_GRN();
        }
    }
    printf( "input ARO file name : " );
}
if ( argc > 1 )
fclose( fin );
return  0;
}
```

(3) TRANS.H 座標変換に関するクラスの定義を行なう。

```
# ifndef __TRANS__
  # define __TRANS__
/*
  trans.h

  座標変換に関するｸﾗｽの定義
*/

# include   <lib¥matrix.h>
# include   <lib¥mystd.h>
# include   <math.h>
# include   <assert.h>

/*
  拡大・縮小変換を定義したクラス
*/
class   ScalingMatrixClass : public
MatrixClass {

public :
  ScalingMatrixClass( double Sx1, double
Sx2, double Sx3 ) : MatrixClass( 4, 4 )
    {
      element( 1, 1 ) = Sx1;
      element( 2, 2 ) = Sx2;
      element( 3, 3 ) = Sx3;
      element( 4, 4 ) = 1.0;
    }
};
```



```
/*
    同次座標表現の点を定義するクラス。
  MatrixClassの導出クラス
*/

// 同次座標表現のベクトルと、その操作を行
なう関数を定義するクラス
class   HCVectorClass : public MatrixClass
{

public :
  HCVectorClass() : MatrixClass( 1,4 )
{ };
  HCVectorClass( MatrixClass& point ) :
MatrixClass( point ){ };
  HCVectorClass( double x1, double x2,
double x3 ) : MatrixClass( 1,4 )
    {
      element( 1,1 ) = x1;
      element( 1,2 ) = x2;
      element( 1,3 ) = x3;
      element( 1,4 ) = 0;
    }
  HCVectorClass&  assign( double x1,
double x2, double x3 ){
      element( 1,1 ) = x1;
      element( 1,2 ) = x2;
      element( 1,3 ) = x3;
      element( 1,4 ) = 0;
      return  *this;
  };
  double& x1() const {return
  element(1,1); };
  double& x2() const {return
  element(1,2); };
  double& x3() const {return
  element(1,3); };
  voidoperator>>( FILE *fp )
  {
    fprintf( fp, "%.3lg, %.5lg, %.3lg",
x1(), x2(), x3() );
  }
  voidoperator<<( FILE *fp )
  {
    fscanf( fp, "%lg, %lg, %lg", &x1(),
&x2(), &x3() );
  }
  double      length() const
  {
    return  sqrt( x1()*x1() + x2()*x2()
+ x3()*x3() );
  }
  voidunit()
  // 単位ベクトル化を行なう
  {
    double  l = length();
    x1() = x1() / l;
    x2() = x2() / l;
    x3() = x3() / l;
  }
  voidabs()
  // 各成分の絶対値を求める
  {
    x1() = ( x1() < 0.0 ) ? -x1() :
x1();
    x2() = ( x2() < 0.0 ) ? -x2() :
x2();
    x3() = ( x3() < 0.0 ) ? -x3() :
x3();
  }
  friend  HCVectorClass
  unit( HCVectorClass vect );
  friend  double  innerProduct( const
HCVectorClass& vect1

  , const HCVectorClass& vect2 );
  friend  HCVectorClass
  vectorProduct( const HCVectorClass&
vect1

  , const HCVectorClass& vect2 );
  voidscaling( const ScalingMatrixClass&
scalingMat )
  {
    HCVectorClass   temp = *this *
scalingMat;
    *this = temp;
  }
  voidscaling( double Sx1, double Sx2,
double Sx3 )
  {
    ScalingMatrixClass
  scalingMat( Sx1, Sx2, Sx3 );
    scaling( scalingMat );
  }
  friend HCVectorClass
  scaling( HCVectorClass vect

const ScalingMatrixClass& scalingMat );
```



```
  friend HCVectorClass
  scaling( HCVectorClass vect

double Sx1, double Sx2, double Sx3 );
};

  inline  HCVectorClass
  unit( HCVectorClass vect )
  {
    vect.unit(); return vect;
  }
  inline double   innerProduct( const
HCVectorClass& vect1

  , const HCVectorClass& vect2 )
  {
    return  vect1.x1()*vect2.x1() +
vect1.x2()*vect2.x2()

  + vect1.x3()*vect2.x3();
  }
  inline HCVectorClass
  vectorProduct( const HCVectorClass&
vect1

  , const HCVectorClass& vect2 )
  {
    HCVectorClass   temp;

    temp.x1() = vect1.x2() * vect2.x3()
- vect1.x3() * vect2.x2();
    temp.x2() = vect1.x3() * vect2.x1()
- vect1.x1() * vect2.x3();
    temp.x3() = vect1.x1() * vect2.x2()
- vect1.x2() * vect2.x1();
    return  temp;
  }

  inline HCVectorClass
  scaling( HCVectorClass vect

const ScalingMatrixClass& scalingMat )
  {
    vect.scaling( scalingMat );
    return vect;
  }

  inline HCVectorClass
  scaling( HCVectorClass vect
```

```
double Sx1, double Sx2, double Sx3 )
    {
        vect.scaling( Sx1, Sx2, Sx3 );
        return  vect;
    }

//  同次座標表現の点を定義するクラス
class   HCPointClass : public HCVectorClass
{

public :
    HCPointClass() : HCVectorClass()
{ element( 1,4 ) = 1; };
    HCPointClass( HCVectorClass& point ) :
HCVectorClass( point ){ };
    HCPointClass( double x1, double x2,
double x3 ) : HCVectorClass()
    {
        element( 1,1 ) = x1;
        element( 1,2 ) = x2;
        element( 1,3 ) = x3;
        element( 1,4 ) = 1;
    }
    HCPointClass&    assign( double x1,
double x2, double x3 ){
        element( 1,1 ) = x1;
        element( 1,2 ) = x2;
        element( 1,3 ) = x3;
        element( 1,4 ) = 1;
        return  *this;
    };
};

/*
    円筒座標表現の点を定義するクラス
*/
class   CylinderClass {

public :
    CylinderClass(){}
    CylinderClass( double _r, double _alpha,
double _y )
    {   radius = _r; alpha = _alpha; y =
_y; }

    double      radius;
    double      alpha;
    double      y;

    voidoperator>>( char* string )
    {
        sprintf( string, "%.4lg, %.4lg",
radius, alpha );
    }
    voidoperator>>( FILE* fout )
    {
        fprintf( fout, "%.4lg, %.4lg",
radius, alpha );
    }
    voidoperator<<( FILE* fin )
    {
        fscanf( fin, "%lg, %lg", &radius,
&alpha );
    }
    voidassign( double _r, double _alpha,
double _y )
    {   radius = _r; alpha = _alpha; y =
_y; }
};

# endif
```

(4) Trunk.CPP TrunkParamClassのメンバ関数の定義を行なう

```
# include   "grain.h"
# include   <math.h>
# include   <lib¥mystd.h>

MatrixClass
    TrunkParamClass::makeTransMat_to_TLR(

    const HCPointClass& pointXYZ )
//  pointXYZにおけるＸＹＺ系からＴＬＲ系へ
の座標変換行列を定義する。
{
    CornClass   pointRAY =
trans_to_Corn( pointXYZ );
    double  cos_alpha =
cos( pointRAY.alpha );
    double  sin_alpha =
sin( pointRAY.alpha );
    double  sin_thin_angle =
sin( pointRAY.thin_angle );
    double  cos_thin_angle =
cos( pointRAY.thin_angle );

    MatrixClass     temp( 4,4 );
    temp.element( 1, 1 ) = sin_alpha;
    temp.element( 1, 2 ) = 0.0;
    temp.element( 1, 3 ) = cos_alpha;
    temp.element( 1, 4 ) = 0.0;
    temp.element( 2, 1 ) = 0.0;
    temp.element( 2, 2 ) =
1.0/cos_thin_angle;
    temp.element( 2, 3 ) = -sin_thin_angle
/ cos_thin_angle;
    temp.element( 2, 4 ) = 0.0;
    temp.element( 3, 1 ) = -cos_alpha;
    temp.element( 3, 2 ) = 0.0;
    temp.element( 3, 3 ) = sin_alpha;
    temp.element( 3, 4 ) = 0.0;
    temp.element( 4, 1 ) = -
pointXYZ.x1()*sin_alpha
+pointXYZ.x3()*cos_alpha;
    temp.element( 4, 2 ) = - pointXYZ.x2()
/ cos_thin_angle;
    temp.element( 4, 3 ) = -pointXYZ.x1() *
cos_alpha

    +pointXYZ.x2() *sin_thin_angle/
cos_thin_angle

    - pointXYZ.x3() * sin_alpha;
    temp.element( 4, 4 ) = 1.0;

    return  temp;
}

MatrixClass
    TrunkParamClass::makeTransMatTLR_to_XYZ
(

    const HCPointClass& pointXYZ )
//  pointXYZにおけるＴＬＲ系からＸＹＺ系へ
の座標変換行列を定義する。
{
    CornClass   pointRAY =
trans_to_Corn( pointXYZ );
    double  cos_alpha =
cos( pointRAY.alpha );
    double  sin_alpha =
sin( pointRAY.alpha );
    double  sin_thin_angle =
sin( pointRAY.thin_angle );
    double  cos_thin_angle =
cos( pointRAY.thin_angle );

    MatrixClass     temp( 4,4 );
    temp.element( 1, 1 ) = sin_alpha;
    temp.element( 1, 2 ) = 0.0;
    temp.element( 1, 3 ) = -cos_alpha;
    temp.element( 1, 4 ) = 0.0;
    temp.element( 2, 1 ) = cos_alpha *
sin_thin_angle;
    temp.element( 2, 2 ) = cos_thin_angle;
    temp.element( 2, 3 ) = sin_alpha *
sin_thin_angle;
    temp.element( 2, 4 ) = 0.0;
    temp.element( 3, 1 ) = cos_alpha;
    temp.element( 3, 2 ) = 0.0;
    temp.element( 3, 3 ) = sin_alpha;
    temp.element( 3, 4 ) = 0.0;
    temp.element( 4, 1 ) = pointXYZ.x1();
    temp.element( 4, 2 ) = pointXYZ.x2();
    temp.element( 4, 3 ) = pointXYZ.x3();
    temp.element( 4, 4 ) = 1.0;

    return  temp;
}

CrossSectionParamClass
    TrunkParamClass::getCrossSection( doubl
e l )
{
    HCPointClass centerUVW( 0.0, l, 0.0 );
    HCPointClass centerXYZ =
trans_to_XYZ( centerUVW );
    CornClass   centerRAY =
trans_to_Corn( centerXYZ );

    CrossSectionParamClass  temp;

    temp.centroid_radius =
centerRAY.radius;
    temp.angle_with_respect_to_Xaxis =
centerRAY.alpha;
    temp.xCentroid = centerXYZ.x1();
    temp.yCentroid = centerXYZ.x2();
    temp.zCentroid = centerXYZ.x3();

    return  temp;
}

double
    TrunkParamClass::radiusOffirstRing( dou
ble y )
//  Y==y平面において、最初に木裏面にあらわ
れる年輪の半径を求める。
//  この論理は板目材にしか使えないよ多分。
```

```
{
    double  thin_angle_of_l =
thinAngle( radius_of_l, y );
    double  r_of_l = radius_of_l + y *
tan( thin_angle_of_l );
    double  dr = ringWidth( y );
    double  r_to_face = zCentroid - height/
2.0;
    if ( r_of_l < r_to_face ){
        while( r_of_l < r_to_face )
            r_of_l += dr;
    } else {
        while( r_of_l > r_to_face )
            r_of_l -= dr;
        r_of_l += dr;
    }
    return  r_of_l;
}
```

(5)　FiberDir.H　UVW系上の任意のベクトルの、T軸、L軸、R軸となす角度、TLR座標系表現を求める。

```
# include   <math.h>
# include   "trans.h"
# include   "grain.h"

class   FiberDirClass {

protected :

    HCVectorClass    Taxis_TLR;
    HCVectorClass    Laxis_TLR;
    HCVectorClass    Raxis_TLR;
    TrunkParamClass* trunk;

public :
    HCVectorClass    pointUVW;
    HCVectorClass    vectTLR;
    double  Tangle;
    double  Langle;
    double  Rangle;

//  コンちゃん
    FiberDirClass(){
        Taxis_TLR.assign( 1.0, 0.0, 0.0 );
        Laxis_TLR.assign( 0.0, 1.0, 0.0 );
        Raxis_TLR.assign( 0.0, 0.0, 1.0 );
    };
    FiberDirClass( TrunkParamClass*
_trunk ) {
        setTrunk( _trunk );
FiberDirClass();
    }
    FiberDirClass( TrunkParamClass* _trunk,
const HCPointClass& _pointUVW ) {
        setTrunk( _trunk );
        setPoint( _pointUVW );
        FiberDirClass();
    }
    FiberDirClass( TrunkParamClass*
_trunk,double u,double v, double w ){
        setTrunk( _trunk );
        setPoint( u, v, w );
        FiberDirClass();
    }

//  コンちゃんで初期化できない場合の設定関
数
    voidsetPoint( double u, double v,
double w ){
        pointUVW.assign( u, v, w );
    }
    voidsetPoint( const HCPointClass&
_pointUVW ) { pointUVW = _pointUVW; }
    voidsetTrunk( TrunkParamClass* _trunk )
{ trunk = _trunk; }

//  任意の座標軸となす角度を返す
    double  getAngle( const HCVectorClass&
axis_TLR, const HCVectorClass& vect_UVW )
    {
        HCVectorClass    axis_UVW = trunk-
>transTLR_to_UVW( pointUVW, axis_TLR);
        double  cos_Angle =
innerProduct( axis_UVW, vect_UVW )
                                    /
axis_UVW.length() * vect_UVW.length();
        return acos( cos_Angle );
    }

    voidanalysys( const HCVectorClass&
vect_UVW )
    {
        Tangle = getAngle( Taxis_TLR,
vect_UVW );
        Langle = getAngle( Laxis_TLR,
vect_UVW );
        Rangle = getAngle( Raxis_TLR,
vect_UVW );

        vectTLR = trunk-
>transUVW_to_TLR( pointUVW, vect_UVW );
    }

    voidoperator>>( FILE* fout )
    {
        fprintf( fout,
"%7.3lg, %7.5lg, %7.3lg,", Tangle, Langle,
Rangle );
        vectTLR >> fout;
    }
};

# endif
```

2.　破壊シミュレーション用のソースリスト

(1) Block.H 木口断面を分割したブロックをメンバとして持つクラスの定義を行う

```
# include   "fiberdir.h"

class   BlockClass : public FiberDirClass {
/*
        任意木口断面を分割したブロックの、異
方性解析およびその結果度を持つクラス
        FiberDirClassの派生クラス。派生した点
は、
        ・極座標値をメンバとして持つ。そのために、関数setPoint()とコンストラクタを継承した。
        ・その点で二つのベクトルについて解析を行なうので、その結果を保持するメンバ変数を定義した。
*/

protected :

    CornClass    pointRAY;
    HCVectorClass    Vaxis_UVW;

public :
    HCVectorClass    Vaxis_TLR;
    HCVectorClass    StressDir_Bend_TLR;
                                    //  曲
げ試験時の応力の作用方向ベクトル
    double  Langle_of_Vaxis;
    double  Langle_of_StressDir;

    BlockClass() { Vaxis_UVW.assign( 0.0,
1.0, 0.0 ); }
    BlockClass( TrunkParamClass* _trunk ) {
        setTrunk( _trunk );
        Vaxis_UVW.assign( 0.0, 1.0, 0.0 );
    }
    BlockClass( TrunkParamClass* _trunk,
const HCPointClass& _pointUVW ) {
        setTrunk( _trunk );
        setPoint( _pointUVW );
        Vaxis_UVW.assign( 0.0, 1.0, 0.0 );
    }
    BlockClass( TrunkParamClass* _trunk,
double u,double v, double w ){
        setTrunk( _trunk );
        setPoint( u,v,w );
        Vaxis_UVW.assign( 0.0, 1.0, 0.0 );
    }

//  コンちゃんで初期化できない場合の設定関
数
    voidsetPoint( double u, double v,
double w )
    {
        pointUVW.assign( u, v, w );
        HCPointClass pointXYZ = trunk-
>trans_to_XYZ( pointUVW );
        pointRAY = trunk-
>trans_to_Corn( pointXYZ );
    }
    voidsetPoint( const HCPointClass&
_pointUVW )
    {
        pointUVW = _pointUVW;
        HCPointClass pointXYZ = trunk-
>trans_to_XYZ( pointUVW );
        pointRAY = trunk-
>trans_to_Corn( pointXYZ );
    }

    voidanalysys()
    {
    //  V軸について解析する。

    FiberDirClass::analysys( Vaxis_UVW );
```

```
        Vaxis_TLR = vectTLR;
        Langle_of_Vaxis = Langle;

    //  曲げ試験時の応力作用軸を求める。L
軸のＵＶ平面への射影ベクトルが応力作
    //  用軸であるとみなす
        HCVectorClass   Laxis_UVW = trunk-
>transTLR_to_UVW( pointUVW, Laxis_TLR );
        HCVectorClass
    StressDir_Bend_UVW( Laxis_UVW.x1(),
Laxis_UVW.x2(), 0.0 );
        StressDir_Bend_UVW.unit();

    //  その軸について解析する

    FiberDirClass::analysys( StressDir_Bend
_UVW );
        StressDir_Bend_TLR = vectTLR;
        Langle_of_StressDir = Langle;
    }

    voidoperator>>( FILE* fout )
    {
        fwrite( &(pointUVW.x1()),
sizeof( double ), 1, fout );
        fwrite( &(pointUVW.x3()),
sizeof( double ), 1, fout );
        fwrite( &(pointRAY.radius),
sizeof( double ), 1, fout );
        fwrite( &(pointRAY.alpha),
sizeof( double ), 1, fout );
        fwrite( &Langle_of_Vaxis,
sizeof( double ), 1, fout );
        fwrite( &(Vaxis_TLR.x1()),
sizeof( double ), 1, fout );
        fwrite( &(Vaxis_TLR.x2()),
sizeof( double ), 1, fout );
        fwrite( &(Vaxis_TLR.x3()),
sizeof( double ), 1, fout );
        fwrite( &Langle_of_StressDir,
sizeof( double ), 1, fout );
        fwrite( &(StressDir_Bend_TLR.x1()),
sizeof( double ), 1, fout );
        fwrite( &(StressDir_Bend_TLR.x2()),
sizeof( double ), 1, fout );
        fwrite( &(StressDir_Bend_TLR.x3()),
sizeof( double ), 1, fout );
    }

    BOOLoperator<<( FILE* fin )
    {
        if ( fread( &(pointUVW.x1()),
sizeof( double ), 1, fin ) < 1 )
            return  FALSE;
        fread( &(pointUVW.x3()),
sizeof( double ), 1, fin );
        fread( &(pointRAY.radius),
sizeof( double ), 1, fin );
        fread( &(pointRAY.alpha),
sizeof( double ), 1, fin );
        fread( &Langle_of_Vaxis,
sizeof( double ), 1, fin );
        fread( &(Vaxis_TLR.x1()),
sizeof( double ), 1, fin );
        fread( &(Vaxis_TLR.x2()),
sizeof( double ), 1, fin );
        fread( &(Vaxis_TLR.x3()),
sizeof( double ), 1, fin );
        fread( &Langle_of_StressDir,
sizeof( double ), 1, fin );
        fread( &(StressDir_Bend_TLR.x1()),
sizeof( double ), 1, fin );
        fread( &(StressDir_Bend_TLR.x2()),
sizeof( double ), 1, fin );
        fread( &(StressDir_Bend_TLR.x3()),
sizeof( double ), 1, fin );

        return  TRUE;
    }
};

class   BlockElasticityClass : public
BlockClass {
/*
    ・calcEratio() : 伝播速度と角度の関係か
らヤング係数低減率を推定。
    ・その点で二つのベクトルについて解析を
行なうので、その結果を保持するメ
      ンバ変数を定義した。
*/

private :
    double  sqr( double x ) { return x *
x; }
    double  regression_equation( double
x ){ return sqr( -1.033 * x + 1.0 ); }

public :
    double  Eratio_V;
    double  Eratio_S;
```



```
    voidcalcEratio()
    {
        Eratio_V =
regression_equation( tan( Langle_of_Vaxis )
);
        Eratio_S =
regression_equation( tan( Langle_of_StressD
ir) );
    }
    voidoperator>>( FILE* fout ) {
        fwrite( &Eratio_V, sizeof( double ),
1, fout );
        fwrite( &Eratio_S, sizeof( double ),
1, fout );
    }
};

    enum FAILURE_MODE { TENTION,
PLANE_TENTION, PLANE_SHEAR };

class   BlockStrClass {

private :
    float   u;
    float   w;

    friend double
    calcStrain_t( BlockStrClass& block,
double stress_L_max );
    friend double
    calcStrain_plane( BlockStrClass& block

    , double stress_plane_max );
    friend double
    calcStrain_shear( BlockStrClass& block,
double shear_max );
    friend void
    limitStrain( BlockStrClass* block,
double strain_t
                                , double
strain_plane, double strain_shear );

public:
    double  Eratio;
    double  Langle;
    double  limit_strain;
    FAILURE_MODEmode;
    friend BOOL
    getLimitStrain( BlockStrClass* block
                            , FILE* fCS, FILE*
fE, double stress_L_max
                            , double
stress_plane_max, double
stress_shear_max );
};

    double  calcStrain_t( BlockStrClass&
block, double stress_L_max );
    double
    calcStrain_plane( BlockStrClass& block,
double stress_plane_max );
    voidlimitStrain( BlockStrClass* block,
double strain_t
                            , double
strain_plane, double strain_shear );
    BOOLgetLimitStrain( BlockStrClass*
block
                            , FILE* fCS, FILE*
fE, double stress_L_max
                            , double
stress_plane_max, double
stress_shear_max );

class   BlockBendClass {

private :
    friend double
    Tstrain( BlockBendClass& block, double
stress_L_max );
    friend double
    Pstrain( BlockBendClass& block

    , double stress_plane_max );
    friend double
    Sstrain( BlockBendClass& block, double
shear_max );
    friend void
    blockStrain( BlockBendClass* block,
double strainain_t
                                , double
Pstrain_ten, double Sstrain_ten );

public:
    float   u;
    float   w;

    double  Eratio;
```

```
    double  Langle;
    float   tensile_strain;
    float   comp_strain;
                // 今回は、外部応力を変えていくのだからこの精度で十分であろう
    FAILURE_MODE tensile_mode;
    FAILURE_MODE comp_mode;

    friend BOOL
    getLimitStrain( BlockBendClass* block
                    , FILE* fCS, FILE* fE, double stress_L_max
                    , double stress_plane_max, double stress_shear_max );
};

# endif
```

(2) Section.H 木材中の任意の木口断面を定義するクラス

```
# ifndef __SECTION__
    # define __SECTION__

/*
    section.h
*/

/*
      板材の中央断面のパラメータ及びメッシュに切ったブロックをメンバとして持つ
    クラスの定義
*/
# include   "block.h"
# include   <stdio.h>
# include   <lib¥mystd.h>

class   CenterSectionClass {

protected:
    TrunkParamClass trunk;

    double  y;
    double  v;
    double  dr;
    double  radius_of_first;
    double  block_size;

public :
    int piece_number;
    long    total_row;
    long    total_column;

    CenterSectionClass() {}

    voidsetTrunk( TrunkParamClass& _trunk, double _block_size )
    {
    // 材の長手方向の中央断面における drとradius_of_firstを求める。
        trunk = _trunk;
        piece_number = trunk.piece_number;
        v = trunk.length/2.0;
        HCPointClass pointUVW( 0.0, v, 0.0 );
        HCPointClass pointXYZ = trunk.trans_to_XYZ( pointUVW );

        y = pointXYZ.x2();
        dr = trunk.ringWidth( y );
        radius_of_first = trunk.radiusOffirstRing( y );

    // 分割するブロックの数（列の数、行の数）を求める。材の幅（厚さを）長さ
    // block_size の正方形のブロックに分割する。
        block_size = _block_size;
        total_row = floorl( trunk.width / block_size );
        total_column = floorl( trunk.height / block_size );
    }

    BOOLanalysys( FILE* fout );
    voidoperator>>( FILE* fout )
    {
        fwrite( &piece_number, sizeof( int ), 1, fout );
        fwrite( &y, sizeof( double ), 1, fout );
        fwrite( &v, sizeof( double ), 1, fout );
        fwrite( &dr, sizeof( double ), 1, fout );
        fwrite( &radius_of_first, sizeof( double ), 1, fout );
        fwrite( &total_column, sizeof( long ), 1, fout );
        fwrite( &total_row, sizeof( long ), 1, fout );
        fwrite( &block_size, sizeof( double ), 1, fout );
    }

    BOOLoperator<<( FILE* fin )
    {
        if ( fread( &piece_number, sizeof( int ), 1, fin ) < 1 )
            return FALSE;
        fread( &y, sizeof( double ), 1, fin );
        fread( &v, sizeof( double ), 1, fin );
        fread( &dr, sizeof( double ), 1, fin );
        fread( &radius_of_first, sizeof( double ), 1, fin );
        fread( &total_column, sizeof( long ), 1, fin );
        fread( &total_row, sizeof( long ), 1, fin );
        fread( &block_size, sizeof( double ), 1, fin );
        return TRUE;
    }
};

class   CenterBlockEClass : public CenterSectionClass {
    BlockElasticityClass block;
    FILE*fout;

public :
    BOOLanalysys( FILE* fin, FILE* fout );
};

    inline int      compBlock( const void* a, const void* b )
    {
        BlockStrClass*  block_1;
        BlockStrClass*  block_2;

        block_1 = (BlockStrClass*)a;
        block_2 = ( BlockStrClass*)b;
        if ( block_1->limit_strain < block_2->limit_strain )
            return  -1;
        else if ( block_1->limit_strain == block_2->limit_strain )
            return  0;
        else
            return  1;
    }

class   CenterSectionStrClass : public CenterSectionClass {

protected :
    double  stress_t_max;
    double  stress_plane_max;
    double  stress_shear_max;
    longtotal;
    longfail_fiber;
    double  Eratio_min;
    double  max_angle;
    double  sum_Eratio;
    double  max_strain;
    BlockStrClass huge* blockStr;

    long    elem( long column, long row ){
        return  column * total_row + row;
    }
    double  calcMOE( int fail_fiber );

public :
    double  Eratio_first;
    double  max_stress;
    double  fail_mode[3];
    longfail_time;

    CenterSectionStrClass(){stress_t_max = 1.0;}
    BOOLanalyzePiece( FILE* fCS, FILE* fE, FILE* fTension
                                    , double stress_plane, double stress_shear );

    voidoperator>>( FILE* fTension )
    {
        fwrite( &piece_number, sizeof( int ), 1, fTension );
        fwrite( &Eratio_first, sizeof( double ), 1, fTension );
        fwrite( &Eratio_min, sizeof( double ), 1, fTension );
```

```
        fwrite( &max_angle,
sizeof( double ), 1, fTension );
        fwrite( &max_stress,
sizeof( double ), 1, fTension );
        fwrite( &max_strain,
sizeof( double ), 1, fTension );
        fwrite( &fail_time, sizeof( long ),
1, fTension );
        fwrite( fail_mode, sizeof( double ),
3, fTension );
    }

    BOOLoperator<<( FILE* fTension )
    {
        if ( fread( &piece_number,
sizeof( int ), 1, fTension ) < 1 )
            return  FALSE;
        fread( &Eratio_first,
sizeof( double ), 1, fTension );
        fread( &Eratio_min,
sizeof( double ), 1, fTension );
        fread( &max_angle, sizeof( double ),
1, fTension );
        fread( &max_stress,
sizeof( double ), 1, fTension );
        fread( &max_strain,
sizeof( double ), 1, fTension );
        fread( &fail_time, sizeof( long ),
1, fTension );
        fread( fail_mode, sizeof( double ),
3, fTension );
        return  TRUE;
    }

    voidwrite( FILE* fTension, double
stress, double strain, double mode ){
        fprintf( fTension,
"%lg, %lg, %lg¥n", stress, strain, mode );
    }
};

class   CenterSectionBendClass : public
CenterSectionClass {

protected :
    BlockBendClass huge* block;
    BOOLisPlasticFail;
    double  neutral_plane;
    double  Z;
    float   A;      // Cross section area
of a Fiber Element.

    double  sum_E_ww;
    double  sum_E_w;
    double  sum_E;
    double  sum_CFstress;
    double  sum_CFstress_w;
    float   Moment;
    float   dM;

    double  stress_t_max;
    double  stress_plane_max;
    double  stress_shear_max;

    int     total_fiber;
    int     fail_fiber;

    voidswap_block( BlockBendClass huge& a,
BlockBendClass huge& b )
    // 実は入れ替える必要はないので上書き
する。
    {
        a = b;
    }
    double  getNeutralPlane(){
        double  temp = A * sum_CFstress_w -
Moment;
        double  child = sum_E_w * temp - A
* sum_E_ww * sum_CFstress;
        double  mother = sum_E * temp - A *
sum_E_w * sum_CFstress;
        return  child / mother;
    }
    double  calcK() {
        double  child = Moment - A *
sum_CFstress_w
                                    + A *
neutral_plane * sum_CFstress;
        double  mother = A * ( sum_E_ww -
2.0 * neutral_plane * sum_E_w
                                +
neutral_plane * neutral_plane * sum_E );
        return  child / mother;
    }

    int elem( int column, int row ){
        return  column * total_row + row;
    }
```



```
    BOOLcheckFail( BlockBendClass huge&
block );
    float   getInitialMoment();

public :
    double  max_stress;
    double  fail_mode[3];
    int     fail_time;

    CenterSectionBendClass(){   stress_t_m
ax = 1.0;   }
    BOOLanalyzePiece( FILE* fCS, FILE* fE,
FILE* fBend
                        , double
stress_plane, double stress_shear );

    voidoperator>>( FILE* fBend )
    {
        fwrite( &piece_number,
sizeof( int ), 1, fBend );
        fwrite( &max_stress,
sizeof( double ), 1, fBend );
        fwrite( &fail_time, sizeof( int ),
1, fBend );
        fwrite( fail_mode, sizeof( double ),
3, fBend );
    }

    BOOLoperator<<( FILE* fBend )
    {
        if ( fread( &piece_number,
sizeof( int ), 1, fBend ) < 1 )
            return  FALSE;
        fread( &max_stress,
sizeof( double ), 1, fBend );
        fread( &fail_time, sizeof( int ), 1,
fBend );
        fread( fail_mode, sizeof( double ),
3, fBend );
        return  TRUE;
    }
};

# endif
```

(3) Section.CPP CenterSectionClass とその派生クラスのメンバ関数を定義する

```
/*
    section.cpp
        CenterSectionClass のメンバ関数の定義
*/

# include   "section.h"

BOOLCenterSectionClass::analysys( FILE*
fout )
{

//  ファイルに情報を出力
    *this >> fout;
    printf( "%d¥n", piece_number );

//  左下のブロックから順番に解析
    double  w = -trunk.height/2.0 +
block_size/2.0;
    for ( int column = 0; column <
total_column; column++, w+=block_size ){
        double  u = -trunk.width/2.0 +
block_size/2.0;
        for ( int row = 0; row < total_row;
row++, u+=block_size ){
            BlockClass  block( &trunk, u,
v, w );
            block.analysys();
            block >> fout;
        }
    }
    return  TRUE;
}

BOOLCenterBlockEClass::analysys( FILE* fin,
FILE* fout )
{
    if ( !( *this << fin ) )
        return  FALSE;
    printf( "%d¥n", piece_number );

    for( int column = 0; column <
total_column; column++ ){
        for( int row = 0; row < total_row;
row++ ){
            block << fin;
            block.calcEratio();
            block >> fout;
        }
    }
    return  TRUE;
}
```

```
BOOL
    CenterSectionStrClass::analyzePiece( FI
LE* fCS, FILE* fE
                , FILE* fTension, double
stress_plane, double stress_shear )
{
    stress_plane_max = stress_plane;
    stress_shear_max = stress_shear;

    if
( !( CenterSectionClass::operator<<( fCS )
) )
        return  FALSE;

    total = total_column * total_row;
    blockStr = new BlockStrClass[total];
    if ( blockStr == NULL )
        return  FALSE;

// ブロックの情報を読み込んで、各ﾌﾞﾛｯｸの最
大ひずみを求める。
    sum_Eratio = 0.0;
    for ( long column = 0; column <
total_column; column++ ){
        for ( long row = 0; row <
total_row; row++ ){

    getLimitStrain( (BlockStrClass*)(blockS
tr+elem( column,row )), fCS, fE
                    , stress_t_max,
stress_plane_max, stress_shear_max );
        sum_Eratio +=
blockStr[elem( column,row )].Eratio;
        }
    }
    Eratio_first = sum_Eratio /
(double)total;

    printf( "%d, %lg, ", piece_number,
Eratio_first );
    fail_fiber = 0;

// 最大ひずみの小さい順番にブロックを並べ
変える
    qsort( blockStr, total,
sizeof( BlockStrClass ), compBlock );
    Eratio_min = blockStr[0].Eratio;
    max_angle = blockStr[0].Langle;

// その順番に繊維が切れていく過程の、みか
けのヤング係数と最大応力の変化をシミ
// ュレートする
    max_stress = 0.0;
    fail_time = 0;
    fail_mode[0] = fail_mode[1] =
fail_mode[2] = 0.0;

    double  Eratio = Eratio_first;
    for ( long i = 0; i < total; i++ ) {
        double  stress = Eratio *
blockStr[i].limit_strain;
        fail_mode[blockStr[i].mode]++;

        if ( max_stress < stress ){
            max_stress = stress;
            max_strain =
blockStr[i].limit_strain;
            fail_time = i;
        } else {
            if ( i > fail_time +
total/10 ){
                i++;
                break;      // もうあがら
ないとみなす
            }
        }
        Eratio = calcMOE( i );
    }
    *this >> fTension;

    printf( "%.5lg, %ld ¥n", max_stress,
fail_time );
    delete  blockStr;
    return  TRUE;
}

double  CenterSectionStrClass::calcMOE( int
fail_fiber )
{
    sum_Eratio -=
blockStr[fail_fiber].Eratio;
    return  sum_Eratio / double( total );
}
```


(4) crossFD1.cpp 材の中央部分の断面部分を一片 1mmの正方形のブロックに分けて、各ﾌﾞﾛｯｸの異方性度を求める。

```
# include   "section.h"
# include   <lib¥mystd.h>
# include   <lib¥mydir.h>
# include   <string.h>
# include   <limits.h>

class   EstimateCrossFDClass {

private :
    FILE*   fin;
    FILE*   fout;
    BOOLcan_open;
    TrunkParamClass     trunk;
    CenterSectionClass  section;

public :
    EstimateCrossFDClass( char *fname )
    {
        fin = fopen( fname, "rb" );
        if ( fin != NULL )
            can_open = TRUE;

        chgExtension( fname, "CS" );
        fout = fopen( fname, "wb" );
        if ( fout == NULL )
            can_open = FALSE;
    }
    ~EstimateCrossFDClass(){
        fclose( fin );
        fclose( fout );
    }
    BOOLreadPiece()  { return
trunk<<( fin ); }
    BOOLcanOpen()    { returncan_open; }

    BOOLanalysis()
    {
        section.setTrunk( trunk, 0.1 );
        return  section.analysys( fout );
    }
};

int     main( )
{
    charfname[CHAR_MAX];

    printf( "input GRN file name : " );
    while( fscanf( stdin, "%s", fname ) !=
EOF ){
        EstimateCrossFDClass
    lumber( fname );
        if ( lumber.canOpen() ){
            while ( lumber.readPiece() )
                lumber.analysis();
        } else
            fopenFatal( fname );
        printf( "input GRN file name : " );
    }
    return  0;
}
```

(5) crossE.cpp .CSファイルを読み込んで、各ﾌﾞﾛｯｸのヤング係数低減率を求める。

```
# include   "section.h"
# include   <lib¥mystd.h>
# include   <lib¥mydir.h>
# include   <string.h>
# include   <limits.h>
# include   <math.h>

inline double   sqr( double a ) { return a
* a; }

class   EstimateEClass {

private :
    FILE*   fin;
    FILE*   fout;
    BOOLcan_open;
    CenterBlockEClass   section;

public :
    EstimateEClass( char *fname ) {
        fin = fopen( fname, "rb" );
        if ( fin != NULL )
            can_open = TRUE;

        chgExtension( fname, "E" );
        fout = fopen( fname, "wb" );
        if ( fout == NULL )
            can_open = FALSE;
```

```
    }
    ~EstimateEClass(){
        fclose( fin );
        fclose( fout );
    }
    BOOLreadPiece() { return
section.analysys( fin, fout ); }
    BOOLcanOpen()    { return can_open; }
};

int     main( )
{
    static char fname[CHAR_MAX];

    printf( "input CS file name : " );
    while( fscanf( stdin, "%s", fname ) !=
EOF ){
        static EstimateEClass
    lumber( fname );
        if ( lumber.canOpen() ){
            while ( lumber.readPiece() )
                ;
        } else
            fopenFatal( fname );
        printf( "input CS file name : " );
    }
    return  0;
}
```

(6) tfail.cpp 繊維モデルによる、引張り破壊のシミュレーションを行なう。

```
# include   "section.h"
# include   <lib¥mystd.h>
# include   <lib¥mydir.h>
# include   <string.h>
# include   <limits.h>

class   EstimateStrengthClass {

private :
    FILE*   fE;
    FILE*   fCS;
    FILE*   fTension;
    BOOLcan_open;
    CenterSectionStrClass   section;
    double  stress_shear_max;
    double  stress_plane_max;

public :
    EstimateStrengthClass( char *fname, int
plane_condition, int shear_condition ) {
        strcat( fname, ".CS" );
        fCS = fopen( fname, "rb" );
        if ( fCS != NULL )
            can_open = TRUE;

        chgExtension( fname, "E" );
        fE = fopen( fname, "rb" );
        if ( fE == NULL )
            can_open = FALSE;

        chgExtension( fname, "TEN" );
        fTension = fopen( fname, "wb" );
        if ( fTension == NULL )
            can_open = FALSE;
        fwrite( &plane_condition,
sizeof( int ), 1, fTension );
        fwrite( &shear_condition,
sizeof( int ), 1, fTension );
    }
    ~EstimateStrengthClass(){
        fclose( fCS );
        fclose( fE );
        fclose( fTension );
    }
    BOOLcanOpen()    { return can_open; }
    voidseek( double plane_max, double
shear_max ){
        stress_plane_max = plane_max;
        stress_shear_max = shear_max;
        fseek( fCS, 0, SEEK_SET );
        fseek( fE, 0, SEEK_SET );
        fwrite( &stress_plane_max,
sizeof( double ), 1, fTension );
        fwrite( &stress_shear_max,
sizeof( double ), 1, fTension );
    }
    BOOLanalyzePiece() {
        if ( section.analyzePiece( fCS, fE,
fTension
                                   ,
stress_plane_max, stress_shear_max ) )
            return  TRUE;
        else
            return  FALSE;
    }
};
```



```
int     main( )
{
    static char fname[CHAR_MAX];
    int     plane_condition = 7;
    static double   stress_plane_max[]
        = { 1.0, 0.01, 0.005, 0.004,
0.003, 0.002, 0.001 };
    int     shear_condition = 11;
    static double   stress_shear_max[]
        = { 1.0, 0.1, 0.009, 0.08, 0.07,
0.06, 0.05, 0.04, 0.03, 0.02, 0.01 };
    printf( "input file name : " );
    while( fscanf( stdin, "%s", fname ) !=
EOF ){
        static EstimateStrengthClass
    lumber( fname, plane_condition,
shear_condition );
        for ( int i = 0; i <
plane_condition ; i++ ) {
            for( int j = 0; j <
shear_condition; j++ ){

    lumber.seek( stress_plane_max[i],
stress_shear_max[j]);
                while
( lumber.analyzePiece() )
                    ;
            }
        }
        printf( "input file name : " );
    }
    return  0;
}
```

(7) bfail.cpp 繊維モデルによる、曲げ破壊のシミュレーションを行なう。

```
# include   "section.h"
# include   <lib¥mystd.h>
# include   <lib¥mydir.h>
# include   <string.h>
# include   <limits.h>


    double  Tstrain( BlockBendClass& block,
double stress_L_max ) {
        double  cos_L =
cos( block.Langle );
        return  stress_L_max /
( block.Eratio*cos_L*cos_L );
    }

    double  Pstrain( BlockBendClass& block,
double stress_plane_max ){
        double  sin_L =
sin( block.Langle );
        return  ( sin_L != 0.0L ) ?
                    stress_plane_max /
( block.Eratio*sin_L*sin_L )
                    : HUGE_VAL;
    }

    double  Sstrain( BlockBendClass& block,
double shear_max ){
        double  sin_L =
sin( block.Langle );
        double  cos_L =
cos( block.Langle );
        return  ( sin_L != 0.0L ) ?
                    shear_max /
( block.Eratio*sin_L*cos_L )
                    : HUGE_VAL;
    }

    voidblockStrain( BlockBendClass* block
                    , double Tstrain_ten,
double Tstrain_comp
                    , double Pstrain_ten,
double Pstrain_comp
                    , double Sstrain_ten,
double Sstrain_comp )
    {
        block->tensile_strain =
Tstrain_ten;
        block->tensile_mode = TENTION;

        if ( Pstrain_ten < block-
>tensile_strain ){
            block->tensile_strain =
Pstrain_ten;
            block->tensile_mode =
PLANE_TENTION;
        }
        if ( Sstrain_ten < block-
>tensile_strain ){
```

```
            block->tensile_strain =
Sstrain_ten;
            block->tensile_mode =
PLANE_SHEAR;
        }

        block->comp_strain = Tstrain_comp;
        block->comp_mode = TENTION;

        if ( Pstrain_comp < block-
>comp_strain ){
            block->comp_strain =
Pstrain_comp;
            block->comp_mode =
PLANE_TENTION;
        }
        if ( Sstrain_comp < block-
>comp_strain ){
            block->comp_strain =
Sstrain_comp;
            block->comp_mode = PLANE_SHEAR;
        }
    }

    BOOLgetLimitStrain( BlockBendClass*
block
                        , FILE* fCS, FILE*
fE, double stress_L_max
                        , double
stress_plane_max, double stress_shear_max )
    //
    //  必要なのは、
    //      Langle_of_StressDir
    //      Eratio_v
    //      u,w座標
    //  だけであるので、他は読み捨てる。
    //
    {
        static double   buf;
        if ( fread( &buf, sizeof( double ),
1, fCS ) < 1 )
            return  FALSE;
        block->u = buf;
        fread( &buf, sizeof( double ), 1,
fCS );
        block->w = buf;
        fread( &buf, sizeof( double ), 1,
fCS );
        fread( &buf, sizeof( double ), 1,
fCS );
        fread( &buf, sizeof( double ), 1,
fCS );
        fread( &buf, sizeof( double ), 1,
fCS );
        fread( &buf, sizeof( double ), 1,
fCS );
        fread( &block->Langle,
sizeof( double ), 1, fCS );
        fread( &buf, sizeof( double ), 1,
fCS );
        fread( &buf, sizeof( double ), 1,
fCS );
        fread( &buf, sizeof( double ), 1,
fCS );

        fread( &buf, sizeof( double ), 1,
fE );
        fread( &block->Eratio,
sizeof( double ), 1, fE );

        double  Tstrain_ten =
Tstrain( *block, stress_L_max );
        double  Tstrain_comp = Tstrain_ten
/ 2.0;
        double  Pstrain_ten =
Pstrain( *block, stress_plane_max );
        double  Pstrain_comp = Pstrain_ten
/ 2.0;
        double  Sstrain_ten =
Sstrain( *block, stress_shear_max );
        double  Sstrain_comp = Sstrain_ten;

        blockStrain( block, Tstrain_ten,
Tstrain_comp
                    , Pstrain_ten,
Pstrain_comp, Sstrain_ten, Sstrain_comp );
        return  TRUE;
    }

    BOOL
    CenterSectionBendClass::checkFail( Bloc
kBendClass huge& block )
    //  blockがモーメントで破壊するか否かの
チェックを行なう
    {
        isPlasticFail = FALSE;
        float   strain = calcK() *
( block.w - neutral_plane );
```



```
        if ( strain < 0 ){      //  引張り
応力が作用している
            if ( -strain >
block.tensile_strain )
                return  TRUE;
        } else {            //  圧縮応力が
作用している
            if ( strain >
block.comp_strain ){
                isPlasticFail = TRUE;
                return  TRUE;
            }
        }
        return  FALSE;
    }

    float
    CenterSectionBendClass::getInitialMomen
t()
    {
        float   min_moment = 1.0e+99;
        double  T = A * ( sum_E_ww - 2.0 *
sum_E_w * neutral_plane
                          + sum_E *
neutral_plane * neutral_plane );
        for( int i = 0; i < total_fiber;
i++ ){
            float   temp_M;
            if ( block[i].w >
neutral_plane ) //  圧縮側
                temp_M = T *
block[i].comp_strain
                                    /
( block[i].w - neutral_plane );
            else if ( block[i].w <
neutral_plane ) //  引張り側
                temp_M = -T *
block[i].tensile_strain
                                    /
( block[i].w - neutral_plane );
            else
                temp_M = 1.0e+99;

            if ( temp_M < min_moment )
                min_moment = temp_M;
        }
        return  min_moment;
    }
```

```
BOOL
    CenterSectionBendClass::analyzePiece( F
ILE* fCS, FILE* fE
                , FILE* fBend, double
stress_plane, double stress_shear )
{
    stress_plane_max = stress_plane;
    stress_shear_max = stress_shear;

    if
( !( CenterSectionClass::operator<<( fCS )
) )
        return  FALSE;

    total_fiber = total_column * total_row;
    Z = total_row * total_column *
total_column / 6000.0;
                            //  u,wがcm単
位でブロックの大きさもmmだから
    A = 0.01;
    dM = 0.01 * Z;          //  モーメント
の増分値が約0.01σということ

    block = new
BlockBendClass[total_fiber];
    if ( block == NULL )
        return  FALSE;

//  ブロックの情報を読み込んで、各ﾌﾞﾛｯｸの最
大ひずみ（断面垂直方向）を求める。
    sum_E_ww = 0.0;
    sum_E_w = 0.0;
    sum_E = 0.0;

    for ( int column = 0; column <
total_column; column++ ){
        for ( int row = 0; row < total_row;
row++ ){

    getLimitStrain( (BlockBendClass*)(block
+elem( column,row ))
                , fCS, fE, stress_t_max,
stress_plane_max, stress_shear_max );
                    //  曲げ、圧縮応力の両
方に対する最大ひずみを算出する。
            sum_E_ww +=
block[elem( column,row )].Eratio
                    *
block[elem(column,row)].w *
block[elem(column,row)].w;
```

```
            sum_E_w +=
block[elem( column,row )].Eratio
                                    *
block[elem(column,row)].w;
            sum_E +=
block[elem( column,row )].Eratio;
        }
    }

//  最初の中立軸を求める。こいつは作用モーメントに対する依存性はない
    neutral_plane = sum_E_w / sum_E;
//  最初の繊維が破壊するモーメントを算出する。
    Moment = getInitialMoment();

//  Moment 下で破壊する全繊維を求める。すべての繊維が破壊するまで繰り返す。

    sum_CFstress = 0.0;
    sum_CFstress_w = 0.0;
    fail_fiber = 0;
    fail_time = 0;
    fail_mode[0] = fail_mode[1] =
fail_mode[2] = 0.0;

    while( fail_fiber < total_fiber ){
        int     current_fail_fiber = 0;
        for( int i = fail_fiber; i <
total_fiber; i++ ) {
            if ( checkFail( block[i] ) ) {
                current_fail_fiber++;
                sum_E_ww -= block[i].Eratio
* block[i].w * block[i].w;
                sum_E_w -= block[i].Eratio
* block[i].w;
                sum_E -= block[i].Eratio;
                if ( isPlasticFail ){
                    double CFstress =
block[i].comp_strain * block[i].Eratio;
                    sum_CFstress +=
CFstress;
                    sum_CFstress_w +=
CFstress * block[i].w;

                } else {

    fail_mode[block[i].tensile_mode]++;
                    //  引張り破壊モードのみを記憶する。
                }
                if ( i != fail_fiber )
                    swap_block( block[i],
block[fail_fiber] );
                fail_fiber++;
            }
        }
        if ( current_fail_fiber == 0 ){
            //  破壊した繊維が一本もない場合にはMomentを増やす
            Moment += dM;
            fail_time = fail_fiber;
        } else {
            //  そうでない場合には、中立軸を再計算してそのMomentの大きさでもう
            //  一度検討を行なう
            if (  fail_fiber <
total_fiber )
                neutral_plane =
getNeutralPlane();
        }
        if ( calcK() < 0 )
            //  正負が逆転してしまった場合にはその時点で終了する
            break;
    }

    max_stress = Moment / Z;
    *this >> fBend;
    printf( "%d, %.5lg, %d ¥n",
piece_number, max_stress, fail_time );
    delete  block;
    return  TRUE;
}

class   BFailClass {

private :
    FILE*   fE;
    FILE*   fCS;
    FILE*   fBend;
    BOOLcan_open;
    CenterSectionBendClass  section;
    double  stress_shear_max;
    double  stress_plane_max;

public :
    BFailClass( char *fname, int
plane_condition, int shear_condition ) {
    strcat( fname, ".CS" );
```



```
    fCS = fopen( fname, "rb" );
    if ( fCS != NULL )
        can_open = TRUE;

    chgExtension( fname, "E" );
    fE = fopen( fname, "rb" );
    if ( fE == NULL )
        can_open = FALSE;

    chgExtension( fname, "B" );
    fBend = fopen( fname, "wb" );
    if ( fBend == NULL )
        can_open = FALSE;
    fwrite( &plane_condition,
sizeof( int ), 1, fBend );
    fwrite( &shear_condition,
sizeof( int ), 1, fBend );
  }
  ~BFailClass(){
    fclose( fCS );
    fclose( fE );
    fclose( fBend );
  }
  BOOLcanOpen()   { return can_open; }
  voidseek( double plane_max, double
shear_max ){
    stress_plane_max = plane_max;
    stress_shear_max = shear_max;
    fseek( fCS, 0, SEEK_SET );
    fseek( fE, 0, SEEK_SET );
    fwrite( &stress_plane_max,
sizeof( double ), 1, fBend );
    fwrite( &stress_shear_max,
sizeof( double ), 1, fBend );
  }
  BOOLanalyzePiece() {
    if ( section.analyzePiece( fCS, fE,
fBend
                               ,
stress_plane_max, stress_shear_max ) )
        return  TRUE;
    else
        return  FALSE;
  }
};

int     main( )
{
  static char fname[CHAR_MAX];
  int     plane_condition = 5;
  static double   stress_plane_max[]
        = { 1.0, 0.001, 0.00075, 0.0005,
0.00025 };
  int     shear_condition = 7;

  static double   stress_shear_max[]
        = { 1.0, 0.03, 0.025, 0.02,
0.015, 0.01, 0.005 };
  printf( "input file name : " );
  while( fscanf( stdin, "%s", fname ) !=
EOF ){
    static BFailClass   lumber( fname,
plane_condition, shear_condition );
    for ( int i = 0; i <
plane_condition ; i++ ) {
      for( int j = 0; j <
shear_condition; j++ ){

  lumber.seek( stress_plane_max[i],
stress_shear_max[j]);
        while
( lumber.analyzePiece() )
            ;
      }
    }
    printf( "input file name : " );
  }
  return  0;
}
```