博　士　論　文

機械学習を用いた
多方向光学式骨密度計測法の開発

Development of a Multidirectional Optical Bone Densitometry
Using Machine Learning Techniques

金沢大学大学院自然科学研究科

機械科学専攻

学籍番号 1824032019

三浦　要

指導教員　田中茂雄

令和 4 年 6 月 20 日

目　次

第 1 章

諸論

骨の強度は，骨量の指標となる骨密度(BMD)と骨の構造や微細骨折などからなる骨質の 2 つの要因によって決まり，BMD は骨強度のおよそ 70%を説明するとされている [1–3]．骨粗鬆症は，この骨強度が低下し，骨折リスクが上昇する骨疾患である [3]．骨量は成長期に増加し，20 歳代に最大骨量に達する．その後比較的安定に推移したのち，加齢に伴い減少する [4]．特に女性においては，閉経に伴い骨量が減少しやすくなる．骨粗鬆症による骨折は，生活機能や生活の質を低下させ [5,6]，長期的には骨折の有無にかかわらず，骨粗鬆症は死亡リスクを有意に上昇させる [7,8]．そのため，その対策が医療のみならず社会的にも重要な課題である．しかしながら，多くの場合，骨量の低下は自覚症状を伴わない [9]．加えて，骨粗鬆症は早期に介入しない限り薬物療法は困難である [3]．従って，早期介入のための，低 BMD の早期発見は有効な手段である．

現在 BMD 計測のゴールドスタンダードは，二重 X 線エネルギー吸収法(DXA)である（詳細は後述）[3,10,11]．DXA では，BMD は面積 BMD (aBMD, g/cm^2)として定量化され，その骨折予知能は優れている．[11–13] しかしながら，X 線を用いた装置は，大掛かりなものであり，被曝の危険性もあるため，骨量減少の早期発見を目的とした使用には限界がある．一方，定量的超音波（QUS）は，電離放射を使用せず骨を測定する唯一の方法である [3,14]．しかしながら，QUS のスコアは，現在の骨塩定量を基盤とする BMD 計測とは一線を画するものである．また，診断基準は定義されていない [1]．従って，骨粗鬆症早期発見のためシンプルで安全な方法で BMD を計測したいというモチベーションから，光を用いた BMD 計測法が研究されている [15,16]．

近赤外光を用いた BMD 計測は，骨粗鬆症の新しいスクリーニング手法になる可能性がある．近赤外光は優れた生体透過性を持ち [17]，その波長域では，BMD と光の散乱現象の間には強い関係性があることが知られている [18,19]．骨基質は，コラーゲン繊維マトリックスにカ

ルシウムを含むハイドロキシアパタイト結晶が堆積した組成を持つ．DXA を含む X 線での計測原理は，この結晶濃度の増加に対しX線吸収が増加することを利用したものである．骨基質に含まれるアパタイト結晶はまた，光子を強く散乱させる．竹内ら(1998)による初期の研究では，DXA により決定された BMD と透過光強度の間に強い相関関係があることが示された [20]．加えて，Ugryumova ら(2004)は，可視光から近赤外光の波長間で，散乱係数が緻密骨 BMD に対し相関することを示した．[18] このように，光の散乱は BMD と密接に関係することがわかる．

しかしながら，生体において光を散乱させるものは骨基質だけではない．生体のほとんどの骨基質は，少なくとも，表皮や真皮，皮下組織等の軟組織に覆われている．可視や近赤外の波長域では，軟組織での光吸収や散乱が強く影響し，定量的な骨計測を妨害する．例えば，Pifferi ら（2004）は，時間分解透過型分光法（TSR）を用いて踵骨を測定したが，個人差や軟部組織の複雑さにより，大きな測定誤差が生じることを示した [21]．また，Chung らは，in vivo で，近赤外光の透過率と aBMD が相関関係を有する可能性を報告したが，軟組織の影響を懸念している [16]．我々は以前，拡散反射光が形成する強度分布の傾きが BMD と相関を示すことを報告したが，皮膚厚に対して非線形な影響を受けることが分かっている [15]．これらの軟組織による影響は，個人によって，色や厚さが異なることに起因する．

軟組織の個人差による測定値のばらつき問題は，光を用いた骨計測の分野を長年身動きが取れない状態にしていた．本論の主題である「多方向光学式骨密度計測法」は，この軟組織による影響を軽減し，高い精度で骨密度が推定可能な方法の理論的な枠組み提案するものである．この第 1 章では，まず骨粗鬆症と骨密度について述べ，次に現在の骨密度計測法について紹介する．そして，現在光を用いた骨密度計測法の研究はどの程度進んでいるのか紹介し，本論で提案する多方向光学式骨密度計測法のコンセプトを説明する．最後に，本論の目的と，構成について述べる．

1.1　骨粗鬆症と骨密度

現在，我が国は世界で最も高齢化が進んだ国であり， 2036 年には 3 人に 1 人が高齢者になると試算されている [22,23]．このような社会の中で，骨粗鬆症を始めとする高齢に起因する病気が社会問題となっている．骨粗鬆症は現在国内において，その患者数は 1300 万人 [1] と推定されており，今後も増え続けることが予想される．

骨粗鬆症は，骨折の危険性が高い状態を指す言葉である．WHO では，骨粗鬆症は「骨量の低下と骨組織の微細構造の劣化を特徴とし，骨の脆弱性の増強とそれに伴う骨折リスクの増大をもたらす疾患」と定義されている [24]．つまり，骨粗鬆症は病気であり，骨折はその結果として起こる合併症の 1 つである．

骨粗鬆症の定義は，1991 年にコペンハーゲンで開催されたコンセンサス会議で提案され，1994 年からは WHO の定義が導入された．WHO では，人口における BMD と骨折発生率の関係から，表 1.1[24] に示すように，「正常，骨減少，骨粗鬆症，重症骨粗鬆症」を骨粗鬆症の診断カテゴリーと定義しており，診断カテゴリーは T スコアで分類される．T スコアは，若年成人の平均 BMD（YAM，young adult mean）からの差を標準偏差（SD）で表したものである．YMA が -1SD 以上を正常，-1SD 未満，-2.5SD 以上を骨減少症，-2.5SD 以下を骨粗鬆症と定義している．なお，骨粗鬆症の診断基準は，各国のガイドラインによって異なり，例えば，日本ではまず脆弱性骨折の有無で分類し，BMD で YAM の 70%以下を骨粗鬆症とする [1]．一方，タイでは WHO の基準で T-score を診断に用いている [25]．そのため，診断基準の取り扱いには注意が必要あるが，国際的には T スコアによる診断が標準であり，日本の診断基準も T スコアを使用したものに移行する動きがある．

表 1.1　WHO による骨粗鬆症の分類 [24]

Diagnosis	T-score
Normal	≥ -1.0
Osteopenia	< -1.0 to > -2.5
Osteoporosis	≤ -2.5
Severe osteoporosis	≤ -2.5 with fragility fractures

骨強度低下要因を説明する前に，骨リモデリングを説明する必要がある．古くなった骨は破骨細胞によって吸収され，骨芽細胞によって作られた新しい骨に補充される．この骨代謝の仕組みを骨リモデリングという [1]．破骨細胞，骨芽細胞，骨表面を覆う裏打ち細胞，骨基質中に存在する骨細胞などの細胞群の協調的な活動が主な原因となるプロセスである [26,27]．骨のリモデリングは，破骨細胞が骨を吸収することから始まる．成熟した破骨細胞は，骨基質との吸着面に酸を分泌してミネラルを溶かす．また，破骨細胞が特異的に生成するタンパク質分解酵素カテプシンKを分泌して，骨基質タンパク質を消化し，吸収腔を形成する．骨吸収が完了すると，骨芽細胞が骨表面に付着し，形成期が始まる．形成期には，骨芽細胞はI型コラーゲンやオステオカルシンなどの骨基質タンパク質を産生し，オステオイドを形成する．そして数日後，カルシウムやリンなどのミネラル成分の沈着による石灰化が起こり，吸収腔が新しい骨で満たされる．このとき，骨芽細胞の一部は骨基質に埋め込まれて骨細胞となり，残りは骨表面の裏打ち細胞となる．この一連の流れが約3ヶ月のリモデリング1サイクルとなり，骨格全体の3〜6%が常に再構築されている．しかも，リモデリングは成長期から成長完了まで，生涯にわたって繰り返される．その結果，骨組織は機械的な刺激に対して劣化を修復し，強度を維持する．また，骨組織は体内のカルシウムの恒常性を維持するための供給源としての役割も担っている．

骨の強度は，BMD と骨質という 2 つの要素で定義され，それぞれが様々な内的・外的要因に依存している [28]．BMD を維持するためには，各骨改造部位の骨吸収量と骨形成量が等しくなければならない [29]．骨基質タンパク質を合成する骨芽細胞の活性化や，石灰化に必要なカルシウムやビタミン D の欠乏は，BMD の低下を招く．また，骨吸収が異常に活性化し，骨形成によって吸収された骨量が十分に補充されない場合にも BMD は減少する．石灰化が不十分な吸収空洞の増加は，微小骨折，破断，連結不良などの骨質の劣化を引き起こす．骨形成期が数ヶ月間続くのに対し，骨リモデリング部位の骨吸収は数週間で終了するため，リモデリングの頻度が高くなると石灰化が不十分となり，骨強度が低下する．BMD は，若年の最大骨量不足，閉経によるエストロゲン不足，カルシウム，ビタミン D，K の欠乏，それに伴う副甲状腺ホルモンによる骨吸収の促進などにより減少する．また，加齢に伴う筋力低下や寝たきりの不動状態による力学的負荷の減少によっても BMD は減少する．

現在の骨粗鬆症診断は，主にBMDを計測することで行われ，BMDは年齢と共に減少していくことも近年認知され始めている．骨も他の臓器と同様に，加齢に伴い自然歴として形態

や機能が変化していく．すなわち，骨粗鬆症とそれに伴う骨折は，骨の「自然史の終末」である．図 1.1 に加齢に伴う骨量の推移を示す[4]．出生時の骨の重さは体重の 1/100 の 30g 程度であり，その後，小児期後半から思春期にかけて形態的な成長とともに量的な増加を見せる．20 歳前後で骨端軟骨は多くの骨格部位で骨化を終了し，骨量もピークを示す．その後，骨量は比較的安定するが，女性の場合，50 歳前後の閉経に伴うエストロゲンの急激な減少により，閉経後 10 年程度で骨量は著しく減少する．そして，骨量は骨減少症や骨粗鬆症と診断される領域に到達する．腰椎 BMD の減少を表すと，20 から 44 歳を 100%とした場合，45 から 49 歳で約 98 %，50 から 54 歳で 90 – 92 %，55 から 59 歳で 82 - 83% 減少する[30]．

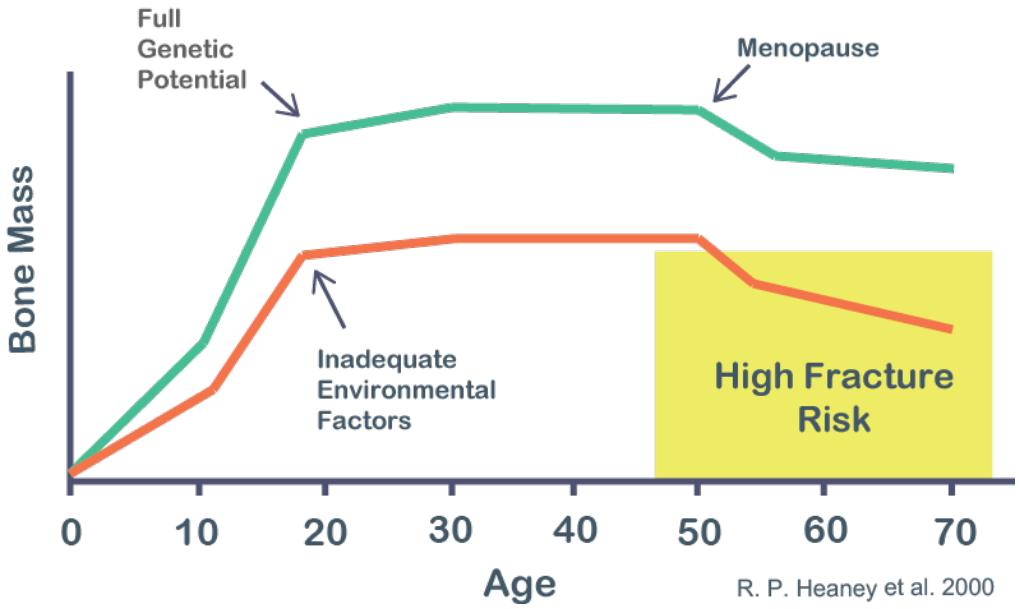


図 1.1　骨量の年齢による変化[4]

骨粗鬆症の治療法としては，薬物療法や，運動療法，食事療法等が存在する．薬物療法の代表的なものは，カルシウム薬やビスフォスフォネート薬である．カルシウム薬は，その名の通り，カルシウム摂取不足を補うためのものであり，その仕組みは次のように説明される[1]．カルシウム摂取量が不足すると，副甲状腺ホルモンの分泌亢進を介した骨リモデリングの亢進により，骨吸収が増加し，骨量が減少する．カルシウム薬を服用し，カルシウムが充足すると，副甲状腺ホルモンの分泌が抑制され，骨のリモデリングが低下し，骨吸収も減少する，という仕組みである．ビスフォスフォネートは，骨吸収を抑制するための薬剤である．骨に取り込まれたビスフォスフォネートは，破骨細胞による骨吸収の際に波状縁から得意的

に破骨細胞に取り込まれ，その破骨細胞はアポトーシス（自死）に至り，骨吸収機能が抑制される [1]．骨粗鬆症における運動療法は，健常者を対象にしたものが多く，ウォーキングや筋力訓練，バランス訓練等による，筋力強化や転倒防止の意味合いが強い [1,31–33]．また，運動により，骨密度が上昇することも知られている [34]．食事療法では，カルシウム摂取量のみを考えるのではなく，栄養素全体の摂取バランスを考えることが重要である [1]．もちろんカルシウムは，骨のミネラル成分の重要な構成栄養素であるが，腸管からのカルシウム吸収量には限界があり，ビタミンD等のカルシウム吸収を助ける栄養素の摂取が必要であるためである [33,35–37]．また，リン，食塩，カフェイン，アルコールの過剰摂取は控えるように心がける．

このように，骨粗鬆症の治療法は存在するが，薬物療法では副作用等が存在し [38–41]，高齢者や既骨折者の運動療法は困難を極める．カルシウム薬では，便秘や血管障害助長，ビスホスホネートでは，顎骨壊死や胃腸障害等の副作用が報告されている [1,41]．また，これらの薬物では，骨の急激な減少を抑止することはできるが，骨密度を増加させることはできない．そして，運動療法や食事療法でも，低下した骨密度をもとに戻すことは困難である．従って，骨粗鬆症において，予防は大きな意味を持つ．つまり，若い時期から骨密度計測により自分の骨の状態を知り，成長期における骨量を十分に増加させ高い骨量頂値を獲得することが重要である．

1.2 従来の骨密度計測法

最も初期の骨密度計測は，図 1.2 に示すように骨試料を切断して，（1）湿潤または焼成重量を試料総体積で割った見かけ密度，（2）湿潤または焼成重量を水置換により求めた骨基質の体積で割った実密度，の2種類の方法で算出しており定義が曖昧であった．Galante は 2 つの密度と圧縮強度を比較した結果，図 1.3 に示すように，見かけの密度と圧縮強度の間には強い相関関係があることが分かった [42]．つまり，骨粗鬆症の診断のために用いられる BMD は，この見かけの密度に相当するものである．

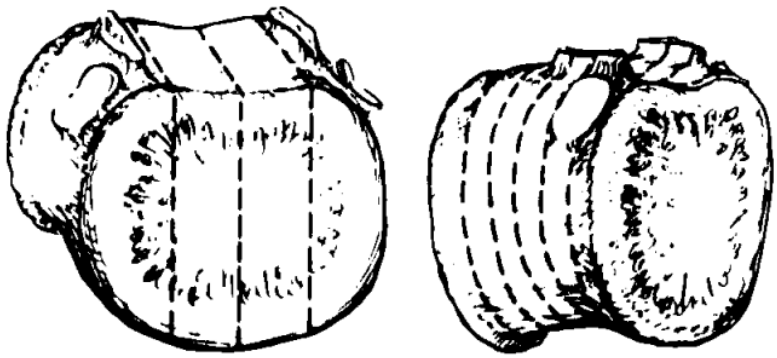


図 1.2　椎体の縦断面（左）と横断面（右）の切断方法図 [43].

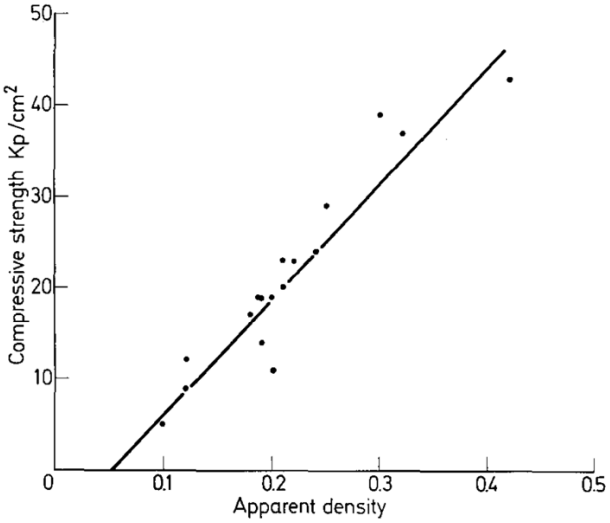


図 1.3　椎骨試料の圧縮強度の見かけ密度依存性 [42].

現在 BMD 計測のゴールドスタンダードは，2 重エネルギーX線吸収法 (DXA)である．DXA は，大腿骨骨折に対し高い骨折予測性能を有する．しかしながら，DXA は 2 次元画像しか生成できず，コストも高い．骨の 3 次画像は定量的 CT 計測法 (QCT) で撮影することができる．コスト面の問題を解決するため，日和見 CT を利用する方法が提案されている．日和見 CT とは，日常的に健康診断等で撮影する CT 画像から骨密度を推定する手法である．しかしながらこれらのX線を用いる方法は，大掛かりな装置や設備，また使用に際し資格等が必要である．現在市販されているX線を用いない唯一の方法は，超音波を用いる定量的超音波測定法 (QUS) である．QUS は DXA とは独立して骨強度を表す可能性はあるものの，現

在の BMD を基準にした診断では一線をかくするものであり，QUS による骨粗鬆症診断にコンセンサスは得られていない．また，BMD を定量化する精度も DXA 等の X 線を用いた方法と比較すると数段劣る [44]．本節では，DXA，QCT，QUS 法について説明する．

1.2.1 Dual-energy X-ray Absorptiometry: DXA

図 1.4 に DXA の外観を示す．^{153}Gd 等から 2 本の γ 線や X 線管からの高低 2 種類のエネルギー線を計測対象に照射し，その透過率の差から骨密度を測定する方法である．体組織に照射される 2 種類のエネルギーの光子計数率（photon/sec）は，空気中から組織を通過する際に指数関数的に減衰する．骨ミネラルの化学組成は一定であるため，低エネルギーと高エネルギーに対する骨ミネラルの減衰係数の比は不変である．このことから骨塩量 (Bone Mineral Content; BMC)，並びに BMD が求められる．

図 1.4　DXA の外観

DXA は，軟部組織の影響を分離して BMD を計測することが可能である．DXA の計測原理を数式で表すと次のようになる．狭い平行線束が物質を通過する場合，物質透過前後の X 線強度を I_0，I とすると次式のようになる．

$$I = I_0 \exp(-\mu\rho T) \tag{1.1}$$

ここで，μ は光子強度の減衰を示す質量減弱係数，ρ は物質の密度，T は厚さを示している．図 1.5 のように骨と軟部組織を X 線が透過する場合，式 1.1 より次式が得られる．

$$I = I_0 \exp[-(\mu_b \rho_b T_b + \mu_s \rho_s T_s)] \tag{1.2}$$

ここで，b，s はそれぞれ骨組織，軟部組織（筋肉，脂肪等)を意味する．高低 2 種類のエネルギーが骨と軟部組織を透過する場合，透過前後の強度は次式のようになる．

$$I_H = I_{0H} \exp[-(\mu_{bH} \rho_b T_b + \mu_{sH} \rho_s T_s)] \tag{1.3}$$

$$I_L = I_{0L} \exp[-(\mu_{bL} \rho_b T_b + \mu_{sL} \rho_s T_s)] \tag{1.4}$$

ここで，H と L は，高エネルギー，低エネルギーをそれぞれ意味する．式 1.3 ならびに 1.4 より，高低エネルギーの強度が得られるため，求める BMD は，

$$\rho_b T_b = \frac{\ln \frac{I_H I_{0L}}{I_L I_{0H}} - \rho_s T_s (\mu_{sL} - \mu_{sH})}{\mu_{bL} - \mu_{bH}} \tag{1.5}$$

となる．ここで，$\rho_s T_s (\mu_{sL} - \mu_{sH})$は，骨がない部位を計測すれば得られる．従って，質量減弱係数が分かっていれば，BMD の定量評価が可能である．

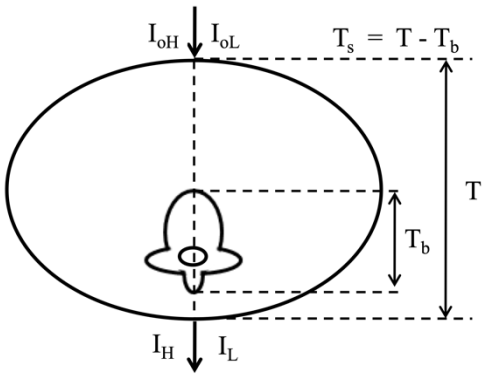


図 1.5　DXA の原理

前述の通り，DXA 法による骨粗鬆症の判定は，基本的に WHO の基準で行われている．DXA の関心領域 (ROI) は股関節，腰椎，前腕であるが，複数の部位が異なる診断基準を満たす場合もある．例えば，大腿骨頭頚部の T スコアが骨減少症で，腰骨が骨粗鬆症の基準を満たすことがある．この場合，3 部位（腰椎，大腿骨頚部，股関節全周）のうち最も低い値によって骨粗鬆症を分類するため [10]，この患者は骨粗鬆症と診断されることになる．このような有病率と危険因子の T-Score の不一致は様々な理由で起こる．1～2 個の椎骨の構造に異常がある場合やアーチファクトがある場合は，それらの値を除外し，少なくとも 2 個の T-

Score の平均を採用する [10]．患者の体重が使用する密度計の仕様（通常，約 300 ポンド）を超えるため，股関節スキャンも腰椎スキャンも使用できない場合，または，両側股関節置換，複数の椎体圧迫骨折，側弯，粗大な骨棘，骨外石灰化，腰椎の手術後などのアーチファクトが存在するため ISCD は前腕を測定することを推奨している [10]．そのため，橈骨の T スコアは，骨粗鬆症診断にも使用されることがある [45]．前腕は，重要な ROI の 1 つであり，閉経後の骨折で最も発生頻度の多い部位であるが，全米骨粗鬆症財団の治療ガイドラインや世界保健機関（WHO）の診断分類では，優先されていない [46]．

DXA で測定された低 BMD と新たな骨折の発生には高い相関がある [47]．骨折リスクは同一部位の BMD と最もよく相関する [11]．腰椎の BMD が 1SD 低下すると椎体骨折のリスクは 2.3 倍になり，大腿骨近位部の BMD が 1SD 低下すると大腿骨近位部骨折のリスクは 2.6 倍になる [11,48]．さらに，大腿骨近位部の BMD は，脊椎骨折を含むすべての部位の骨折を予測するのに優れている [49]．大腿骨近位部の BMD が 1SD 減少すると，あらゆる部位の骨折のリスクが 1.6 倍増加することを意味する [11]．このように，DXA は優れた骨折予測性能を示す．

DXA は X 線を用いた測定法であるため，様々な問題点がある．第一に，X 線被曝の問題である．DXA の被曝量は QCT などの X 線装置と比較して少ないが，妊婦に使用することができない．妊娠は骨量減少の原因の 1 つである．さらに，BMD を測定するためだけに使用される DXA 装置は比較的高価であり，ほとんどの発展途上国では広く普及していない [50]．さらに，DXA の利用は 50-64 歳の女性被験者や脆弱性骨折のある患者であっても一般的とは言えない [51,52]．DXA の利用率は，米国で 2006 年には，1000 人あたり 144 件，2012 年に閉経後女性で 1000 人あたり 110 件であった [51]．デンマークの 17,155 人の 65 から 81 歳の女性では，24.7%の DXA スキャンの経験があった [53]．このように，先進国においても，DXA の利用率は高くなく，ことさら発展途上国においては，特に地域の病院では利用率は低いと考えられる．

DXA と類似の方法としては，X 線吸収法 (RA) とデジタル X 線ラジオグラム法 (DXR) があげられる．RA は初期の骨密度測定法の一つであり，X 線画像で骨を評価する．そして，DXR は RA をデジタル化したものである．RA や DXR は X 線被曝量が多いため，一般的には手などの抹消で行われる．RA や DXR で計測された BMD は，DXA による計測に対し，r = 0.65 程度の相関を有する [54]．

1.2.2 Quantitative Computed Tomography: QCT

QCT は mg/cm^3 と 3 次元 BMD を算出する唯一の方法である．DXA で測定される 2 次元の面積密度とは異なり，QCT は 3 次元の骨密度を測定することができる [55]．QCT では，X 線 CT を用い，骨塩量として炭酸カルシウムやハイドロキシアパタイトのファントムを数種類の濃度で測定対象者と同時にスキャンする．そして，関心領域の CT 値とファントムの CT 値を比較することで，骨密度を標準物質の濃度（mg/cm^3）に変換する．図 1.6 に QCT の外観を示す．

図 1.6　QCT の外観 [http://www.qct.com/].

QCT では，3 次元画像が得られるため，様々な力学的解析が可能である．しかし，測定時間が長いこと，X 線被曝量が多いこと，関連コストが高いことなどから，骨粗鬆症の主要な診断手段としては限界がある．この被曝の問題を軽減するために提案されたのが，Peripheral QCT（pQCT）である．pQCT は身体の末梢で測定するため，臓器への放射線被曝の量を低減することが可能である．さらに，より高い解像度で画像を取得できる高解像度 pQCT（HR-pQCT）も開発された．HR-pQCT は，3 次元 CT 画像を用いた力学解析などの分野でも注目されている．

近年，コスト制約の軽減のため，日和見 CT（oCT）が注目されている．骨の健康のための DXA スクリーニング率は低く [56,57]，通常，低エネルギー骨折が発生した後にしか実施さ

れない．一方，一般的な CT スキャンの稼働率は非常に高い．例えば，カナダでは，年間数百万件の CT スキャンが実施されており，そのうち 54.5%は骨粗鬆症性骨折部位で実施されている [56]．oCT とは，他の臨床的理由で最初に得られた CT スキャンの二次解析として，骨の健康状態を評価することを意味する．二次解析であるため，追加のスキャン費用や放射線被曝の制限もない．CT 撮影は一般的な方法であり，CT を用いた骨格評価法は十分に開発されている．しかし，骨折予知に関する知識不足と方法論の問題から，oCT を用いた BMD の定量的評価に関する研究はあまり報告されていない．

1.2.3 Quantitative Ultra-Sound: QUS

QUS では，図 1.6 に示すように，踵を対向する 2 つのスタンドオフで挟んで計測する．2 つのスタンドオフには超音波振動子が内蔵されており，一方のスタンドオフ側の超音波振動子から送信された超音波信号は踵を通過し，他方の超音波振動子で受信される．QUS は，主に超音波の音速（SOS）または広帯域超音波減衰（BUA）を測定することにより骨密度を評価する方法である．図 1.7 に超音波の速度（A）と超音波の減衰（B）に関するいくつかの QUS 変数の計算方法を示す．SOS はインパルス信号が移動した距離（プローブ間の距離）とその距離を信号が移動するのにかかった時間の比として計算される．図 1.7 (A) の AD-SoS は，2mV の閾値を持つ振幅依存の速度を反映し，BTT は受信した信号の最初のピークが最大レベルに達する時間と，2 つのトランスデューサーの間に骨がなく軟組織のみが存在するはずかどうかを測定した時間との間隔である．物質を伝わる音速 c は，ヤング率 E と密度 ρ で次のように記述できる．

$$c = \sqrt{\frac{E}{\rho}} \tag{1.6}$$

カーターの式（1977）[58] によれば，骨のヤング率は質量密度を用いて以下の式で表される．

$$E = 2875\rho^3 \tag{1.7}$$

つまり SOS は，音速が骨密度に比例することを利用し骨密度を計測する．一方，BUA は周波数変化に対する振幅減衰比であり，魚群探知機と同じ原理に基づいている．図 1.7 (B) に

示すように，BUA は，BUA = Δdb / ΔMHz という式による周波数に対する減衰の回帰直線の傾きである [59].

図 1.6　QUS の外観.

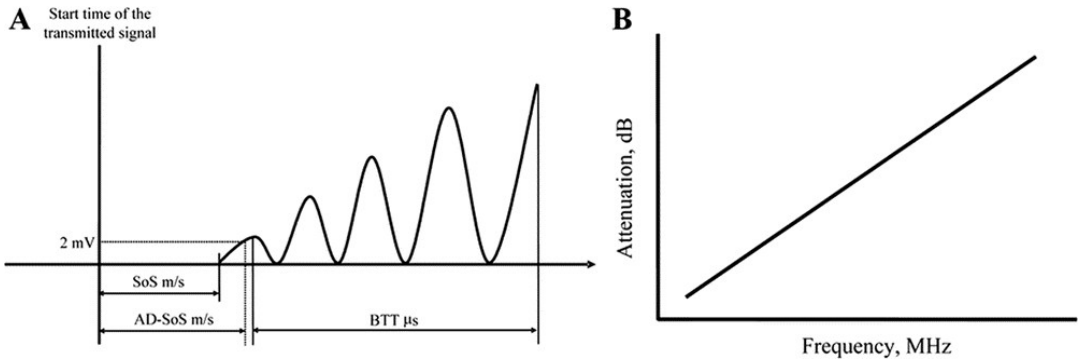


図 1.7 超音波の速度（A）と超音波の減衰（B）に関する QUS 変数の計算方法 [59].

QUS の臨床的な研究としては，測定方法の特性上，骨靭性，骨折リスクなど BMD よりも骨質の評価を重視したものが主流であるが，BMD を重視した研究も行われている．表 1.2 に QUS と骨密度の Pearson の 相関係数 r をまとめたものを示す．このように，QUS の測定値は測定位置や他の骨密度との相関は，決して高くないことがわかる．また，Nayak のメタアナリシスでも，BUA と SOS の骨粗鬆症判別性能はあまり高くないことが分かっている [44] .

表 1.2　異なる部位における QUS パラメータと BMD の相関 [60].

	BMD femoral neck	BMD spine	BMD calcaneus	BMD distal radius
BUA calcaneus	0·47 (0·30–0·87) 8042 (25)	0·47 (0·32–0·83) 9212 (28)	0·66 (0·53–0·74) 1221 (7)	0·37 (0·29–0·42) 1226 (4)
SOS calcaneus	0·52 (0·35–0·73) 4981 (14)	0·53 (0·33–0·77) 5545 (15)	0·47 (0·44–0·50) 148 (2)	0·33 170 (1)
SI calcaneus	0·61 (0·45–0·77) 3193 (9)	0·62 (0·43–0·80) 4027 (12)		0·37 170 (1)
SOS patella		0·36 (0·34–0·41) 346 (3)		0·36 (0·32–0·43) 1428 (2)
SOS tibia	0·40 (0·31–0·47) 527 (2)			0·63 307 (1)
SOS distal radius				0·68 313 (1)

超音波を使用するため，QUS による測定・解析時間は短い．また，現在市販されている装置の中で唯一X線を使用せず，コンパクトで安全な測定が可能である．ただし，測定部位である踵にジェルを塗布して測定する必要があり，準備も含めた計測に使用する時間は長くなる傾向にある．

1.3　これまでの光を用いた骨密度計測法の研究

骨量低下の早期発見のため，安全かつ非侵襲的に骨密度の簡便なスクリーニングを実現する装置の開発が望まれており，光を用いた計測方法が現在注目を集めている．近赤外光は生体内の情報を非破壊的に得る手段として，様々な医療分野で利用されているが，皮膚層等の生体組織に影響されやすく正確な情報が得るのが困難なことが問題である．本節では，光をもいた骨密度計測法に関するこれまでに実施された研究を説明する．

光を用いた骨密度計測法はまだ発展途上な技術である．光を用いた骨密度測定の最初の報告は，Takeuchi らによるものである [20]．竹内は，図 1.8 に示すように，牛の骨ブロックを用いて，透過光の減衰が BMD に強い相関を持つことを示した．竹内らの検証の後，

Ugryumova ら [18] によって，透過光の減衰と BMD の相関関係は，骨による光吸収よりも，散乱による影響が大きいことが分かっている．また，竹内らは Patterson らの時間分解法 [61] を用いて，近赤外パルスレーザーの時間応答にて骨とその他の軟組織に分離して測定できることを示した．この結果は，軟組織の影響を考慮せずに骨密度を測定できることを示唆しているが，現実問題として時間分解法を用いた軟組織影響の分離は困難である．Pifferi ら（2004）は，時間分解透過型分光法（TSR）を用いて踵骨を測定したが，軟部組織の複雑さにより，骨密度の定量計測は困難であった [21]．Pifferi らよりも大胆な方法で光を用いた BMD 計測を試みた例も存在する．Chung ら [16] は，近赤外光を用いてヒト橈骨における透過光量と DXA によって決定された BMD を比較した結果，強い相関（r = -0.901）を示すことを報告した．Chung らの方法は，光を検出するための COMS カメラと 850 nm 波長 28 mW のレーザー，腕を支持するための台座のみで構成されており非常にシンプルである．しかしながら，検証は，10 人の非常に小さなサンプルサイズで行われていた．また，Chung ら自身も指摘しているように，このシステムでは軟部組織や個体組織の大きさの違いは考慮されていない．実際，Chaichanakol の報告 [62] によれば，近赤外光の透過光は試料の厚さに大きく影響されることが示されている．

(A)　　(B)

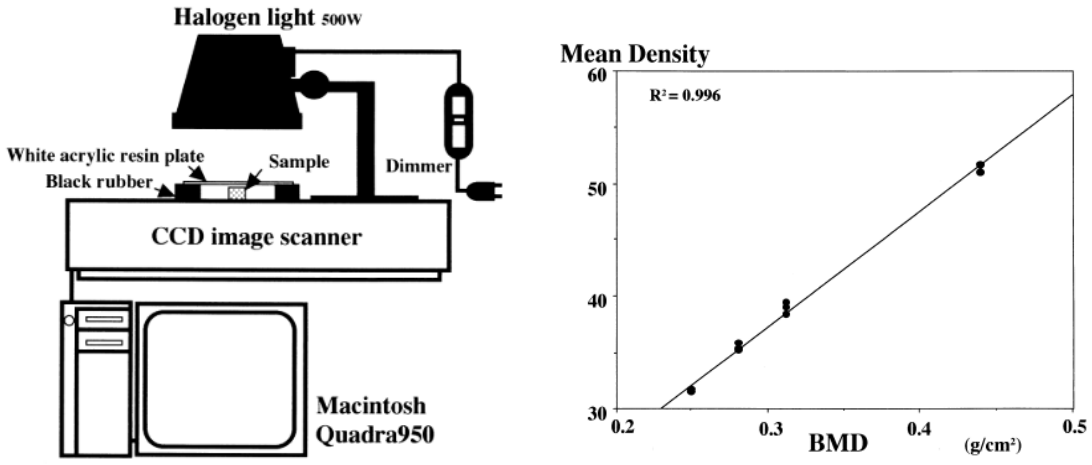


図 1.8　竹内(1997)らによる初期の光を用いた骨密度計測 [20]．(A)連続インコヒーレント光測光システム図．(B) 左図から得られた BMD と平均密度の関係.

我々も過去に，いくつかの光学式骨密度計測法を提案している．１つ目は，吸光度比を利用した骨密度計測法であり[63]．２つ目は，反射散乱光強度分布を利用した方法である[64]．３つ目は，２つ目の方法の皮膚影響の軽減効果を強化した手法である[65].

吸光度比を利用した骨密度計測法では，波長域によって異なる骨の吸光度の差を用いて骨密度を評価する．図 1.9 に示すように，骨突出部の一側面より２つの LED から波長の異なる近赤外光を交互に照射し，骨内の吸収・散乱により減衰した光を反射側面に設置したフォトダイオード(PD)により検出する．PD での検出強度と LED の照射強度から吸光度を求め，2 波長間の吸光度比および差を算出し，骨密度評価パラメータとした．この方法は，皮膚が無い場合に限り，良好な計測値が得られるが，皮膚がある場合，皮膚の情報も含んだ信号を検出してしまうという問題がある．

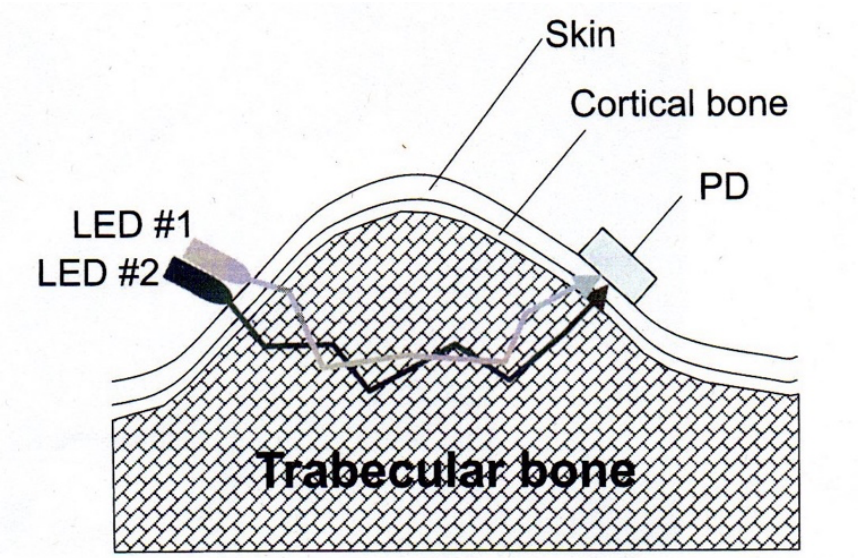
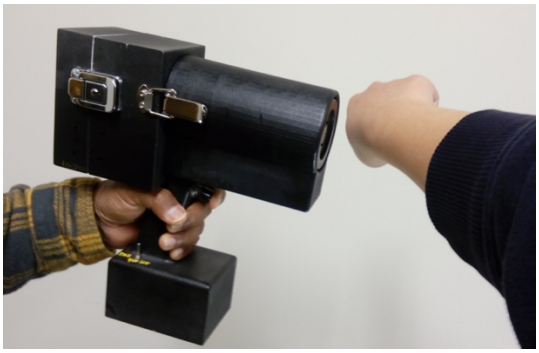


図 1.9　近赤外光を用いた骨密度計測装置[63]

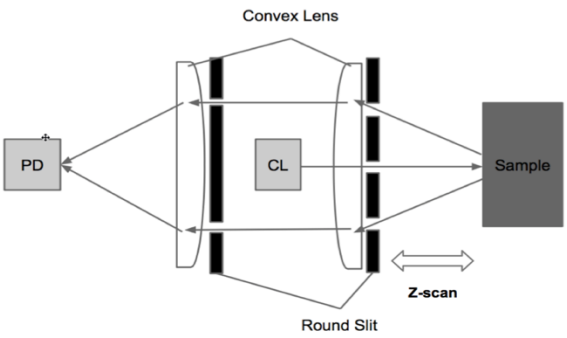
反射散乱光強度分布を利用した方法では，生体内にコヒーレント光（CL）を入射した際に，皮膚表面上で形成される光の強度分布を用いる．この光強度分布は，骨密度に依存しており，図 1.10 (A)に示すように，骨密度が高い場合，光の入射位置から半径方向に急激な強度減少を示し，低密度の場合は，緩やかな強度減少を示す．つまり，強度分布の変化の度合い（傾き）を計測することで，骨密度を評価できる．またこの方法では，準直進光のみを限定的に計測することによってこの問題の解決を試みている．図 1.10 (B)は，光式骨密度計測

装置の構成図である．本装置は，レーザー光を一定強度で組織に照射し，組織内で散乱・吸収により減衰され帰ってきた光を二枚の平凸レンズにより集光，そしてフォトダイオードにより光強度を計測する．この時，両レンズ前面に 2 枚の円形スリット設置し，それらを経由させることで，光路を限定し，準直進光のみを検出する．そして，レンズ及びスリットと試料間の距離を変化させることで，図 1.10 (C)に示すように，異なる半径位置での準直進光を検出し，試料表面の強度分布を得る．また，この 2 枚のスリットによる準直進光の検出は，皮膚による影響の除去という機能も持ち合わせている．皮膚による光散乱はスリットによって除去されるため，皮膚内をほぼ直進する光のみが検出されることになる．そのため，得られた強度分布は，皮膚の厚さによらず，同じような傾きを示すことになる．この方法では，ファントムを用いた実験では，光強度分布の傾きと骨密度は正の相関を示し，皮膚厚が 1 から 2mm の範囲では皮膚厚に影響を受けにくい骨密度計測が可能であった（図 1.11）．しかしながら，皮膚厚による影響の除去性能は限定的であった．

(A)



(B)



(C)

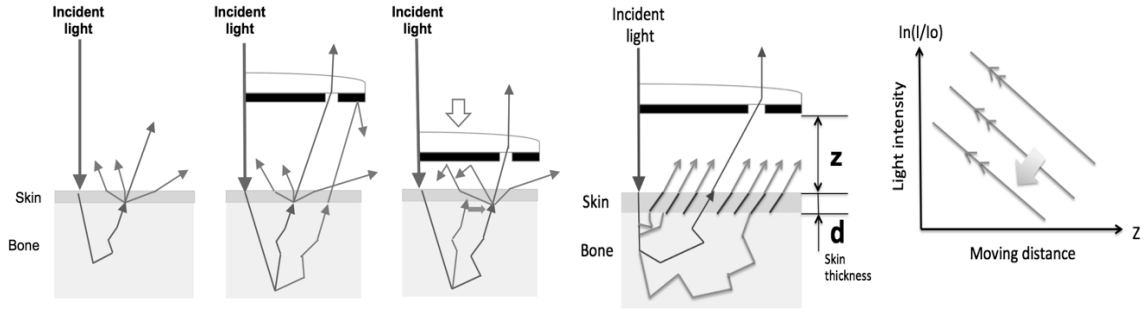


図 1.10　(A) 装置の外観，(B) 装置の光学系及び (C) 計測原理

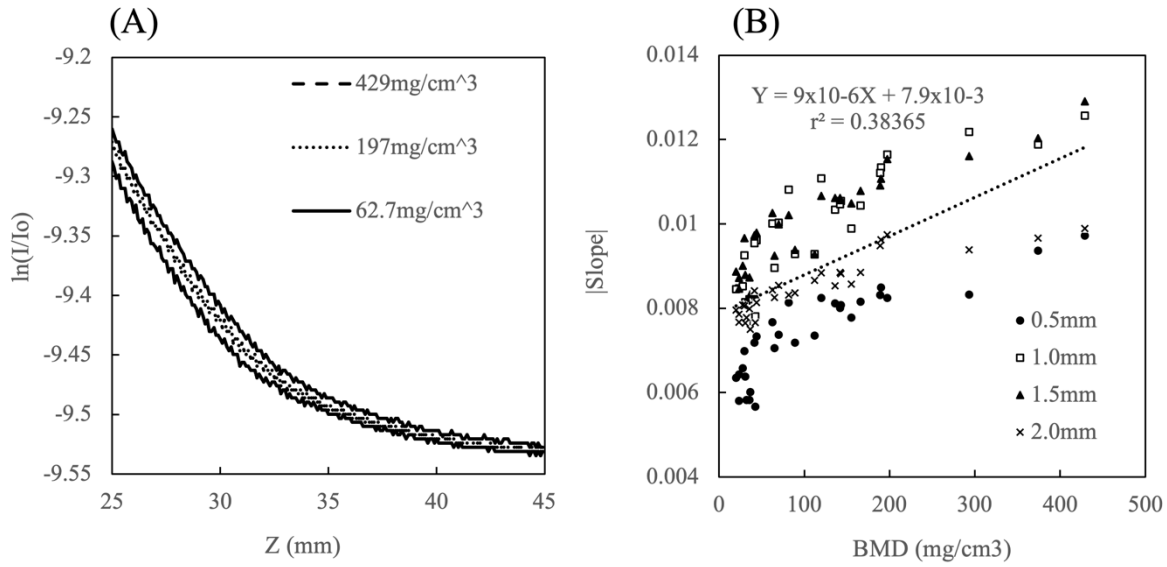


図 1.11 （A）厚さ 1mm の皮膚ファントムを用いた密度の異なる 3 つの骨ファントムの光強度分布．（B）骨ファントムの密度（BMD）と皮膚の厚さが異なる場合の光強度分布の傾きの関係．[15]

反射散乱光強度分布を利用した方法では，計測可能皮膚厚の範囲が 1 から 2 mm と狭いことが問題であった．この問題を解決するため，波長の異なる複数のレーザーを用いた測定方法が提案された．図 1.12 に提案された光学系を示す．生体組織の光学特性は，波長によって異なる．緑色光などの可視光は，近赤外光に比べて皮膚への光侵入深さが浅い[66,67]．つまり，緑色光のような近赤外光よりも短い波長の光は，皮膚の厚さの違いによる光学的距離の変化に対して鈍感であると考えられる．つまり，図 1.12 のスリット間隙に配置された，近赤外光 (LD2) と緑色光 (LD3) の差を測定することで，皮膚の厚さを測定する．そして，測定した皮膚厚さにより，皮膚厚による計測バイアスを補正する．ファントムを用いた実験では，この方法により補正された計測値は，骨密度に対し高い決定係数を示した（図 1.13）．このように，提案された手法は，高い精度で骨密度を定量評価できる可能性がある．しかしながら，光学式骨密度計測法を困難にする要因は，皮膚厚さだけではない．例えば，個人によって異なる皮膚色や複数層で構成される皮膚の影響も実際には考慮する必要がある．また，我々は，計測に反射散乱光を用いてきたが，実際の生体で，真皮や皮下組織，緻密骨の下にある海綿骨の情報を捉え切れているかは謎である．実際に，本法を用いたヒト骨密度の計測と DXA の比較では，十分な相関が得られなかった[68]．

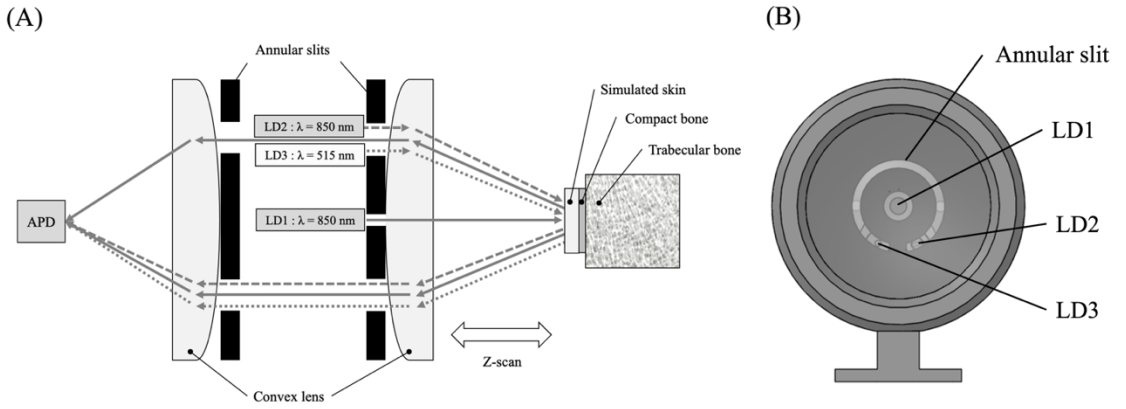


図 1.12　2 波長レーザーにおける皮膚影響オフセットのための光学系．APD：アバランシェフォトダイオード，LD：レーザーダイオード．(B)光学系の正面図.

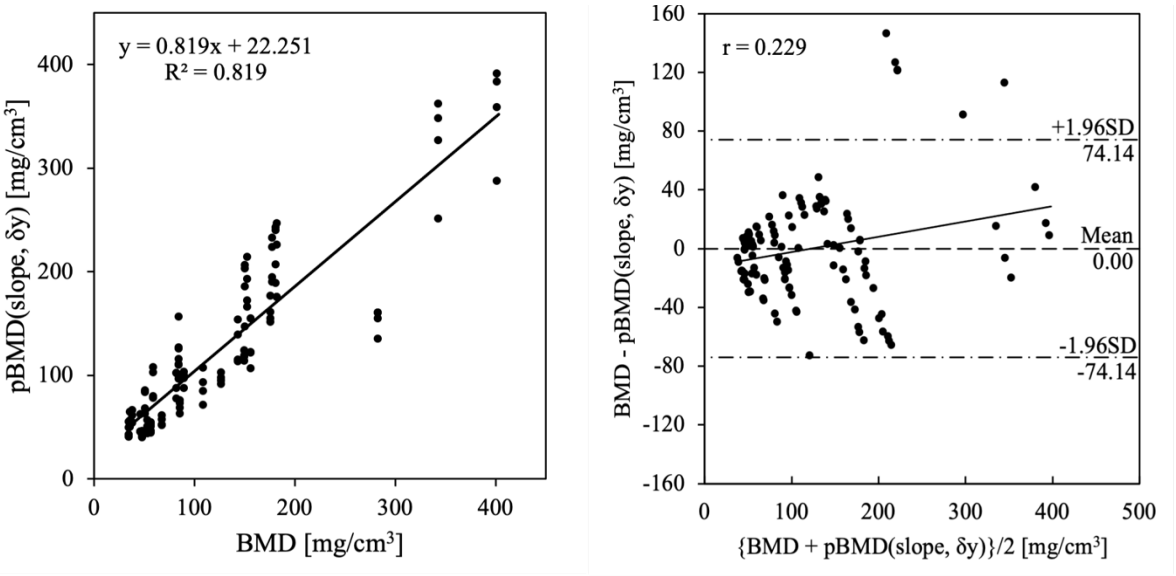


図 1.13　補正された計測値と μCT によって決定された骨密度の比較（左）とその誤差（右）.

上記に示すように，光を用いた骨密度計測法の研究では，これまで様々な手法が提案されてきた．しかしながら，正確に骨密度の定量評価が可能な，光を用いた計測法は未だ確立されていない．この分野の課題は，いくつか考えられる．１つ目は，個人によって異なる皮膚の色や厚さの影響を軽減する方法が存在していないことである．これまでも見てきたように，光を用いた非侵襲骨密度計測では，真皮や皮下組織等，多層構造を有する皮膚の色や厚さに

影響を受ける．2 つ目は，拡散反射光，拡散透過光のみでは，骨密度を推定するのに十分ではない可能性である．拡散反射光は，骨まで光が到達していた場合，軟組織層も確実に通過し，おそらく骨よりも軟組織に影響を受ける．拡散透過光は確実に骨を通過しているが，X線のように生体内を直進せず生複雑な経路を辿って透過に至るため，これだけでも骨密度の推定は難しいだろう．それでは，拡散反射光と拡散透過光を組み合わせたらどうだろうか．さらに言及すると，拡散反射光と拡散透過光はそれぞれ後方方向ならびに前方方向で観測される光であり，実際に有限な大きさを持つ生体組織では，側方方向に出てくる光も存在する．そしてそれら異なる方向，異なる位置で観測される光は，様々な階層に存在する組織から異なる影響を受ける．つまり，前方，後方，側方で観測される光を組み合わせることで，骨密度の推定精度を向上させることが可能かもしれない．これは，本論で提案する「多方向式骨密度計測法」のコンセプトである．

1.4　多方向光学式骨密度計測法

生体内に光子を照射すると，体内の様々な組織の影響を受け，あるものは体外に放出される．それらの体外に放出された光子は，光子が通過した経路に存在する組織の情報を持つことになる．光子が通過した平均的な経路は，光源からどの程度離れた位置から観察されたかによって，あらかた推定できると考えられる．たとえば，光照射位置から生体を挟んで裏側で観察される光子は，生体内を最短距離で通過した光子である．このような理論は，生体直進性の強いX線の派長域では成り立つ．しかしながら，近赤外の波長を持つ光子は良好な生体透過性を示すものの，生体内で拡散され，ランダムで複雑な経路をたどり，様々な組織に多重の影響を受ける．したがって，体外に放出される近赤外波長の光子は，個人によって異なる生体の光学特性ならびに構造特性の情報をもつものの，その情報は，生体内での光拡散によって複雑で非線形に絡み合ったものとなる．

光を用いた骨密度計測を困難なものとしているのは，軟組織層の個人差による計測値の非線形な変化である．骨を覆う軟組織は，外側から一般的に肌と呼ばれる表皮や真皮，一つ下の階層には主に脂肪分で構成される皮下組織が存在する．そして，これらの光学特性や厚さ

は，個人によって異なり，光輸送を複雑なものとする．従って，骨密度値は一定であっても，軟組織の個人差によって光学測定値は，複雑な変動を示すようになる．

軟組織の個人差による光学測定のばらつきによる問題を解決するために本研究では，次のようなアプローチを提案する．まず，空間的に分解された定常拡散光を媒質に対して 3 方向で測定する．ここで，3 方向とは，弾道光の照射方向に対して，後方，前方，側方の 3 方向である．この方法は，光が計測される方向と骨による光学的散乱の間には，機能的な関係があるという考えに基づいている．加えて，反射光を使用した場合，組織への浸透深さは，光源と検出器の間の距離の半分にほぼ等しいと考えられており [69]，前方および側方方向に観測される光には，そこに至るまでの光路の情報がすべて反映される．つまり，異なる方向に空間分解された拡散光は，光が通過するすべての組織の構造や光学特性に関する情報が混在しており，測定する場所によって影響の度合いが異なると考えられる．次に，異なる位置で取得した拡散光と BMD の関係を，機械学習の手法を用いて一般化する．機械学習とは，データの背後にあるルールやパターンをコンピュータが自動的に学習・発見するデータ解析手法である [70]．本研究では，この機械学習技術よって，軟組織層の個人差によって形成する非線形な光計測値の変動の解決を試みる．機械学習モデルが十分な汎化性能を発揮するためには，軟組織や BMD の個人差を表すのに十分な分散のある集団からのデータが必要である．

機械学習によって，軟組織の個人差に影響されにくい BMD 予測モデルの開発を実現するには，いくつかの課題が存在する．最も大きな課題は，どのようにデータ収集するかである．機械学習の主なコンセプトは，データの中に存在するルールを計算機自身に発見させることであり，汎化性を高めるためには，大量のデータを必要とする．この制約は，新規医用計測方法の研究開発への導入では特に困難な問題となる．例えば，実際に生体を使用してデータを集める場合，膨大なコストと倫理的な障壁が発生する．機械学習を計測原理のベースに据える場合，モデル開発のためデータの収集は必須になるが，誰も見たことのないものを試すには生体計測はリスクが高いと判断されるのである．一方，コストや倫理的な問題を回避するため，実験モデルを作成してデータを集めることを考える．しかしながら，データを集めるため，数百から数千種類の実験モデルを作成することは，時間や労力，精神的な面から見てもあまり現実的とは言えない．そこで第 3 の選択肢として，本論では，数値シミュレーションによるデータの収集を実施する．数値シミュレーションには，仮想と現実の差による批判が常につきまとうものの，その制限を認識して用いる分には現象を深く洞察する助けとな

る．そして，数値シミュレーションには現実にはない利点がいくつもある．例えば，値を設定するだけで，無尽蔵にデータを集めることができ，条件や環境を完全に制御することが可能である．これらの利点は，機械学習モデルを用いた理論の検証に適していると考えられる．

骨組織を再現した光輸送シミュレーションを実施するためには，主に 2 つの技術的な課題が存在する．１つ目は海綿骨組織のモデリングであり，２つ目はモンテカルロシミュレーションの３次元構造への拡張である．モンテカルロシミュレーションは，生体組織等の光拡散媒質内の光輸送を計算するためのゴールドスタンダードモデルである [71]．

海綿骨組織のモデリングは，光輸送シミュレーションに海綿骨特有の形状からくる光学特性と，海綿骨の骨密度を再現するために必要である．骨の光散乱特性と骨密度の間には，Ugryumova らの調査により，正の相関関係があることがわかっている（図 1.14）[18]．最もシンプルなアイデアでは，散乱係数を変更することで，密度の異なる骨を再現することである．しかしながら，図 1.14 の縦軸を参照すればわかるように，この相関関係は，およそ 1 g/cm^3 以上の骨密度に限られている．この骨密度は，一般的に緻密骨の範囲であり，海綿骨の骨密度は 0.08 から 0.3 g/cm^3 [72] であり，図 1.14 の結果は海綿骨には適用できない．また，図 1.14 中に記載された式を用いて，海綿骨の光散乱係数を算出すると負の値となる．加えて，海綿骨は，3 次元の複雑な構造をもつため，そこを通過する光がどのように影響されるかは不明である．これらの理由から，海綿骨組織全体に密度によって異なる単一の散乱係数を与えることは困難である．一方，海綿骨の骨基質部分ほぼ均一な媒質を仮定でき，単一の散乱係数を与えることができる．言い換えると，骨梁構造のモデルさえ作成してしまえば，海綿骨は骨基質とそれ以外の均質を仮定できる組織（骨髄）で構成されるため，それぞれに光学特性を与えることができる．また，骨組織全体の体積に対する骨基質の体積を定義するだけで骨密度を再現できる．

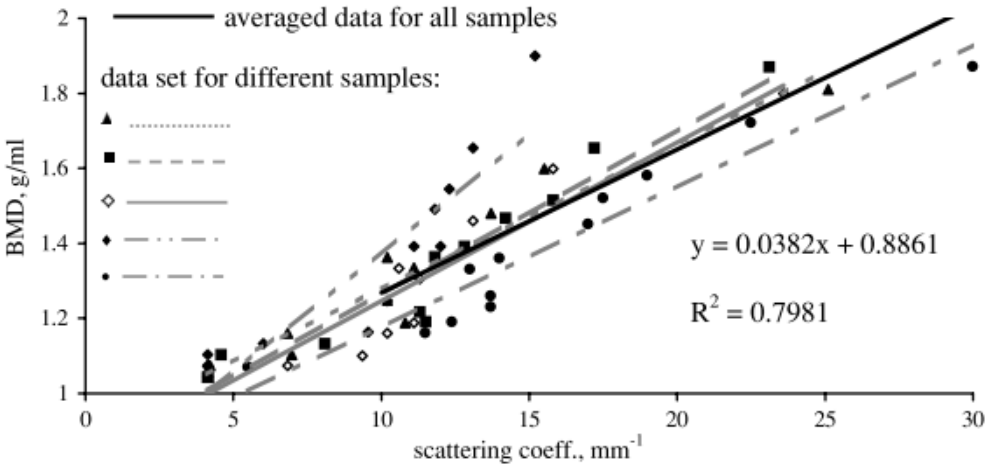


図 1.14　骨密度と散乱係数の関係 [18]

本論では，海綿骨モデリングに，Alan Turing の反応拡散モデル [73] を用いる方法提案する．この反応拡散モデルは，提案者の名前をとり，チューリングモデルとも呼ばれている．チューリングモデルは，生物学におけるパターン形成過程を記述する数学的理論として広く用いられており，骨においては石灰化や発生の初期段階で観察される [74,75]．チューリングモデルを用いる利点は，海綿骨構造を一般化でき，ランダムな構造をもつモデルを大量に生成できる点にある．そして，生体を計測したデータを使用したいため，コスト面，また倫理的な障壁も存在しない．本来海綿骨は，計測される位置によって密度が大幅に異なる [76]．しかしながら，このようなばらつきが存在する場合，不確実性の高い要素が多く入り込むため，初期の理論的な検討において望ましいことではない．したがって，海綿骨構造を一般化できることは，理論的な検討におけるシミュレーションでは有用である．また，チューリングモデルでは，乱数を用いて初期分布を決定する．したがって，乱数のパターンによって無尽蔵に様々な形状を持つモデルを生成可能である．これは，チューリングモデルによって生成された海綿骨形状が，海綿骨を一般化する妥当な形態を保つ限り，機械学習モデルを訓練するためのデータ生成に有用である．

海綿骨モデルは 3 次元構造をもつ必要があるが，多層均質媒質を想定した既存のモンテカルロシミュレーションアルゴリズムでは，この 3 次元構造を表現することができない．したがって，Wang らによる多層組織における定常光輸送のモンテカルロモデル（MCML）[77,78] を 3 次元的に拡張したボクセルベースのモンテカルロモデル（VMC）を開発した．ボクセルは，3 次元画像の立方体要素を示す言葉で，ボクセルベースのモンテカルロモデルは，ボ

クセルサイズを十分小さくすることで，自由局面境界に近い結果を導出することが可能である．また，チューリングモデルで生成される海綿骨モデルは，正六面体要素で構成されるため，海綿骨モデルに直接，計算を実施することが可能である．

その他の課題としては，機械学習モデルを訓練するため，現実的な範囲の色や厚さの分散をもつ皮膚を再現することがある．これは，過去の文献を調査し，これらのパラメータの範囲を特定し，その範囲の中でランダムに決定することで対応した．海綿骨モデルに加え，これら軟組織を持つモデルを合成生体組織モデルと名づけた．

1.5 研究目的と本論の構成

本研究の目的は，拡散光を学習データとした機械学習予測モデルをモンテカルロシミュレーションにより開発し，検証することである．モンテカルロモデルは，橈骨最遠位端を仮定した生体組織モデルの 3 方向から取得された，空間分解定常拡散光データをシミュレートするものである．本手法が十分な精度を有していれば，簡便かつ安全な骨密度測定に利用でき，骨粗鬆症の早期発見に寄与することが期待される．

本論は，全 5 章の構成となっており，前節で説明した 2 つの技術的な課題について，第 2 章，第 3 章で説明し，機械学習のモデリングについても第 4 章で説明した．本論の構成を次に示す．

第 2 章では，チューリングモデルを用いて生成した仮想海綿骨モデルの検討を行う．まず，Miura と Mani の論文[79]を元に，チューリングモデルを用いた骨梁パターン生成のための，支配方程式およびその解法について説明する．次に，チューリングモデルを用いた海綿骨モデルの生成を行う．最後に，生成された海綿骨モデルに 3 次元骨形態計測を実施し，モデルを評価し，シミュレーションでの使用について妥当性を検証する．

第 3 章では，ボクセルベースのモンテカルロ法 (VMC) について述べる．まず，VMC のアルゴリズムについて説明する．VMC は，MCML と比較して，3 次元構造モデルを扱えることが大きく異なる．つまり，六面方向にある境界のどこに光子が当たるかを簡単に計算でき

るようにする必要がある．また，VMC では，計算を高速化するため，GPU を用いた超並列計算を適用している．これについても説明する．次に，過去に他の研究者によって実施されたモンテカルロ法と VMC を比較し，コードの妥当性を検証する．ここでは，主に Wang の MCML と比較し，GPU を用いた計算により，どの程度高速化されるかも調査する．加えて，第 3 章では，海綿骨や皮膚を考慮した際の初期的なシミュレーションを行う．ここでは，3 方向に放出された光子がどの程度骨や皮膚の情報を保ちうるかについても考察する．

第 4 章では，モンテカルロ法でシミュレートされたデータを用いて，機械学習モデルを作成する方法並びに，生成された機械学習モデルの BMD 推定精度について述べる．まず，シミュレーションにより機械学習モデル訓練用のデータを生成する方法について説明する．シミュレーションデータは，第 2 章で説明した海綿骨モデルから合成生体組織モデルを生成し，それに対し第 3 章で説明した VMC を実施することで生成する．次に，シミュレーションデータと BMD を結びつけるための機械学習モデルについて説明する．機械学習モデルは，4 つの異なる構造をもつモデルを訓練し，最も予測性能が高いものを採用する．加えて，3 つの方向で取得されるデータとその組み合わせが骨密度の予測精度を向上させるか調査する．そして，本論で提案する多方向光学式骨密度計測法でどの程度の予測性能が得られるのかを検証する．最後に，本研究の考察並びに制限について議論し，今後の課題や方針についても述べる．

第 5 章では，研究の総括を述べる．

第 2 章

チューリングパターンを用いた

骨組織モデルの生成

2.1　緒言

骨組織の表面は，緻密で均質に見えるがその内部はスポンジ状である．この表面の緻密な骨を緻密骨もしくは皮質骨と言い，内部のスポンジ状の骨を海綿骨と呼ぶ．海綿骨は，骨梁間隙と網目状の骨梁からなり，長い大型な骨では，中心に向かうにつれ骨梁間隙の大きさと数が増え髄腔を形成し，逆に末端に向かうほど骨梁が増加する．海綿骨組織の骨梁間隙と髄腔は，骨髄で満たされている．この骨梁と骨髄で形成される海綿骨の骨基質は，非常に複雑だが，どこか規則的なパターンをもつ．

複雑な形態をもつ骨梁のパターンは，生物が発生してから自然と形作られていく．様々な生物が，同じような見た目の海綿骨を持つことから，海綿骨の発生は，遺伝のようなあらかじめプログラムされた現象というよりもむしろ，物理的な必然性のもとに行われるのではないだろうか．このように生物の体に存在するパターンをその発生メカニズムから説明する数理モデルの 1 つが，チューリングの反応拡散モデルである．このモデルは，熱帯魚や動物の体表の模様，四肢の発生の研究に頻繁に用いられている．

本章では，チューリングモデルを利用し，骨の物理的な発生メカニズムから，仮想的な海綿骨モデルを生成する．また，生成した海綿骨モデルに骨形態計測を実施し，実際の海綿骨とどの程度一致するのか確認する．

2.2　チューリングパターン

海綿骨を含め，生物のパターンや形態は発生段階から形成されていく．その形成過程を研究する分野を発生学という．１つの受精卵から複雑で多様な形状が形作られていく過程は神秘的で，生物学者だけでなく数学者を含め様々な分野の研究者を魅了してきた．

数理的な発生のパターン形成問題を論じる前に，遺伝子の発生におけるパターン形成への関与を論じる必要がある．ノーベル賞を受賞したショウジョウバエでの形態を決定する転写因子群の発見や遺伝子改変マウスの作成技術に関する研究からも分かるように，遺伝子はパターン形成に不可欠な役割を示している．しかしながら，発生現象に関与する遺伝子群のリストが出揃ってきたものの，それらがどのように相互作用し生物のパターンを形成しているかはわかってない．現在では，遺伝子そのものだけではパターンを生み出せず，遺伝子はパターン形成のための青写真を提供しているに過ぎないと考えられている [80]．従って，発生のパターン形成を記述するモデルには遺伝が含まれていない．

それでは，どのように生物のパターン形成は行われるのだろうか．発生生物学には，モルフォゲン(morphogen)の濃度勾配によりパターン形成が生じるという考え方がある．発生の特定の段階において細胞外に放出されるシグナル因子，つまりモルフォゲンが拡散しつつ広がることで，生産部位を中心とする濃度勾配が生じる．細胞たちが応答するシグナル因子はあらかじめプログラムされており，それに従って，各細胞が化学的な濃度座標における自身の位置を知り，それに応じて分化したり，適切な細胞形態変化を経たり，遊走したりする，という考え方である．つまり，濃度勾配というプレパターンが一度確立してしまえば，形態形成はそれに隷属するプロセスとなる．この濃度勾配の値は位置情報（positional information）と呼ばれている [80].

骨発生では，まず元となる軟骨の周期的な構造の枠組みが形成され，それが骨に徐々に置き換わっていく．チューリングパターンは，このような周期的な構造を生み出す代表的なメカニズムである．このチューリングパターンを形成する数理モデルは，チューリングモデル，もしくはモデルの特徴から反応拡散モデルと呼ばれ，数学者である Alan Turing（1912-1954）が 1952 年から亡くなる 1954 年までの間に行った，固体発生における形態形成に関する研究の１つである．チューリングは，この研究の他にも，電子計算機に関する情報処理の初期的

また原理的な分野において貢献し，第２次世界大戦中は，ドイツのエニグマ暗号機を利用した暗号通信を読解した功績で知られている．

本節では，このチューリングパターンの詳細について記述する．このモデリングの説明は，Miura と Maini (2004)の報告 [79]，並びに三浦岳著の「数理発生生物学」[81] に基づいている．

2.2.1　チューリングモデルの系

話をシンプルにするため，横の長さ 1，たての長さ dy の 1 次元の組織を考える．この 1 次元の組織の中で細胞は２種類の化学物質，活性化因子（activator）と抑制因子（inhibitor）を生産する．そして，それらの化学物質が，細胞による化学物質の生産と分解をコントロールしながら隣り合う細胞に拡散しているという状態を考える．本節で論じる系は，この状態がある時間続いたとき，これら２種類の化学物質がどのように変化していくかを観測する．

空間的な分布変化を考えるため，図 2.1 に示すように，1 次元の組織を dx ずつ分割し，離散化する．ここで，dx と dy は，１と比べて非常に小さく，コンパートメント（組織片）内の分布は無視できると仮定する．計算は次のステップで説明する．まず，活性化因子と抑制因子は，この組織片の中で相互作用する（反応）．次に，ある組織片と両隣の組織片との間でやりとりを行う（拡散）．そして，それらを足し合わせたものを時間変化として計算する．ここで，図 2.1 に示すように，時刻 m dt における n 番目の組織片の中の活性化因子濃度を p(n, m)，抑制因子の濃度を q(n, m)と記述する．

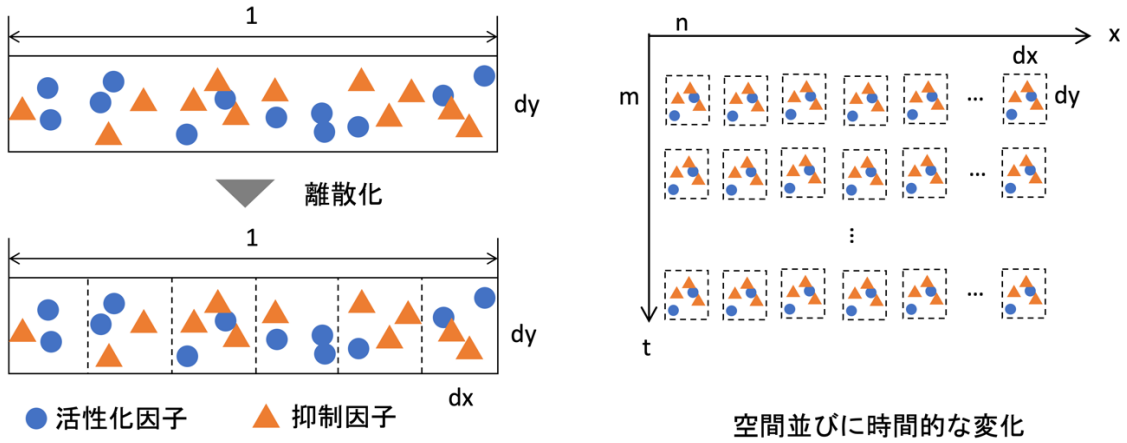


図 2.1　1 次元的なチューリングモデルの系ならびにその離散化と時間変化

2.2.2　反応項

ある時刻 mdt における活性化因子と抑制因子の濃度分布がわかっていると仮定する．周期的なチューリングパターンを生成する反応系では，次の 2 点を満たす必要がある． 1，活性化因子は，活性化因子自身の生産を促すと同時に，抑制因子の生産を促進する． 2，抑制因子は活性化因子の生産を抑制する．従って，反応系は，活性化因子と抑制因子の濃度変化は，f(p, q)ならびに g(p, q)として次式のように書ける．

$$f(p,q) = c_0 p - c_1 q, \qquad g(p,q) = c_2 p - c_3 q \tag{2.1}$$

ここで c は，活性化因子と抑制因子を制御する係数である．このような関数を定義し，

$$f(p(n,m),q(n,m))\mathrm{d}t, \qquad g(p(n,m),q(n,m))dt \tag{2.2}$$

とすれば，微小組織空間 dxdy 中に存在する，時刻 mdt から(m+1)dt の活性化・抑制因子濃度変化が算出できる．この項は，反応拡散の中で化学物質の反応を表していることで，反応項と呼ばれている．

2.2.3　拡散項

時刻 mdt から(m+1)dt の間に起こる隣接する組織間の相互作用を考える．活性化・抑制因子が受動的に拡散して隣接する組織に影響を及ぼすとする．n 番目の組織の活性化因子濃度を p(n, m)とすると，右隣の組織は p(n+1,m)である．単純な拡散の場合，物質の移動量は Fick の法則より，$\frac{p(n+1,m)-p(n,m)}{dx}$と dy に比例するため，dt 間に右隣の移動する活性化因子の量は,

$$\frac{d_p\{p(n+1,m)-p(n,m)\}}{dx}dydt \tag{2.3}$$

と書ける．ここで d_p は活性化因子の拡散係数である．従って，この拡散により引き起こされる n 番目の組織の濃度変化は，組織面積（*dxdy*）で除すことで,

$$\frac{d_p\{p(n+1,m)-p(n,m)\}}{dx^2}dt \tag{2.4}$$

となる．ここまでは，n 番目の組織片とその右側に隣接する組織片とのやりとりを述べたが,同様のやりとりは，左側に隣接する組織片とも行われる．そのため，n 番目の組織中にある活性化因が，時刻 mdt から(m+1)dt の間に周囲への拡散によって変化する量は,

$$\frac{d_p\{u(n+1,m)+u(n-1,m)-2u(n,m)\}}{dx^2}dt \tag{2.5}$$

となる．同様のやりとりは，抑制因子でも行われるため，抑制因子の濃度変化は,

$$\frac{d_q\{v(n+1,m)+v(n-1,m)-2v(n,m)\}}{dx^2}dt \tag{2.6}$$

となる．ここで，d_q は抑制因子の拡散係数を意味する．この項は，拡散を表すため，反応拡散系では，拡散項と呼ばれている．

2.2.4　支配方程式

式 2.2 ならびに式 2.5 より，ある時刻 mdt から(m+1)dt の間に活性化因子が変化する量 p(n,m+1)-p(n,m)は,

$$p(n,m+1)-p(n,m)=\left\{f\big(p(n,m),q(n,m)\big)+\frac{d_p[p(n+1,m)+p(n-1,m)-2p(n,m)]}{dx^2}\right\}dt \tag{2.7}$$

また，式 2.2 ならびに式 2.6 より，抑制因子が変化する量 q(n,m+1)-q(n,m)は,

$$q(n,m+1)-q(n,m)=\left\{g\big(p(n,m),q(n,m)\big)+\frac{d_q[q(n+1,m)+q(n-1,m)-2q(n,m)]}{dx^2}\right\}dt \tag{2.8}$$

となる．従って，(m+1)dt における活性化因子と抑制因子の濃度は，次式のようになる．

$$p(n,m+1)=p(n,m)+\left\{f\big(p(n,m),q(n,m)\big)+\frac{d_p[p(n+1,m)+p(n-1,m)-2p(n,m)]}{dx^2}\right\}dt \tag{2.9}$$

$$q(n,m+1)=q(n,m)+\left\{g\big(p(n,m),q(n,m)\big)+\frac{d_q[q(n+1,m)+q(n-1,m)-2q(n,m)]}{dx^2}\right\}dt \tag{2.10}$$

つまり，活性化因子と抑制因子の初期値がわかっていれば，この式を繰り返し適用して計算することで，任意の時刻の濃度分布を取得することができる．

次に，境界条件を定義する．ここでは，領域が十分広いと仮定し，周期的境界条件を用いる．周期的境界条件とは，初期配置の右端は左端と，上端は下端と繋がっていると考えることである．

dt と dx を無限にゼロに近づけると，上述の離散な支配方程式は，連続の支配方程式として記述できる．連続関数として，時刻 t における位置 x の活性化因子と抑制因子の濃度をそれぞれ u(x, t)ならびに v(x, t)とすると，式 2.9 と式 2.10 は次式のように記述できる．

$$\frac{u(x,t+dt)-u(x,t)}{dt}=f\big(u(x,t),v(x,t)\big)+d_u\frac{\frac{u(x+dx,t)-u(x,t)}{dx}-\frac{u(x,t)-u(x-dx,t)}{dx}}{dx} \tag{2.11}$$

$$\frac{v(x,t+dt)-v(x,t)}{dt}=g\big(u(x,t),v(x,t)\big)+d_v\frac{\frac{v(x+dx,t)-v(x,t)}{dx}-\frac{v(x,t)-v(x-dx,t)}{dx}}{dx} \tag{2.12}$$

ここで，du と dv は，それぞれ活性化因子と抑制因子の拡散係数である．また，左辺は時間の 1 次微分，右辺の拡散項は空間の 2 次微分となるため，次式のように記述できる．

$$\frac{\partial u(x,t)}{\partial t}=f\big(u(x,t),v(x,t)\big)+d_u\frac{\partial^2 u(x,t)}{\partial x^2} \tag{2.13}$$

$$\frac{\partial v(x,t)}{\partial t}=g\big(u(x,t),v(x,t)\big)+d_v\frac{\partial^2 v(x,t)}{\partial x^2} \tag{2.14}$$

従って，反応拡散モデルの連続支配方程式は，

$$\begin{cases}\dfrac{\partial u}{\partial t}=f(u,v)+du\Delta u\\[2ex]\dfrac{\partial v}{\partial t}=g(u,v)+dv\Delta v\end{cases} \tag{2.15}$$

となる．これは，Activator-Inhibitor system とも呼ばれており，非線形項 f，g と，拡散項で表される．ここで，拡散項のΔ は，ラプラス演算子であり，反応項の f と g は式 2.1 に対応す

る．これまでチューリングパターンの系を 1 次元として扱ってきたが，この連続支配方程式は，2 次元や 3 次元でも同様である．系の次元を上げるには，拡散する方向を 2 次元や 3 次元方向に拡張すれば良い．

2.2.5 解法

時刻に対する変化を計算する際，時間の進め方には，陽解法と陰解法と呼ばれる 2 つの手法がある．陽解法は，現在の時刻 t における値をもとにして，時刻 t + dt の値を代数的に求め，それを繰り返していく．陽解法は，比較的簡単な演算であることが多く，1 ステップの計算負荷も少ない．ただし，陽解法では dt を大きく取ってしまうと，計算結果が発散してしまうため，非常に多くの時間ステップを計算する必要がある．そのため，2 次元や 3 次元空間等の高負荷な計算には不向きな手法である．一方，陰解法は，少し未来の時間における値を仮定し，その過程が正しいかどうかをその場の支配方程式を用いて調べ，誤差がなくなるように仮定値を収束させていく．この場合，巨大な行列式を解く計算が必要になるため，1 つの時間ステップあたりの計算量は多くなるが，ステップ数は少なくできる．ただし，行列演算は，グラフィックス・プロセシング・ユニット (GPU)の得意分野であるため，大幅な計算の高速化が望める．従って，本研究では支配方程式の解法に陰解法を用いた．

3 次元空間に拡張された式 2.15 の支配方程式は，陰解法で次のように書ける．

$$\begin{cases} u(x,y,z,t+dt) = u(x,y,z,t) + \dfrac{dt\,du}{dx^2}\begin{bmatrix} u(x+dx,y,z,t+dt) + u(x-dx,y,z,t+dt) \\ +u(x,y+dy,z,t+dt) + u(x,y-dy,z,t+dt) \\ +u(x,y,z+dz,t+dt) + u(x,y,z-dz,t+dt) \\ -6u(x,y,z,t+dt) \end{bmatrix} \\ v(x,y,z,t+dt) = v(x,y,z,t) + \dfrac{dt\,dv}{dx^2}\begin{bmatrix} v(x+dx,y,z,t+dt) + v(x-dx,y,z,t+dt) \\ +v(x,y+dy,z,t+dt) + v(x,y-dy,z,t+dt) \\ +v(x,y,z+dz,t+dt) + v(x,y,z-dz,t+dt) \\ -6v(x,y,z,t+dt) \end{bmatrix} \end{cases} \quad (2.16)$$

式 2.16 は，u(x, y, z, t)，v(x, y, z, t)の分布を示す行列 U(t), V(t)と，拡散を示すカーネル Ku, Kv の畳み込み積から，

$$\begin{cases} Ku * U(t+dt) = U(t) \\ Kv * V(t+dt) = V(t) \end{cases} \quad (2.17)$$

となる．ここで，*は畳み込み積を意味し，Ku，Kv は，算出したいモデルのグリッド空間のサイズと同等の大きさの 3 次元配列であり，

$$Ku_{0,0,0} = 1 + 6u(x, y, z, t + dt),$$

$$Ku_{1,0,0} = Ku_{-1,0,0} = Ku_{0,1,0} = Ku_{0,-1,0} = Ku_{0,0,1} = Ku_{0,0,-1} = -dt\frac{du}{dx^2},$$

$$Kv_{0,0,0} = 1 + 6v(x, y, z, t + dt),$$

$$Kv_{1,0,0} = Kv_{-1,0,0} = Kv_{0,1,0} = Kv_{0,-1,0} = Kv_{0,0,1} = Kv_{0,0,-1} = -dt\frac{dv}{dx^2} \tag{2.18}$$

である．畳み込みの定理より式 2.17 は，

$$\begin{cases} \mathcal{F}[Ku]\,\mathcal{F}[U(t+dt)] = \mathcal{F}[U(t)] \\ \mathcal{F}[Kv]\,\mathcal{F}[V(t+dt)] = \mathcal{F}[V(t)] \end{cases} \tag{2.19}$$

と書ける．したがって，t+dt 後の U，V は，

$$\begin{cases} U(t+dt) = i\mathcal{F}\left[\dfrac{\mathcal{F}[U(t)]}{\mathcal{F}[Ku]}\right] \\ V(t+dt) = i\mathcal{F}\left[\dfrac{\mathcal{F}[V(t)]}{\mathcal{F}[Kv]}\right] \end{cases} \tag{2.20}$$

である．ここで，$\mathcal{F}$はフーリエ変換，$i\mathcal{F}$は逆フーリエ変換を示す．計算の流れとしては，まず反応項を計算し，次にそのフーリエ変換，そして拡散を示すカーネル K をフーリエ変換したもので除し，最後にフーリエ逆変換をする．

本項では，3 次元の陰解放について述べたが，2 次元にこの解法を定期要する場合は，これらを単に 2 次元に変更すれば良い．

2.2.6　チューリングモデルの計算例

これまでは，チューリングモデルの支配方程式の導出，その解法について述べてきた．しかしながら，それだけでは実際のモデルの挙動や，どのような解が導出されるかイメージしにくい．そこで本項では，これまでに述べたチューリングモデルの具体的な計算例を示す．

まず，式 2.15 の連続支配方程式において，反応項を次のように定義する．

$$f(u, v) = 0.6u - v - u^3, \qquad g(u, v) = 1.5u - 2v\,. \tag{2.21}$$

この式における項は，

- 0.6u：活性化因子は自らの生産を促進する．
- -v：抑制因子は活性化因子の生産を抑制する．
- $-u^3$：活性化因子はある程度以上の濃度になると生産が抑えられる．
- 1.5u：活性化因子は抑制因子の生産を促進する．
- -2v：抑制因子は何もしていないと減衰する．

をそれぞれ意味する．次に，活性化因子と抑制因子の拡散係数 d_u，d_v は，それぞれ 0.0002 ならびに 0.01 とした．また，式 2.15 を解くためには，活性化因子と抑制因子の初期濃度分布が必要である．初期濃度分布は，活性化因子，抑制因子ともに-0.5 から 0.5 の範囲の一様な乱数とした．最後に，計算するグリッド空間のサイズと 1 グリッドの大きさ，計算試行回数を決定する．グリッド空間のサイズは，対象とする大きさの組織を意味し，ここでは 256 × 256 × 256 の 3 次元配列とした．1 グリッドの大きさは，図 2.1 における dx 並びに dy の大きさ，つまり解像度を意味し，$\frac{1}{64}$ とした．計算試行回数は，2.2.5 項で示した計算手順を繰り返す回数であり，ここでは 100 回とした．

図 2.2 に計算試行回数ごとに活性化因子が形成するパターンの変化を示す．設定された初期条件からもわかるように，Gen 0 では，ホワイトノイズが得られる．ここで Gen は，generation を意味し，計算の試行回数を示す．そして，試行回数が増えるに従い，周期的なパターンが徐々に形成され，図 2.2 では Gen 20 にはおおよその形状が既に形成され，Gen 95 では，よりはっきりしたパターンが形成されていることがわかる．この周期的なパターンをチューリングパターンと呼ぶ．過去の研究より，チューリングパターンが現れる必要十分条件は，次の 4 条件であることがわかっている．

1. $f_u + g_v < 0$
2. $f_u g_v - f_v g_u > 0$
3. $d_v f_u + d_u g_v > 0$
4. $(d_v f_u + d_u g_v)^2 - 4 d_u d_v (f_u g_v - f_v g_u) > 0$

ただし，$d_u \ll d_v$ である場合，条件 3 と 4 は自動的に成り立つ．そして，式 2.21 は，条件 1 と 2 を満たしている．従って，本条件では解が収束し，チューリングパターンが現れることがわかる．解の収束に関して，より詳しい解析を実施には，線形安定性解析 [73,80,81] を導入す

ることを推奨する．本論での説明は省略するが，線形安定性解析を実施することで，チューリングパターンが現れるか，幾つの周期的パターンが現れるか等があらかじめ把握できる．

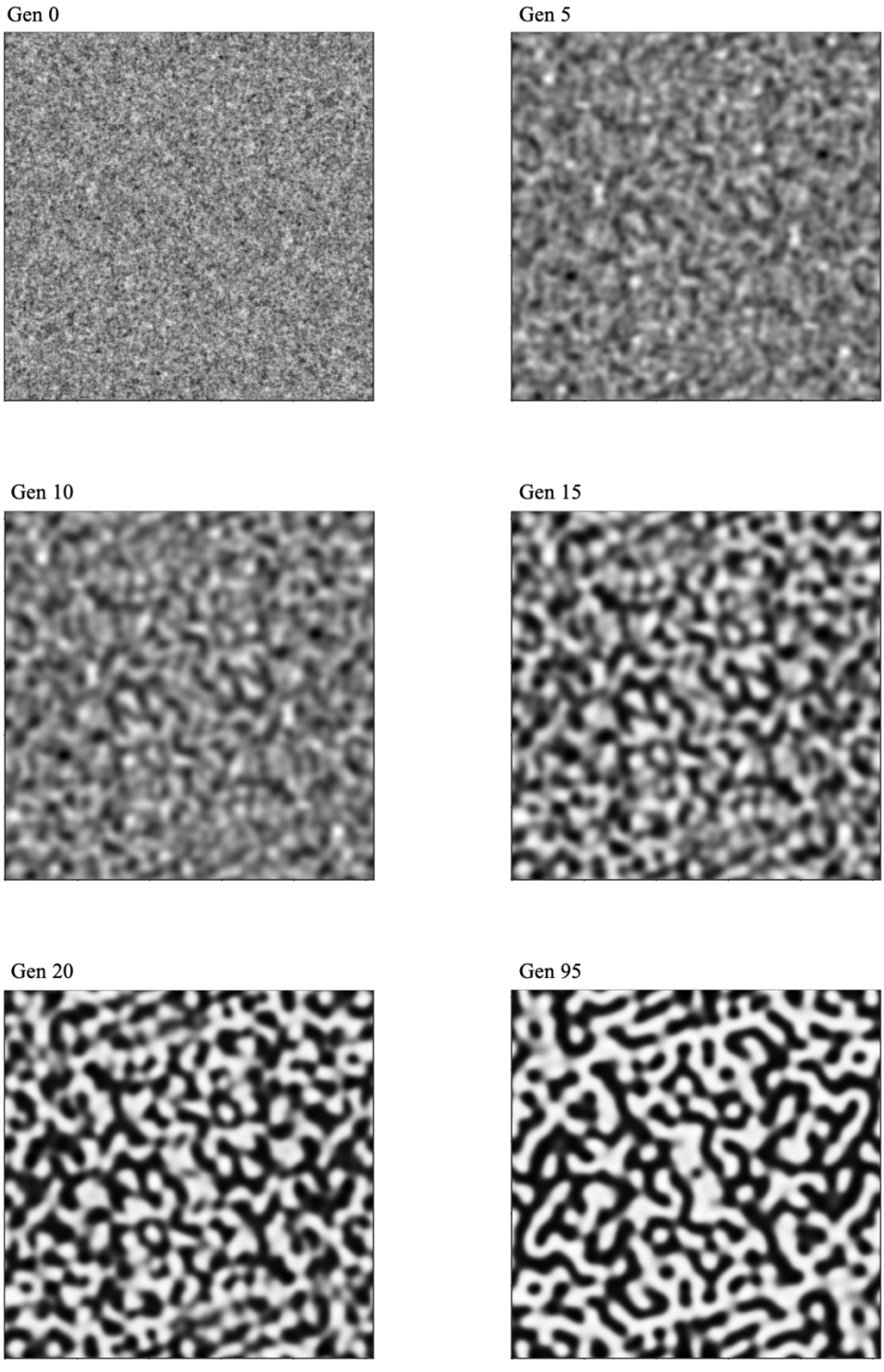


図 2.2　活性化因子 u の計算試行回数によるパターン形成

図 2.3 に 100 回計算を繰り返した際に形成される，活性化因子 u と抑制因子 v のパターンとその濃度をヒートマップで表見したものを示す．なお，ここで表示されたものは，z 軸で 128 枚目をスライスした画像である．活性化因子と抑制因子は，同じようなパターンを示した．また，活性化因子は，-0.6 から 0.6 の値を示し，抑制因子は，活性化因子よりも狭い範囲である-0.15 から 0.15 の範囲の値を示した．活性化因子と抑制因子の濃度パターンを比較したものを図 2.4 に示す．ここで，図 2.4 は図 2.3 の y 軸 128 番目のスライスを表示したものである．このよう，活性化因子と抑制因子のパターンの山と谷は定性的に一致し，周期構造を形成する．このような周期構造が形成される理由は，標準的な説明を引用すると次のようになる [81,82]．まず，活性化因子と抑制因子の初期分布が完全に一致することは自然な状態では考えにくく，初期状態ではごくわずかなピークが無数に存在する．すると，血清課因子は自分自身の生産を促進するという性質によって，活性化因子の小さな揺らぎが増強され徐々に山が形成される．次に，活性化因子が抑制因子の生産を促進するという性質によって，同じ部分に抑制因子の山が形成される．ただし，抑制因子の方が，拡散速度が速いため（$d_v > d_u$），活性化因子の山よりも抑制因子の山の方がなだらかになる．それらの山の中心から少し外れた部分では，活性化因子よりも抑制因子の働きが相対的に強くなり，抑制因子は活性化因子の働きを抑えるという作用から，既に形成された山のすぐ横では，新たな活性化因子の山が形成されにくくなる．この働きによって，ほぼ一定幅のパターンのみが形成される，という説明である．より正確な説明は，本項で既に説明したチューリングパターンが現れる 4 つの条件である．図 2.2 から 2.4 の計算例は，これらの説明を視覚的に表現していることがわかる．

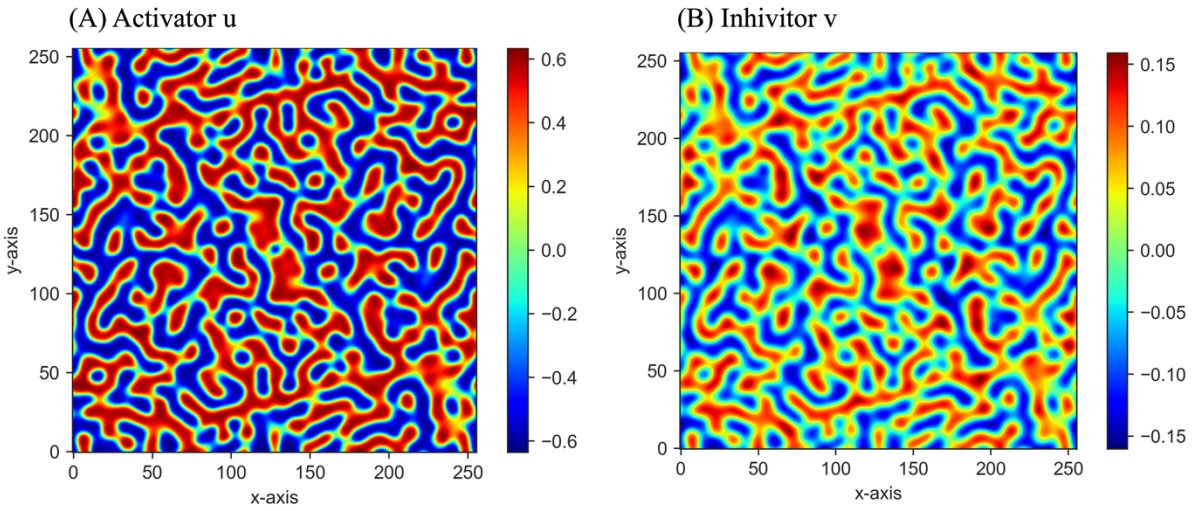


図 2.3　計算終了後の（A）活性化因子 u と（B）抑制因子 v の濃度とパターン

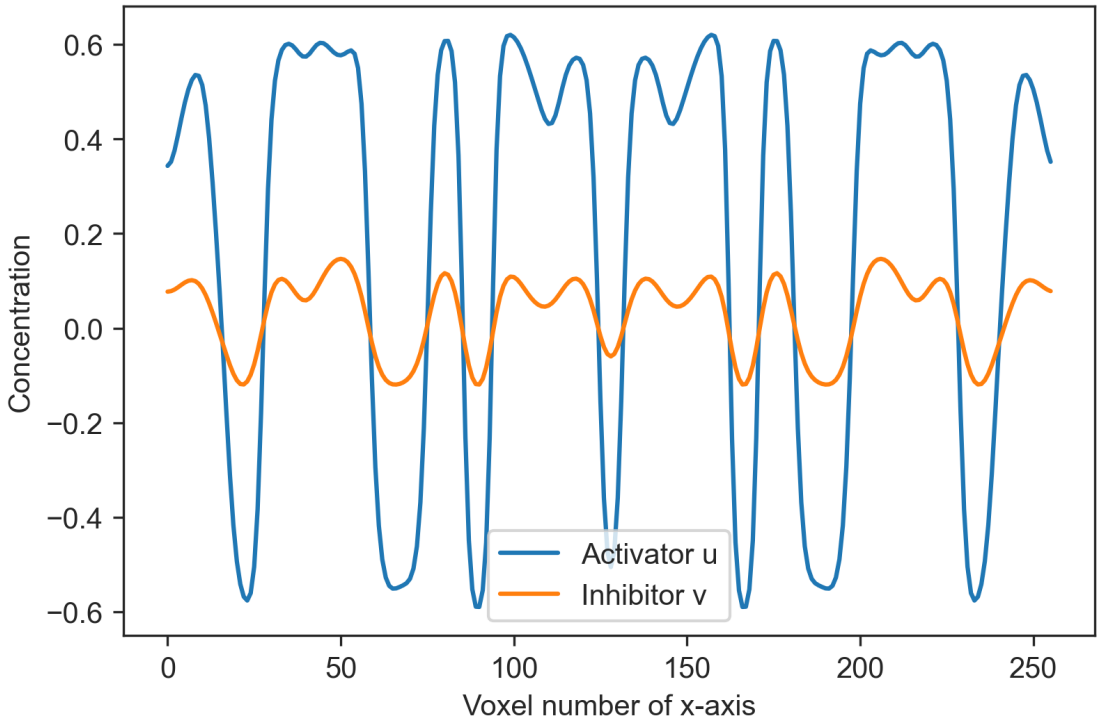


図 2.4　計算終了後の活性化因子と抑制因子の濃度パターンの比較

2.3　骨組織モデルの生成

前節（2.2 節）では，チューリングの反応拡散モデルを用いて，発生学の考えに基づき骨梁の周期的パターンを生成する方法を説明した．上記のモデルは，実際に四肢の発生や熱帯魚の模様の発生等の周期的なパターンを伴う発生を説明する上で重要なモデルである．骨形成においても，前節で説明したように，まず基本の枠組みとなる周期的な構造の軟骨が形成され，それが骨に徐々に置き換わっていく．本節では，チューリングモデルで生成した骨構造の枠組み，つまり活性化因子 u を用い，骨組織を定義する方法を説明する．

骨組織は，図 2.5 に示す手順で定義した．まず，前節の方法を用いて，活性化因子 u の濃度パターン，つまり骨梁パターンを生成する．次に，骨梁と骨梁間隙である骨梁間隙部分を定義する．そして，骨のスケールを定義する．最後に，海綿骨部を覆う皮質骨を定義する．また，本節では，骨粗鬆症における重要指標である面積骨密度（aBMD）を骨組織モデルから算出する方法についても説明する．

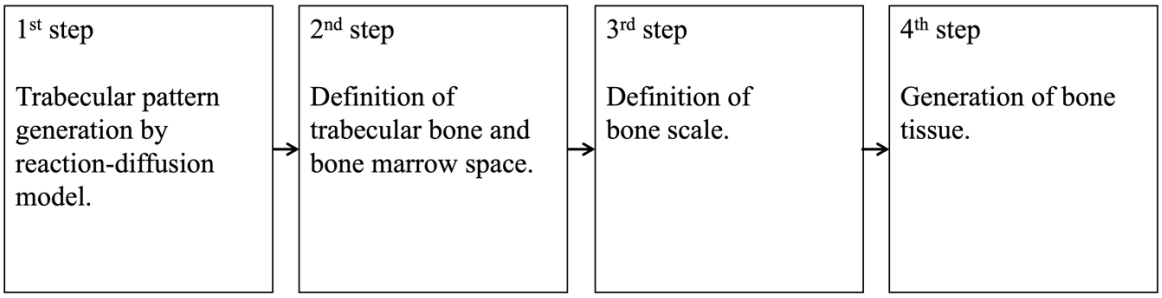


図 2.5　骨組織モデルの作成手順

2.3.1　骨梁パターンの生成

式 2.15 の連続支配方程式における反応項は，式 2.21 を用いた．活性化因子と抑制因子の拡散係数 d_u，d_v は，それぞれ 0.0002 ならびに 0.01 とした．また，活性化因子 u と抑制因子 v の初期濃度分布は，活性化因子，抑制因子ともに-0.5 から 0.5 の範囲の一様な乱数とした．

グリッド空間のサイズは，ここでは 720 × 720 × 720 の 3 次元配列とした．1 グリッドの大きさは，$\frac{1}{64}$ とした．計算試行回数は 100 回とした．以上の条件で活性化因子濃度 u を算出した．

2.3.2　骨梁と骨梁間隙の定義

骨梁と骨梁間隙の境界を閾値 u_{th} で定義した．まず，10 種類の初期条件で u をランダムに生成して決定した 10 個の 3 次元チューリングパターンに u_{th} を適用した．従って，$u \geqq u_{th}$ を海綿骨，$u < u_{th}$ を骨梁間隙と定義し，同様の BV/TV を持つ海綿骨を 10 個作成した．これを図 2.6 のように 0.5 から 0.6 の活性化因子 u 濃度範囲にて 0.01 間隔で u_{th} を繰り返し決定した．このように，異なる閾値 u_{th} を定義し，チューリングパターンを骨梁と骨梁間隙に 2 値化することで，BV/TV（骨量, bone volume fraction）の異なる海綿骨を定義することが可能である（図 2.6）．ここで，BV/TV (bone volume / tissue volume) は，骨体積を全海綿骨組織体積で除したものである．また，u_{th} と BV/TV の間には次式のような関係が得られる．

$$u_{th} = -7.662(BV/TV)^3 + 1.645(BV/TV)^2 - 0.452\,BV/TV + 0.604 \qquad (2.23)$$

この式は，図 2.5 のプロットを最小二乗法で 3 次多項式関数にフィッティングしたものである．式 2.23 を用いることで，BV/TV で 2.5 から 22 %の範囲の海綿骨を定義する u_{th} を BV/TV 値から決定することができる．

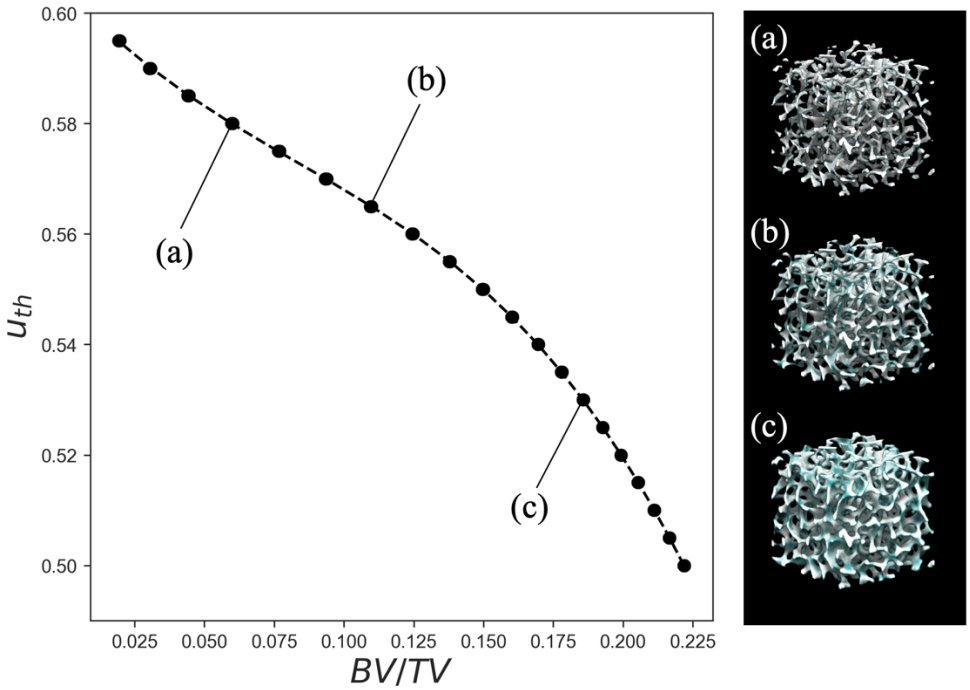


図 2.6　BV/TV と u_{th} の関係（u_{th} は活性化因子 u の閾値）

2.3.3　スケールの定義

活性化因子濃度 u のボクセルのサイズ l_v（体積要素のサイズ）を決定し，骨組織のスケールを定義する．l_v は骨梁数 (Tb.N, mm^{-1})から定義した．Tb.N は，楕円体近似における中軸の骨梁平均間隔の逆数で表現される骨形態パラメーターである [83]．つまり，１つの軸（例えば x 軸）に対し Tb.N は次式のように表される．

$$Tb.N = \frac{Peak\ number}{l_v \times Voxel\ number} \tag{3.17}$$

反応拡散モデルでは，全ての軸方向に等方的に拡散するため，l_v は，

$$l_v = \frac{Peak\ number}{Tb.N \times Voxel\ number} \tag{3.18}$$

となる．ここで，Voxel number は活性化因子濃度 u を構成するボクセル数である．Peak number は図 2.4 に示すような活性化因子 u の正の数におけるピークの数である．

Peak number 並びに Tb.N は，BV/TV によって異なる値を示す．従って，本論では橈骨最遠位端（ultra-distal（UD）radius）における骨粗鬆症 3 状態（健常，骨減少症，骨粗鬆症）の

BV/TV の平均値（0.134, 0.103, 0.085）[83] から式 3.16 にて u_{th} を導出生成した．また Tb.N は，骨粗鬆症の 3 状態から，健常を 1.71 mm^{-1}，骨減少症を 1.44 mm^{-1}，骨粗鬆症を 1.32 mm^{-1} とした [83]．式 3.18 に Peak number 並びに Tb.N を代入すると，l_v は 24.5 ± 0.4 μm となった．従って，平均値である 24.5 μm を l_v とした．この l_v は，μCT で取得できる 3 次元画像の解像度と同程度である．

2.3.4 骨組織モデルの生成

骨組織モデルは，海綿骨組織モデルに皮質骨領域を定義することで生成した．まず，モデルの方向は，骨軸方向を y 軸とし，骨周囲方向をそれぞれ x，z 軸とし，右手形で定義した．次に骨組織全体の大きさを定義した．骨軸方向に十分な大きさを確保するため，生成した海綿骨を y 軸方向にコピーし接合した．2.2.3 項で既に説明したように，海綿骨の形態を定義するチューリングパターンの支配方程式の境界条件は，周期的境界条件である．従って，y 軸方向の接合面は完全に一致し，連続したパターンとなる．UD radius [83] の断面積から，一辺 17.15 mm の正方形を x-z 平面における外骨表面（OBS）と定義した（図 3.4）．したがって，骨組織の大きさは，x-z 軸で 17.15 mm，y 軸で 35.28 mm となった．次に，OBS から海綿骨の中心に向かって厚さ C.Th の皮質骨を定義し，骨組織を生成した．

2.3.5 面積骨密度の定義

骨組織の面積骨密度 aBMD [g/cm^2]は，BV/TV，X 軸方向の OBS の大きさ l_{obs} [cm]，骨基質量 mBMD [g/cm^3]，皮質骨厚さ C.Th [cm]を用いて，次式のように定義した．

$$aBMD = mBMD\left[1-(1-BV/TV)\left(1-\frac{2C.Th}{l_{obs}}\right)^2\right] \tag{3.19}$$

例えば，UD radius の C.Th は，Boutroy (2005) [83] らの HR-pQCT の計測によると，健常者で 804 ± 149 μm，骨減少症で 571 ± 173 μm，骨粗鬆症で 487 ± 138 μm である．

2.4　骨組織モデルの評価

生成された海綿骨組織モデルが，定性的また定量的な観点から，どの程度実際の海綿骨に近いのか評価した．定量的な指標としては，海綿骨モデルに骨形態計測を実施し，過去の研究で調査されたヒト海綿骨の 3 次元形態計測値と比較した．

図 2.7 に示すようにチューリングモデルで生成された海綿骨モデルと，ウシ大腿骨近位部から採取した海綿骨ブロックの μCT 画像を比較した．海綿骨モデル並びにウシ海綿骨は，同様に複雑な 3 次元構造を示した．海綿骨モデルは，実際の骨で観察される海綿状骨梁パターンを再現しているように見えた．

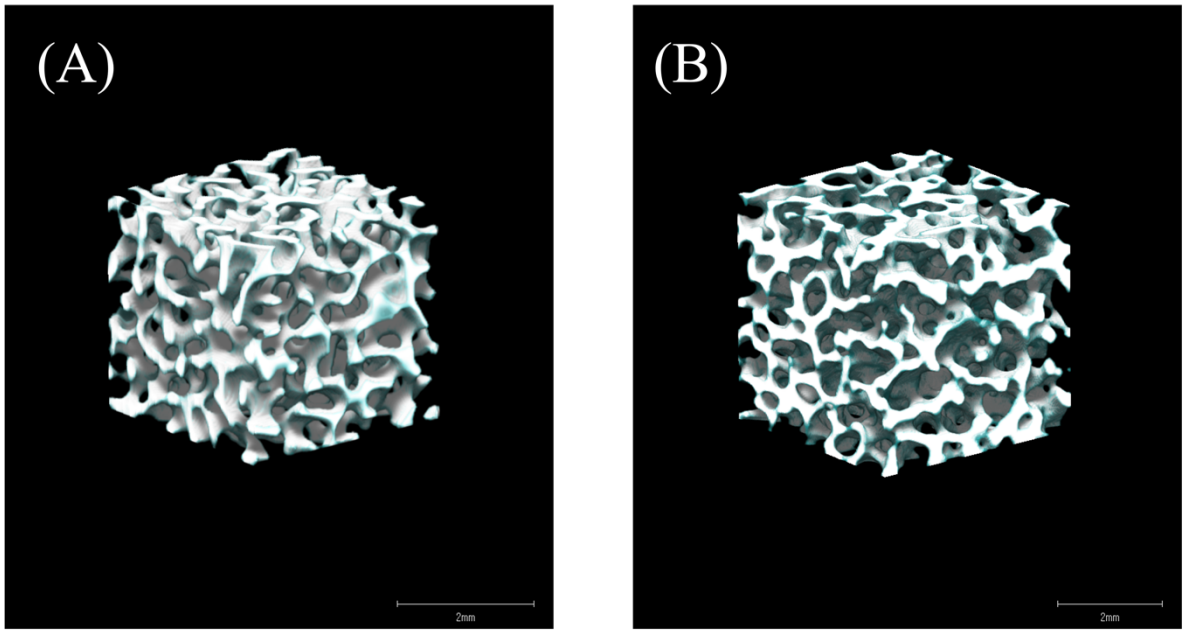


図 2.7　(A) 反応拡散モデルで生成した海綿骨と，(B) 大腿骨頚部から採取したウシ海綿骨の μCT 画像との外観比較

生成した海綿骨とヒト UD radius の骨組織計測値を表 2.1 に示す．生成された海綿骨は，骨梁幅(Tb.Th)，フラクタル次元，構造モデル指数（SMI）に関して，UD radius と同程度であった．フラクタル次元は，骨梁を 3 次元図形としてフラクタル次元，つまり骨梁表面の複

雑さを数値化したものであり[84,85]，SIM は骨梁の板，棒状の指標であり，骨梁構造を理想的な板状の場合 0，棒状の場合 3，球状の場合 4 として混合状態を中間地で表現したものである[86,87]．つまり，構造的な指標において，生成された海綿骨モデルと，実際の海綿骨は一致する．また，BV/TV は海綿骨組織の骨梁と骨梁間隙を定義する際に使用し，骨梁数 Tb.N も海綿骨のスケール定義に使用したパラメータであるが，これらの値も実際の骨梁に近い値を示した．

一方，骨梁間隙(Tb.Sp)およびその分散（Tb.Sp SD）は，UD radius のそれよりも小さい．これは，UD radius の異方性（Degree of anisotropy）に起因すると考えられる．Tb.Sp は骨梁間隙中で骨梁表面までの距離 r を直接計測し，極大値をとる点 2r の平均を計測したものであり，異方性は回転楕円体近似長軸長を単軸長で除したものである．海綿骨モデルが異方性を有しない原因は，反応拡散モデルにおいて活性化因子と抑制因子の等方的な拡散が想定されているためである．モデルと実際の海綿骨で最も異なる点が，連結密度 connD である．連結密度はトポロジーの考えを導入し，骨の形を評価するための指標の 1 つである．トポロジーでは，連続的に変形させて重なり合う形は全て同じ形と考える．そのため，トポロジーの考えにおけるものの形は，連結成分，わっか，空洞の 3 つでしかあり得ない．連結密度は，連結成分の個数 b0 と空洞の個数 b2 の和をわっかの数 b1 で引いたオイラー数 E,

$$E = b_0 - b_1 + b_2 \tag{3.20}$$

を骨組織体積（TV）で除したものである[88]．つまり，どれだけ骨梁が繋がっているかの指標である．生成された海綿骨の連結密度は実際のものと比較し 1 桁多くなる．従って，生成された海綿骨モデルは実際ものよりも異方性が少なく，均質な大きさの骨梁間隙を有し，より連続した構造であることがわかる．

表 2.1　反応拡散モデルで生成した海綿骨とヒト超遠位橈骨の骨組織計測値の比較

	Generated trabecular bone			**Trabecular bone of the ultra-distal radius**			
	Normal	Osteopenia	Osteoprosis	Normal	Osteopenia	Osteoprosis	Reference
BV/TV [%]	13.3	10.6	8.6	13.4 ± 2.8	10.3 ± 3.0	8.5 ± 2.2	
Tb.N [mm^{-1}]	1.74	1.51	1.32	1.71 ± 0.22	1.44 ± 0.29	1.32 ± 0.21	
Tb.Th [μm]	56	52	49	78 ± 11	71 ± 11	63 ± 11	Boutroy 2005[83]
Tb.Sp [μm]	204	211	219	517 ± 88	656 ± 187	714 ± 140	
Tb.Sp SD [μm]	40.6	42.5	44.3	212 ± 58	342 ± 201	340 ± 89	
Fractal dimension	2.13	2.03	1.93	2.33 ± 0.04	-	2.24 ± 0.03	Bayarri 2010[85]
SMI	2.04	2,27	2.44	2.26 ± 0.38			
The degree of anisotropy (a/c)	1.01	1.02	1.03	1.45 ± 0.09			Zhou 2016[86]
ConnD [mm^{-3}]	62.1	41.0	20.7	2.22 ± 0.78			

本論の海綿骨モデルは，リモデリングによる形態変化を考慮しておらず，これは明らかな制限である．第 1 章に述べたように，骨はリモデリングにより形態の経年変化を示し，この病的な変化が骨粗鬆症である．チューリングモデルは，骨の発生段階をよく表すが，それ以降の経年変化は表現できない．この制約が，異方性におけるモデルと，文献値の不一致に関係しているものと考えられる．海綿骨の経年変化を表現するためには，チューリングモデルで生成した骨梁パターンに対し，リモデリングによる形態変化を表現する数理モデルの導入する必要がある．しかしながら，このモデルが潜在的に重要なのは，海綿骨とよく似た形態をもつモデルを作成可能であり，またそれらのパラメータがある程度制御可能な点にある．本モデルを使用することで，海綿骨と光伝搬の相互作用について，理解が深まるかもしれない．

本研究では，BMD の変化を BV/TV で表現でき，骨梁のパターンが実際の骨に似た形態を持つことから，本モデルを海綿骨として選択した．なお，骨形態パラメータは骨形態計測ソフトウェア(TRI/3D-BON-FCS, Ratoc system engineering co. ltd, Japan)を用いて算出した．

2.5 結言

本章では，生物のパターン発生を説明するチューリングモデルを用いて骨梁パターンを生成し，骨組織モデルを生成する方法について述べた．また，生成された骨組織モデルを定量的，定性的な視点で評価した．骨組織モデルの海綿骨形状の見た目は，実際の海綿骨と我々の目からは同じものに見えた．骨形態計測値における定量的な比較では，BV/TV，骨梁数，フラクタル次元，構造モデル指数に関して，骨組織モデルと橈骨最遠位端の海綿骨は一致していた．一方，骨組織モデルの海綿骨形態は実際の骨よりも異方性に乏しく，連結密度が高かった．我々は，BMD 値が調節可能で，海綿形状のパターンが実際の骨の形状に似ていることから，本骨組織モデルを採用することとした．

第 3 章

VMC: Voxel-based Monte Carlo simulation

3.1 緒言

光は，光子と呼ばれる粒子である．均一な屈折率を持つ物質中に光子が照射された場合，光は直進し，形態変化に伴う屈折率の変化がある場合，光は散乱する．この光を強く散乱させる媒質を特に強散乱媒質とよび，生体は強散乱媒質の代表的なものである．強散乱媒質中で光がどのような経路をたどり，媒質外に飛び出すかを計算するのがモンテカルロシミュレーションである．

モンテカルロシミュレーションは，中性子の伝搬挙動を探るために Ulam が考案し Neumann により命名された手法．そして，Wilson と Adam (1983)[89] によってレーザーと生体組織の相互作用のシミュレーションに導入された．多層組織へのモンテカルロシミュレーションは，Wang と Jacques (1995)[77] によって開発された．Wang らは C 言語で記述された MCML (Monte Carlo modeling of light transport in multi-layered tissues)と呼ばれるモンテカルロシミュレーションを無料で配布した．そのため，MCML は現在でも簡単に使用することができる．しかしながら，MCML は，均質な多層構造しか扱えず，本論の第 2 章で説明したような 3 次元的な構造は計算できない．従って，3 次元構造を扱えるモンテカルロシミュレーションを開発する必要がある．

本章では，3 次元的な構造を持つ骨組織内の光輸送計算するため，ボクセルベースのモンテカルロ法(VMC)を構築した．また，VMC の妥当性を検証するため，MCML やそのほか過去の文献値と計算例を比較した．モンテカルロ法は，乱数を用いた膨大な数の繰り返し計算を実行する必要があるため，ある程度時間がかかる．VMC では，計算速度向上のため，グラフィックス・プロセシング・ユニット (GPU)を用いた計算を実施する．よって，GPU での計算アルゴリズム，ならびに GPU を使用することで，どの程度計算が高速化できるかについても検証する．本章の最後には，第 2 章で説明した骨組織モデルを軟組織層で覆っ

た合成生体モデルを使用し，計算例を示した．この計算例では，骨密度値や軟組織の光学特性と厚さが変化した場合，計算結果にどの程度変化するかを定性的に検証した．

3.2 Voxel-based Monte Carlo (VMC) method

本論で紹介するモンテカルロ法 VMC は，Monte Carlo modeling of light transport in multi-layered tissues (MCML)[77,90] を，非線形 3 次元構造をもつ組織に適用にするために拡張したものである．そして，組織に垂直に入射した無限に狭い光子ビームの輸送を扱う．無限に狭い光子ビームに対する応答は，インパルス応答と呼ばれる．3 次元構造の各要素は有限の大きさを持ち，光学的な特性は，屈折率 n，吸収係数 μ_a，散乱係数 μ_s，異方性係数 g で表される．3 次元構造要素は，ボクセル（voxel）と呼ばれ，6 面体要素を意味する．実際の組織間の境界は連続した自由局面であるが，組織の 3 次元的な空間配置を表現できるという点でボクセルを要素とした方法は，組織を多層平面で扱う MCML よりも，3 次元的な非線形パターンの海綿骨を含むモデルを扱うためには適している．

図 3.1 は，VMC の 1 つの光子パケットの伝搬挙動を計算するためのフローチャートである．VMC では，ボクセル内にて MCML と同様に，光子の伝搬挙動について乱数を用いた繰り返し計算を行う．このとき，光子は，散乱と吸収を繰り返し，またその伝搬挙動はランダムウォークと呼ばれる振動を繰り返す．ボクセルから光子が飛び出した場合，隣接するボクセルに光子を移動させる．ボクセルは 6 面体であるため，ここでの隣接するボクセルは，6 方向を考える必要がある．このような VMC の特殊性については，第 3.2.6 項に記載した．

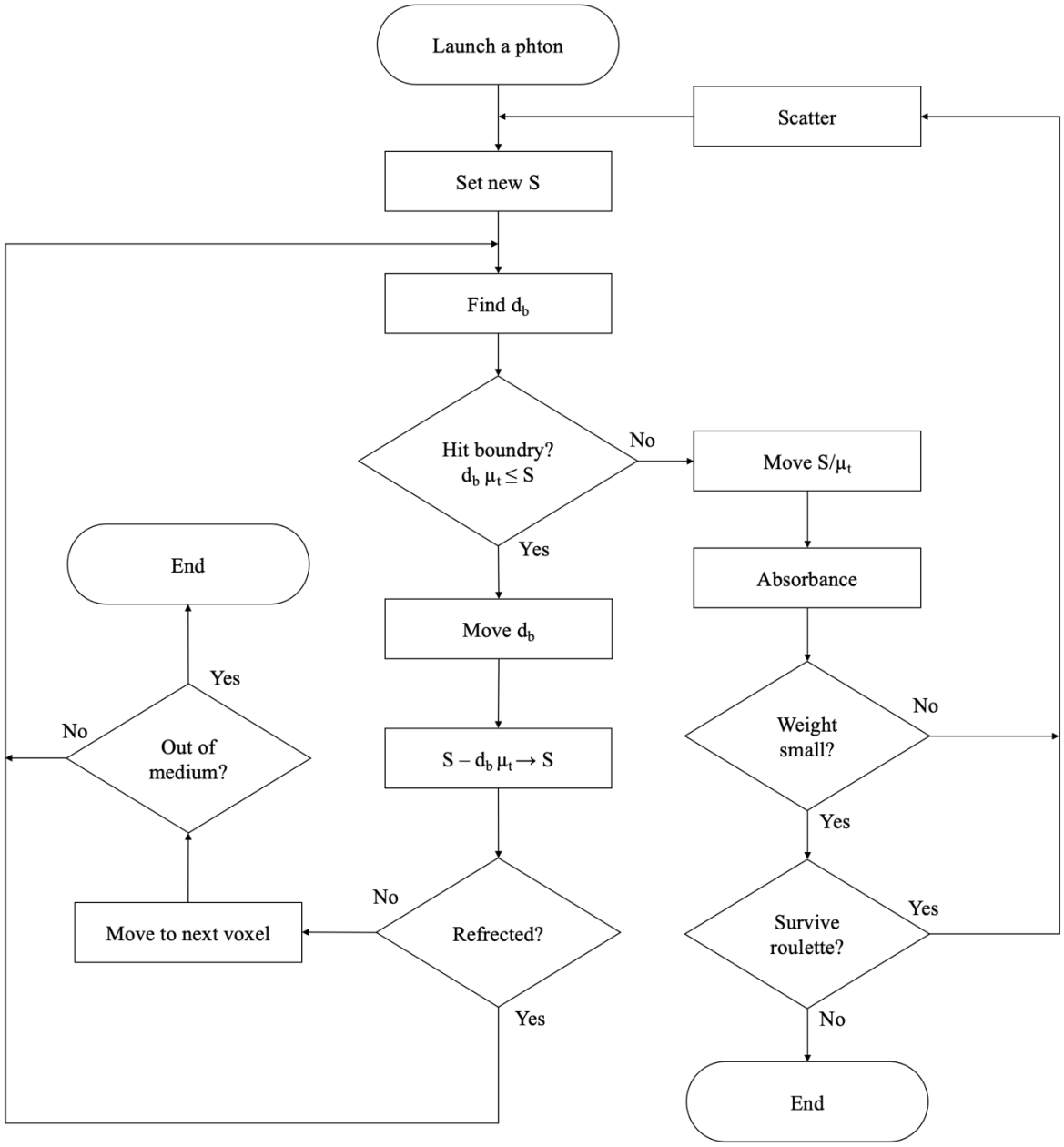


図 3.1　VMC における 1 光子の伝搬挙動を記述するフローチャート

3.2.1　光子パケットと 3 次元構造の表現

1 つの光子パケットは，4 つのパラメータを有する．1 つ目は，voxel 内部の光子パケットの位置 **p** である．**p** は直交座標系で $[p_x, p_y, p_z]$と表される．2 つ目は，光子が存在する voxel のアドレス **a** である．**a** は直交座標系で $[a_x, a_y, a_z]$と表現される．3 つ目は，光子の進行方向

μ で，方向余弦で[μ_x, μ_y, μ_z]と表される．4 つ目は，光子重量（photon weight）W である．W は，0 から 1 の値をとり，光子照射による反射や光子伝搬によって減少していく．

光子パケットの 4 つのパラメータは，各伝搬ステップにおいて，光学特性である屈折率 n, 吸収係数 μ_a，散乱係数 μ_s，異方性係数 g に影響を受ける（後述する）．吸収係数 μ_a は単位無限小経路長 (unit infinitesimal pathlength) あたりの光子吸収率，散乱係数 μ_s は無限小経路長あたりの光子散乱率として定義される．表記を簡単にするために，μ_a と μ_s の和である全相互作用係数 μ_t が使われることもある．つまり，相互作用係数は単位無限小経路長あたりの光子相互作用の確率を意味する．異方性係数 g は，偏向角の余弦値の平均値である．

VMC は最小単位要素を voxel とする 3 次元ブロックの構造体（図 3.2）に対して計算を適用するアルゴリズムである．Voxel 要素は，一辺のサイズ l_v の正六面体として定義される．各 voxel は，光学特性と紐付いた固有値とその固有値にアクセスするための 3 次元のグローバルアドレス $\mathbf{a}_g$ を持つ．例えば，ある光子パケットが [a_x, a_y, a_z] の位置にいるとき，$\mathbf{a}_g$[a_x, a_y, a_z] の固有値は，その固有値に連動した n，μ_a，μ_s，g を示す．また $\mathbf{a}_g$ の固有値には，組織外に飛び出した光子パケットの判別，プログラムを停止させるためのシグナル変数（End point）も含まれる．このシグナル変数は，End layer（図 3.2）と呼ばれる位置に光子パケットが到達した場合に真になる．End layer は，ボクセルで構成されたモデル全体を覆うように存在する層である．各 voxel 要素には，中心座標を原点とする 3 次元空間が存在し，その位置座標は前述の **p** によって決定する．つまり，組織全体に対して，i 番目のステップのある光子が存在する位置 $\mathbf{p}_{gi}$ は，

$$\boldsymbol{p}_{gi} = l_v(\boldsymbol{a}_i - \boldsymbol{C}) + \boldsymbol{p}_i \tag{4.1.1}$$

となる．ここで $\mathbf{p}_g$ はグローバル座標を意味し，**C** は組織全体に対しての座標原点のアドレスを補正するための係数である．例えば，図 4.1.2 に示すように，3 次元配列 $\mathbf{a}_g$ が 0 から 2 までの値をとる場合，組織全体の原点のアドレスが直交座標系 (x, y, z) で [1, 1, 0] であるとき，**C** は [1, 1, 0] である．つまり，VMC では，光子の伝搬挙動を voxel という最小単位で線形的に分割された計算を実施する．

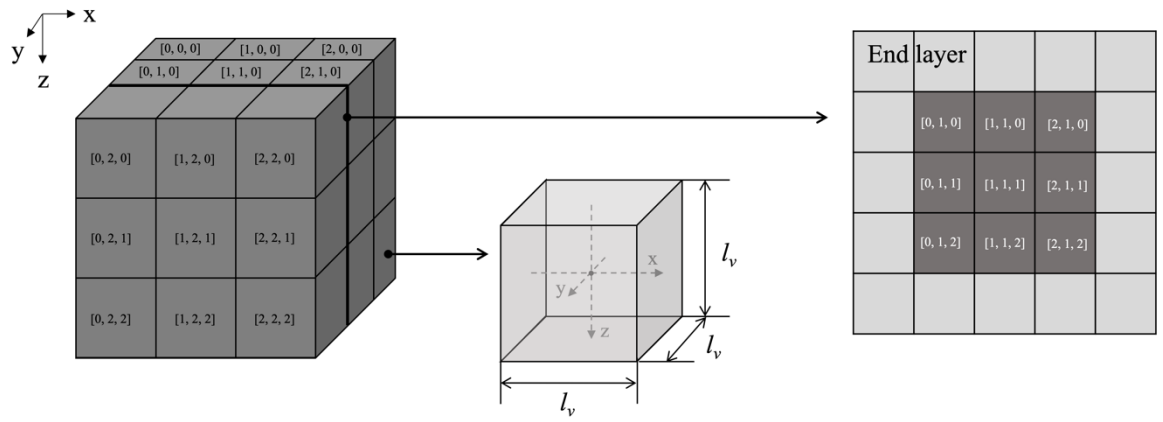


図 3.2　ボクセルを用いた 3 次元構造モデルの概略

3.2.2　光子の照射

VMC では，コリメートされた無限に細い光子ビームを想定した光子パケットを組織に入射した．光子パケットは，組織モデルの xy 平面上中心に対して直交するように照射された．このパケットの入射座標をグローバル座標で pg_0 = [0, 0, 0] とした．図 4.1.2 で示すと，ag = [1, 1, 0] かつ p = [0, 0, -l_v/2] の座標が pg_0 である．方向余弦 [μ_x, μ_y, μ_z] は，[0, 0, 1] に設定され，初期重み w_0 は 1 に初期化された．

光子が照射されたとき，組織と周囲の媒質の間に屈折率が不一致な界面が存在すれば，鏡面反射が発生する．外部媒質と組織の屈折率をそれぞれ n_1 と n_2 とすると，鏡面反射率 R_{sp} が，次式のように規定される．

$$R_{sp} = \frac{(n_1 - n_2)^2}{(n_1 + n_2)^2} \tag{3.1}$$

照射された光子が初めに衝突する媒質が透明で，その次に n_3 の屈折率がある媒質がある場合，透明な層の 2 つの境界における多重反射と透過を考慮する必要がある．この場合，鏡面反射率は次のようになる．

$$R_{sp} = r_1 \frac{(1 - r_1)^2 r_2}{1 - r_1 r_2} \tag{3.2}$$

$$r_1 = \frac{(n_1 - n_2)^2}{(n_1 + n_2)^2} \tag{3.3}$$

$$r_2 = \frac{(n_3 - n_2)^2}{(n_3 + n_2)^2} \tag{3.4}$$

ここで r_1 と r_2 は，透明層の 2 境界におけるフレネル反射率である．次に，初期値として 1 に設定された W_0 は，光子パケットが媒質に入る際に R_{sp} だけ減少し，次式となる．

$$W_0 = 1 - R_{sp} \tag{3.5}$$

3.2.3 ボクセル内の光子の移動

光子パケットのステップサイズは，光子の自由行程の確率分布のサンプリングに基づいて計算された．光子のステップサイズ S は，Wang らの MCML[77] における移動距離を用いた．

$$S = -\ln(\xi) \tag{3.6}$$

ここで，ξ は 0 から 1 の値をとる一様分布の乱数である．また，実際に光子が移動した物理量に相当するサブステップサイズ s_i は，

$$s_i = \frac{-\ln(\xi)}{\mu_t} \tag{3.7}$$

ここで μt は，$\mu_a + \mu_s$ である．また，s_i は，$0 \leq s_i < \infty$ の値をとり，乱数ξ は光子ステップごとに初期化される．

サブステップサイズ s_i が決定したら，ボクセル内の光子パケットの位置 **p** が，次式のように更新される．

$$\boldsymbol{p} + \boldsymbol{\mu} s_i \rightarrow \boldsymbol{p} \tag{3.8}$$

ここで，右方向の矢印（→）は，左辺の処理を行い，右辺の値を更新することを意味する．

3.2.4　光子の吸収

光子が相互作用位置に到達すると，その部位で吸収され，光子重量 W が減衰する．W は，次式のように更新される．

$$W\left(1-\frac{\mu_a}{\mu_t}\right)\rightarrow W \tag{3.9}$$

組織内を伝播している光子重量 w が，散乱や吸収の何段階もの相互作用の後，閾値（例えば W_{th} = 0.001）以下になった場合，それ以上の伝播はほとんど情報をもたらさない．しかし，光子の分布が偏ることなく，エネルギーを保存するためには，適切な終了処理を行う必要がある．W ≤ W_{th} のときに光子パケットを終了させるために，ロシアンルーレット [77,91] と呼ばれる手法を使用した．ロシアンルーレットでは，1/m（例えば，m=10）の確率で，光子パケットが mW の重さで生き残ることができる．

$$W'=\begin{cases} mW & if\ \xi \le \dfrac{1}{m} \\ 0 & if\ \xi > \dfrac{1}{m} \end{cases} \tag{3.10}$$

ここで，W'は更新後の W を示し，ξ は 0 から 1 の値をとる一様分布の乱数である．この方法は，エネルギーを供給しながらも，光子を偏りなく終端させることができる．組織内で消滅した光子の全パラメータは 0 に更新し，処理を終了させた．

3.2.5 光子の散乱

光子パケットが相互作用サイトに到達し，光子重量 w を更新，そして重量 w > 0 である場合，光子パケットを散乱させる．統計的にサンプリングされる偏向角（deflection angle）$\theta(0 \le \theta < \pi)$と方位角（azimuthal angle） $\psi\ (0 < \psi < 2\pi)$が存在する．偏向角の余弦 cos θ の確率分布は，Henyey-Greenstein が銀河系散乱のために最初に提案した散乱関数によって記述される．次に Henyey-Greenstein 関数の Wang ならびに Jacques らによる変形式を示す．

$$cos\theta=\begin{cases} \dfrac{1}{2g}\left[1+g^2-\left(\dfrac{1-g^2}{1-g+2g\xi}\right)^2\right] & if\ g \neq 0 \\ 2\xi-1 & if\ g=0 \end{cases} \tag{3.11}$$

ここで，異方性係数 g は，-1 から 1 の間の値を持ち，0 は等方性散乱，1 に近い値は前方指向性散乱を示す．次に，0 から 2π の区間に一様に分布する方位角ψを次式のようにサンプリングする．

$$\psi = 2\pi\xi \tag{3.12}$$

偏向角と方位角が決まれば，i 番目の光子パケットの方向は，次式のように更新する．

$$\begin{aligned}\mu_x{}' &= \frac{\sin\theta\left(\mu_x\,\mu_z\sin\theta-\mu_y\sin\psi\right)}{\sqrt{1-\mu_z{}^2}}+\mu_x\cos\theta\\ \mu_y{}' &= \frac{\sin\theta\left(\mu_y\,\mu_z\cos\psi+\mu_{x\,i}\sin\psi\right)}{\sqrt{1-\mu_z{}^2}}+\mu_y\cos\theta\\ \mu_z{}' &= \frac{-\sin\theta\cos\psi}{\sqrt{1-\mu_z{}^2}}+\mu_z\cos\theta\end{aligned} \tag{3.13}$$

ここで $\mu_x{}'$ $\mu_y{}'$ $\mu_z{}'$は，それぞれ更新後の方向余弦を示す．光子の方向が十分 Z 軸方向に近かった場合（$|\mu_z| > 0.99999$），光子パケット方向の更新は次式で行う．

$$\begin{aligned}\mu_x{}' &= \sin\theta\cos\psi\\ \mu_y{}' &= \sin\theta\sin\psi\\ \mu_z{}' &= SIGN(\mu_z)\cos\theta\end{aligned} \tag{3.14}$$

ここで，SIGN は，μ_zが正の数である場合 1 を，負の数である場合-1 を返す．

3.2.6　ボクセル間の光子の振る舞い

サイズ S のステップ中に光子パケットが境界に衝突する可能性がある．このとき境界は，定義されたボクセル間の境界であり，光子パケットは，この境界で内部反射されるか境界を越えて透過する．光子パケットのボクセル境界における振る舞いは，次に示す 4 つのステップでシミュレートされる．

ステップ 1，ボクセル境界までの距離 d_b を算出する．

$$d_b = \min\begin{cases}\dfrac{\frac{l_v}{2}-p_x\,sign(\mu_x)}{|\mu_x|}\\ \dfrac{\frac{l_v}{2}-p_y\,sign(\mu_y)}{|\mu_y|}\\ \dfrac{\frac{l_v}{2}-p_z\,sign(\mu_z)}{|\mu_z|}\end{cases} \tag{3.15}$$

ここで光子パケットは，d_b に最小値を与える軸方向と直行する面で衝突し，進行方向はその方向余弦から決定できる．例えば，p_x 並びに μ_x を含む式が，他の 2 つの式と比較して小さい値を示す場合，その光子パケットは x 軸と直行する y-z 平面に衝突し，μ_x が正の数である

場合，光子の進行方向は x 軸正方向である．d_b に最小値を与える軸方向を γ とする．γ は，d_b にて p_x 並びに μ_x を含む式が最小値をとる場合 x に，p_y 並びに μ_y を含む式が最小値をとる場合 y に，p_z 並びに μ_z を含む式が最小値をとる場合 z になる変数とする．

ステップ 2，光子パケットのステップサイズ S（式 3.6）が d_b を超えるかどうか判別する．

$$d_b\mu_t \leq S \tag{3.16}$$

式 3.14 が成立する場合，光子パケットはボクセル間の境界に衝突する．つまり，光子パケットを境界上に移動させ，サブステップサイズ S を次式のように更新する．

$$S - d_b\mu_t \rightarrow S \tag{3.17}$$

式3.14が成立しない場合，ステップは現在のボクセルで終了し，光子をSだけ移動させる．そして，吸収と散乱を計算し，S をゼロに更新する．次に，再びステップ 1 へ戻り，新しい無次元ステップサイズを$-\ln\xi$ を用いて生成する．

ステップ 3，光子パケットがボクセル境界に衝突した場合，境界で光子が透過するか，反射するのかを計算する必要がある．まず，現在のボクセルと透過後のボクセル間で屈折率 n に差がない場合，光子パケットは境界で透過する．そのため，反射率 $R(\alpha_i)$は 0 となる．次に，２つのボクセル間で，屈折率に差がある場合，ボクセル境界で光子パケットは，反射もしくは屈折する可能性がある．光子パケットが現在のボクセル内部に反射する確率は，光子が境界に入射する角度 α_i に依存する．変数 γ を用いると，α_i は次式で算出できる．

$$\alpha_i = \cos^{-1}\left(\left|\mu_\gamma\right|\right) \tag{3.18}$$

ここで，μ_γ は，式 3.15 において最小値を示す軸方向のベクトルである．また，$\alpha_i = 0$ は境界平面に対する垂直入社を示す．スネルの法則より，入射角 α_i，透過角 α_t，光子が入射する媒体の屈折率 n_i，透過する媒体の屈折率 n_t は次のような関係になる．

$$n_i \sin\alpha_i = n_t \sin\alpha_t \tag{3.19}$$

α_i が臨界角 $\sin^{-1}(n_t/n_i)$ より大きい場合，$R(\alpha_i)$ は 1 となり，光子は内部反射される．ここで，α_i が臨界角より大きくなる可能性があるのは，$n_i > n_t$ の場合のみである．それ以外の場合は，Fresnel の反射率を用いて $R(\alpha_i)$は計算される．

$$R(\alpha_i) = \frac{1}{2}\left[\frac{\sin^2(\alpha_i - \alpha_i)}{\sin^2(\alpha_i + \alpha_i)} + \frac{\tan^2(\alpha_i - \alpha_i)}{\tan^2(\alpha_i + \alpha_i)}\right] \tag{3.20}$$

本シミュレーションの光は，特定の偏光を持たないと仮定しているため，式3.18は直交する2つの偏光方向の反射率の平均値となる．

ステップ 4，算出した反射率をもとに，ボクセル境界で内部反射するか，透過するかを乱数ξで次のように決定する．

$$\begin{cases} If\ \xi \leq R(\alpha_i)\ then\ the\ photon\ is\ internally\ reflected \\ If\ \xi > R(\alpha_i)\ then\ the\ photon\ transmits \end{cases} \tag{3.21}$$

光子パケットは内部反射した場合，光子を境界上で待機させ，方向余弦を更新する．方向余弦の更新は，衝突する軸方向のベクトルに，-1を乗じたものである．つまり，変数γを用いて表現すると，方向余弦μの更新は，次のようになる．

$$-\mu_\gamma \rightarrow \mu_\gamma \tag{3.22}$$

ここでγは，前述の通り光子が衝突する軸方向を示す．方向余弦の更新後，ステップ1に戻る．

光子パケットがボクセル境界を超え透過した場合，光子が存在するアドレス a，光子の位置 p を更新し，透過前後で屈折率の異なる場合，方向余弦も更新する．アドレスは，正方向透過した場合，透過した軸方向のアドレスに1を加え，負方向に透過した場合，-1を加える．つまり，変数γを用いて表現すると，アドレス a の更新は，次のようになる．

$$\begin{cases} a_\gamma + 1 \rightarrow a_\gamma\ if\ photon\ hit\ to\ positive\ direction \\ a_\gamma - 1 \rightarrow a_\gamma\ if\ photon\ hit\ to\ negative\ direction \end{cases} \tag{3.23}$$

光子の位置 p は，透過する軸方向に-1を乗じる．従って，変数γを用いて表現すると，p の更新は次のようになる．

$$-p_\gamma \rightarrow p_\gamma \tag{3.24}$$

このとき p_γ は，必ず $l_v/2$ となる．透過前後のボクセルで屈折率 n が異なる場合，透過する軸方向の方向余弦は，次のように更新する．

$$SIGN(v_\gamma)\cos\alpha_t \rightarrow v_\gamma \tag{3.25}$$

また，それ以外の2つ軸の方向余弦 v_ε は，次のように更新する．

$$v_\varepsilon \frac{n_i}{n_t} \rightarrow v_\varepsilon \tag{3.26}$$

ここで，γ が x であるとき，ε は y 並びに z であり，γ で表される方向以外の軸方向を示す．光子が透過すると，再びステップ 1 に戻り，ステップ 1 から 4 の処理を$S - d_b\mu_t < 0$となるまで繰り返す．光子パケットのボクセル内の移動とボクセル間の移動を視覚化すると図 3.3 のようになる．

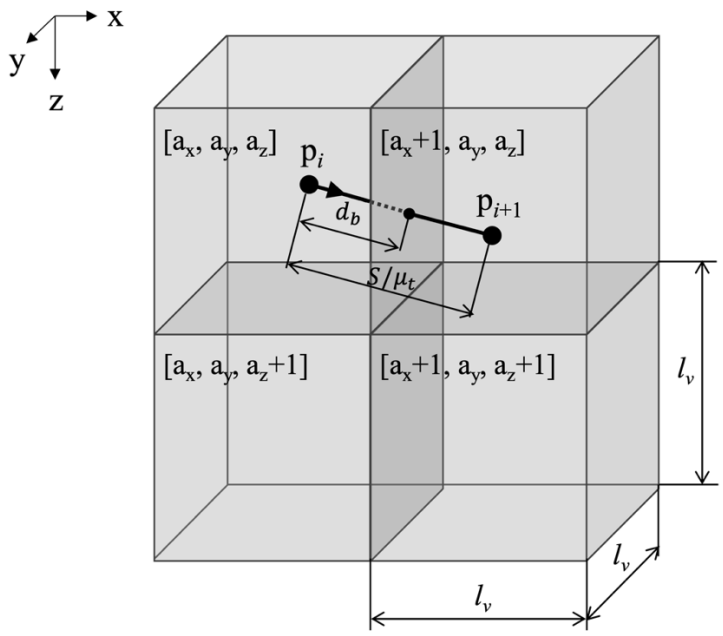


図 3.3　ボクセル間の光子パケットの転送

3.2.7　GPU を用いた並列計算

図 3.4 に GPU を用いた並列計算のフローチャートを示す．計算は，まず，各光子パケットのパラメータ W，**v**，**p**，**a** の初期値を設定し，対象となる組織のボクセルモデルを生成する．次にそれらのデータを GPU のメモリに転送し，GPU 内でモンテカルロシミュレーションを実施する．このとき GPU での計算アルゴリズムを記述したコードを vmc_kernel と呼ぶこととする．GPU での計算が終了すると，CPU で計算結果である W，**v**，**p**，**a** の戻り値を受け取り，$\mathbf{p}_g$ はグローバル座標を算出する．最終的な結果は，光子のウェイト W，光子が出てくる位置である $\mathbf{p}_g$，光子が飛び出す方向余弦 **v** が得られる．図 3.4 の CPU 計算部は，Python

3.8，GPU 計算部は，CUDA 11.7 でコーディングした．W，**v**，**p**，**a** は 32 bit，ボクセルモデルは，8 bit で定義された．

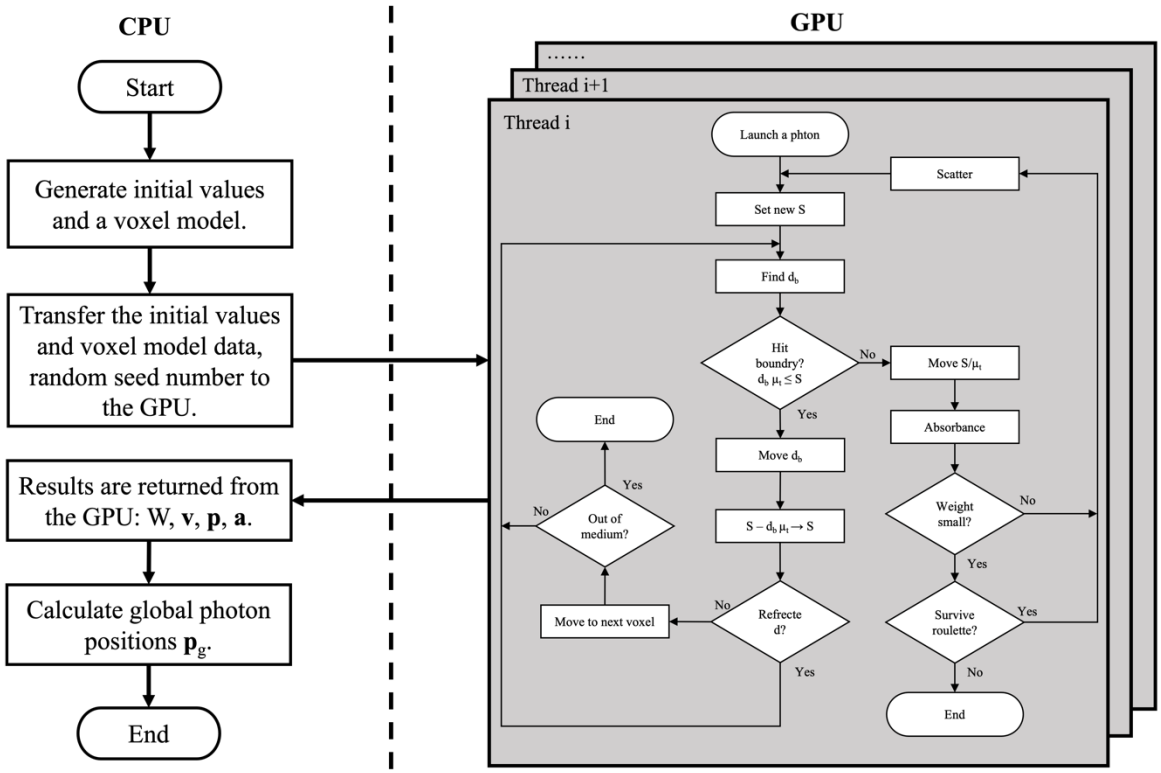


図 3.4　GPU を用いた並列計算のフローチャート

GPU での並列計算では，１つの光子パケットの伝搬挙動を１つのスレッドで計算する．光子の状態を示すパラメータである W，**v**，**p**，**a** は，光子数の長さをもつ配列として，GPU のグローバルメモリに保存されている．各スレッドは，独自のインデックスが割り振られており，インデックスごとに W，**v**，**p**，**a** の位置情報を取得する．それ以外のメモリは読み込み専用もしくは，ローカルな変数である．従って，並列計算で問題になるメモリへの書き込み競合や順序の逆転，メモリのリーク等の問題は生じない．

乱数の生成には，CUDA の cuRAND ジェネレータを用いた．cuRAND のジェネレータは，疑似乱数または準乱数のシーケンスを生成するのに必要なすべての内部状態をカプセル化する．乱数は，スレッドごとにランダムにシードされた．乱数のシード値は，CPU にて int 型の疑似乱数を生成し GPU に転送した．

3.2.8　物理量のスコアリング

光の入射方向に対し，組織外に 3 方向に飛び出した光子パケットを，後方散乱光（B），前方散乱光（F），側方散乱光（L）とそれぞれ定義した．この飛び出した光子の物理スコアは空間分解，並びに角度分解，反射率，透過率で表現した．

光入射方向に対し，組織表面から 3 方向に飛び出すパケットの光強度分布のスコアリングは，次のようになる．B と F は，図 3.5 に示すように，光子パケットの発射位置のグローバル座標（p_{gx}＝0，p_{gy}＝0）から半径方向の距離 r ごとに，Δr の範囲で光子の重みの総和 W を計算した．ここで，r は$[0, \Delta r, 2\Delta r, 3\Delta r, \ldots, n\Delta r]$の配列で表され，$r_i$ と r_{i+1} の間の範囲から W の総和が計算される．また，このとき r の座標は，$\left[\frac{1}{2}\Delta r, \Delta r, \frac{3}{2}\Delta r, \ldots, \frac{n}{2}\Delta r\right]$となり，$r_i$ と r_{i+1} の間の中心座標となる．空間分解された光強度を示す配列 B_r と F_r は，

$$B_r = \frac{I_B}{N\,\Delta A_r}\ [mm^{-2}] \tag{3.27}$$

$$F_r = \frac{I_F}{N\,\Delta A_r}\ [mm^{-2}] \tag{3.28}$$

ここで，N は総光子数であり，I_B と I_F は B と F に関する r_i ごとの W の合計を表す配列であり，ΔA_r は環状リングの面積を表し，次のように計算される．

$$\Delta A_r = 2\pi\left(i + \frac{1}{2}\right)\Delta r^2\ [mm^2] \tag{3.29}$$

L では，図 3.5 に示すように，光子パケットの照射位置のグローバル座標（p_{gy}＝0，p_{gz}＝0）から z 軸方向に正の距離ごとに，範囲 Δz で光子の重みの総和 W を算出した．すなわち，z を配列$[0, \Delta z, 2\Delta z, 3\Delta z, \ldots, n\Delta z]$で表し，$z_i$ と z_{i+1} で囲まれた範囲から W の総和を算出した．また，このとき z の座標は，$\left[\frac{1}{2}\Delta z, \Delta z, \frac{3}{2}\Delta z, \ldots, \frac{n}{2}\Delta z\right]$となり，$z_i$ と z_{i+1} の間の中心座標となる．ここでは，Y 軸方向の有効幅を±Δy とした．Z 軸方向の光強度配列 L_z は，

$$L_z = \frac{I_L}{N\,\Delta A_z}\ [mm^{-2}] \tag{3.30}$$

ここで，I_L は L に関する z_i ごとの w の総和を表す配列であり，ΔA_z は正方形の面積を表し，次のように計算される．

$$\Delta A_z = 2\Delta y \Delta z\ [mm^2] \tag{3.31}$$

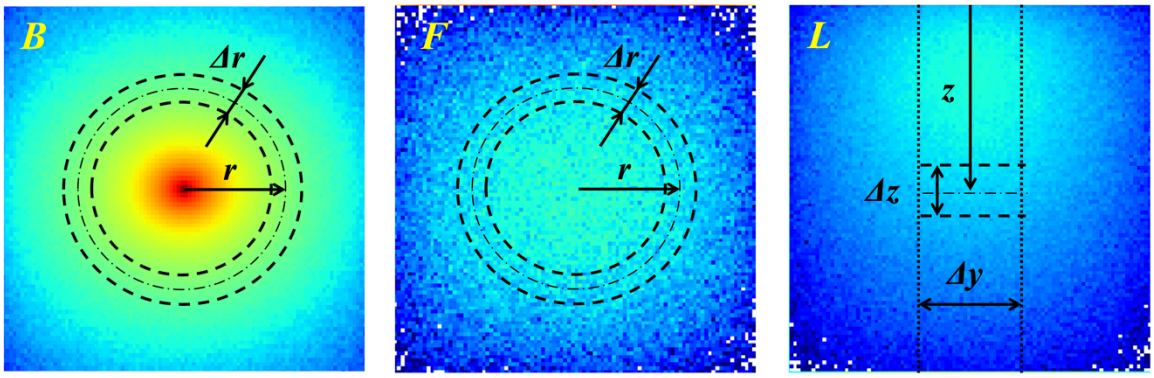


図 3.5　3 方向で計測される光強度分布のスコアリング

組織外に飛びたした光子の角度分解は，後方散乱光 B ならびに前方散乱光 F に対し導出した．角度方向の分解能 Δα は，次のようになる．

$$\Delta\alpha = \frac{\pi}{2n_\alpha} \tag{3.32}$$

ここで n_α は，$\pi/2$ の分割数を示す．従って，角度分解における各インデックス α は，$[0, \Delta\alpha, 2\Delta\alpha, 3\Delta\alpha, \ldots, n\Delta\alpha]$の配列で表され，$\alpha_i$ と α_{i+1} の間の範囲から W の総和が計算される．このとき α の座標は，$\left[\frac{1}{2}\Delta\alpha, \Delta\alpha, \frac{3}{2}\Delta\alpha, \ldots, \frac{n}{2}\Delta\alpha\right]$となり，$\alpha_i$ と α_{i+1} で定義される中心の座標となる．空間分解された光強度を示す配列 B_α と F_α は，

$$B_\alpha = \frac{I_{B\alpha}}{N\,\Delta\Omega}\ [sr^{-1}] \tag{3.33}$$

$$F_\alpha = \frac{I_{F\alpha}}{N\,\Delta\Omega}\ [sr^{-1}] \tag{3.34}$$

となる．ここで，$I_{B\alpha}$ と $I_{F\alpha}$ は B と F に関する α_i ごとの W の合計を表す配列であり，ΔΩ は立体角(solid angle)であり，次のように算出する．

$$\Delta\Omega = 4\pi \sin\left[\left(i + \frac{1}{2}\right)\Delta\alpha\right] \sin\frac{\Delta\alpha}{2}\ [sr] \tag{3.35}$$

反射率 R_d は，後方散乱光 B に関する W の総和を総光子数 N で除したものであり，透過率 T_d は，前方乱光 F に関する W の総和を総光子数 N で除したものとする．

3.3　VMC の計算例と検証

本節では，いくつかの計算結果について例として述べる．本プログラムを検証するため，いくつかの結果を他の理論や他のモンテカルロシミュレーション結果と比較した．本論でのモンテカルロシミュレーションは GPU (GeForce RTX 2070 SUPER, Nvidia, USA) にて計算し，言語は CUDA 11.4 で記述した．

3.3.1　全拡散反射率および全透過率

相対屈折率 n = 1 (屈折率整合境界)，吸収係数 μ_a = 10 cm^{-1}，散乱係数 μ_s = 90 cm^{-1}，異方性係数 g = 0.75，厚さ 0.02 cm の光学特性を持つ濁った媒体のスラブの全拡散反射率 R_d と全透過率 T_d を算出した．また，ボクセルサイズ l_v は 0.01 cm，x-y 軸方向サイズは 10 cm とした．計算は，500,000 個の光子からなる 10 回のモンテカルロシミュレーションで行った．表 3.1 に全拡散反射率ならびに前透過率の平均値と標準偏差を示す．また，表には van de Hulst の表 [77,92]，Prahl らによるモンテカルロシミュレーション [92]，Wang らの論文に記載された MCML[77] の結果も記載した．全ての結果は一致した．

表 3.1　屈折率整合境界を持つスラブにおける全拡散反射率および全透過率

Ref	R_d average	R_d error	T_d average	T_d error
van de Hulst (1980)[77]	0.09739		0.66096	
Prahl et al. (1989)[92]	0.09734	0.00035	0.66096	0.00020
Wang et al. (1992)[77]	0.09711	0.00033	0.66159	0.00049
VMC	0.09696	0.00115	0.66154	0.00168

周囲の媒質と屈折率が不一致の半無限混濁媒質についても同様に計算し，配布されている MCML の計算結果と表 3.2 で比較した．媒質は，相対屈折率 n = 1.5, μ_a = 10 cm^{-1}, μ_s = 90 cm^{-1}, g = 0 の光学的特性を持ち，l_v = 1 cm，厚さならびに x-y 軸方向サイズを 10 cm とし半無限とした．5000 個のフォトンパケットからなる 10 回のモンテカルロシミュレーションを行い，

全拡散反射率の平均と標準誤差を算出した．MCML と VMC の全拡散反射率の結果は，一致した．

表 3.2　屈折率不一致の境界を持つ半無限媒質における全散漫反射率

Ref	R_d average	R_d error
MCML	0.21983	0.00137
VMC	0.21920	0.00079

3.3.2　角度分解された拡散反射率および透過率

VMC を用いて，次の光学特性を持つ濁った媒体のスラブの角度分解された拡散反射率（後方散乱光 B_α）および 透過率（前方散乱光 F_α）を計算した．相対屈折率 n = 1, μ_a = 10 cm^{-1}，μ_s = 90 cm^{-1}，g = 0.75，厚さ 0.02 cm，x-y 軸方向サイズ 4 cm，l_v = 0.001 cm．シミュレーションでは，500,000 個の光子パケットを使用し，角度格子要素数は 30 であった．この結果を van de Hulst の表並びに MCML のデータと比較すると，図 3.6 のようになる．角度分解された拡散反射率および 透過率は，van de Hulst ならびに MCML の結果は，定量的な一致を示した．

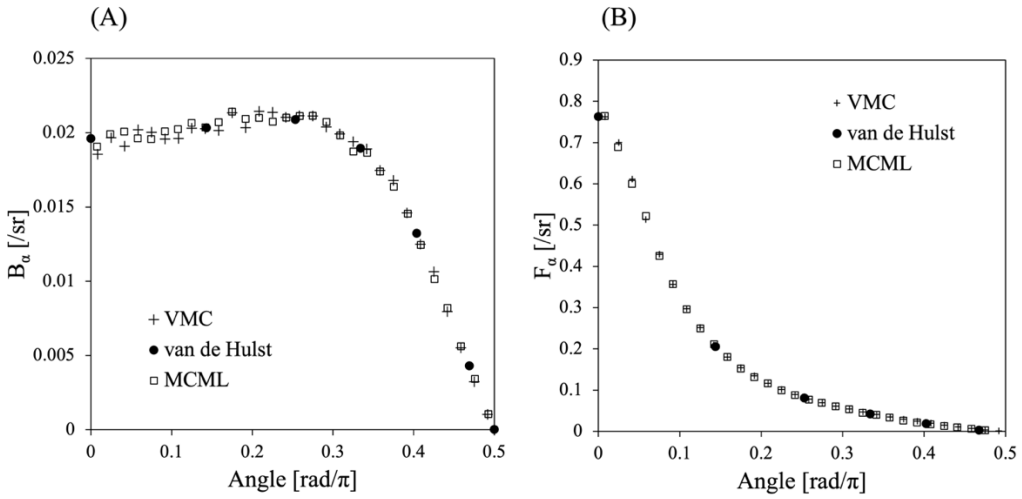


図 3.6　角度分解された(A) 後方散乱光 B_α と(B) 拡散透過率 F_α の角度 α に対する比．α は光子の出射方向と媒体表面の法線の間の角度.

3.3.3　空間分解拡散反射率および透過率

光源に対して半径方向に分解した拡散反射率（後方散乱光強度分布 B_r）のシミュレーションの例として，光学的特性が相似関係で支配される 2 つの半無限媒体を比較する．光学特性の 2 つのセットは，同じ吸収係数 μ_a および減衰散乱係数 $\mu_s(1\text{-}g)$を共有している．類似性関係が有効であれば，これら 2 つの媒体はほぼ同じ拡散反射率を与えるはずである．図 3.7 に示すように，媒質境界付近の光子源には相似関係が当てはまらないことが確認された．これは，Wyman らの予想 [93] 並びに，図 3.7 (C，D) に示す Wang らの MCML の計算結果 [90] と同様の結果である．図 3.7 において，曲線 A の光学特性は，$\mu_a = 0.1\ cm^{-1}$，$\mu_s = 100\ cm^{-1}$，$g = 0.9$，$n = 1$ である．また，曲線 B の光学特性は，$\mu_a = 0.1\ cm^{-1}$，$\mu_s = 10\ cm^{-1}$，$g = 0$，$n = 1$ である．曲線 A と曲線 B はともに，厚さならびに x-y 軸方向サイズが 20 cm の半無限遠が仮定できる媒質に対し，$1x10^6$ の光子数を用いたモンテカルロシミュレーションの結果であり，r 方向のグリッド線の間隔は 0.005 cm，グリッドの要素数は 200 である．

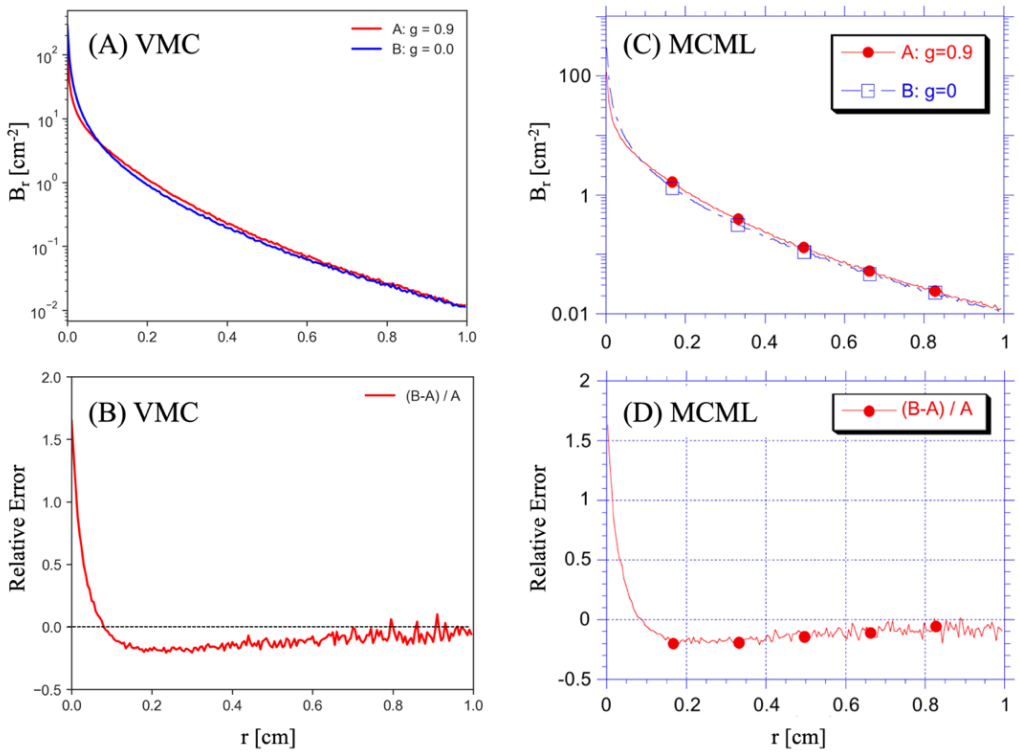


図 3.7　(A, C) 類似性関係に従って光学的特性が等価である 2 つの半無限媒体の空間分解された後方散乱光強度分布の比較と(B, D) 相対的な差．(A, B) VMC，(C, D) MCML.

3.3.4　多層構造組織の空間分解拡散反射率および透過率

VMC を用いて，多層構造の光学特性を持つ媒体の空間分解された拡散反射率（後方散乱光 B_r）および 透過率（前方散乱光 F_r）を計算した．多層構造組織の光学特性を表 3.3 に示す．また，媒質周囲環境の屈折率 n は 1 とし，媒質の x-y 軸方向サイズは 10 cm とし，l_v = 0.1cm とした．シミュレーションでは，500,000 個の光子パケットを使用し，半径方向 r 方向のグリッド間隔は 0.1 cm であった．この結果を MCML のデータと比較すると，図 3.8 のようになる．空間分解された後方散乱光 B_r および 前方散乱光 F_r は，MCML の結果は，定量的な一致を示した．VMC の計算値が多少ばらつくのは，乱数の影響が考えられる．

表 3.3　多層構造組織の光学特性

	n	μ_a [cm^{-1}]	μ_s [cm^{-1}]	g	Thickness [cm]
Layer 1	1.37	1	100	0.9	0.1
Layer 2	1.37	1	10	0	0.1
Layer 3	1.37	2	10	0.7	0.2

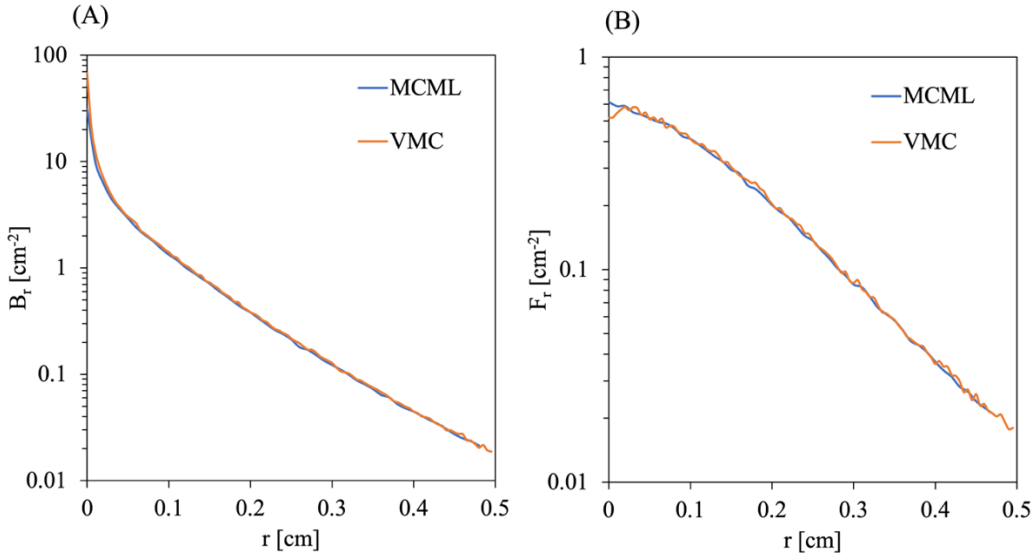


図 3.8　空間分解された(A) 後方散乱光 B_r と(B) 拡散透過率 F_r の距離 r に対する比．r は光照射位置からの半径方向距離.

3.3.5　ボクセルサイズと計算誤差・計算速度

VMC の計算対象媒質は，ボクセルで構成されるため，そのボクセルサイズ l_v によって計算誤差が生じる可能性がある．VMC を用いて，l_v が空間分解された後方散乱光 B_r に及ぼす影響を，次の光学特性を持つ濁った媒体のスラブについて調査した．相対屈折率 n = 1.5, μ_a = 0.01 mm^{-1}，μ_s = 10 mm^{-1}，g = 0.9，厚さ並びに x-y 軸方向サイズ 5 mm とした．シミュレーションでは，1,000,000 個の光子パケットを使用し，半径方向 r 方向のグリッド間隔は 0.01 mm であった．図 3.9 に，ボクセルサイズ l_v が媒質サイズと同等の 5 mm の場合と，1/1000 倍である 0.005 mm のときの空間分解された後方散乱光強度分布を比較した．ここで，モンテカルロシミュレーションの乱数シードはボクセルサイズに寄らず一定である．図 3.9（A）によると 2 つの曲線は，十分一致しているように見える．また，(B) の相対的な差では，Br は±0.002 から 0.001 程度の不規則な誤差が生じていることがわかる．この誤差は，ボクセル境界の計算により生じたステップ差，並びに 32bit の計算精度の丸め込み誤差に起因すると考えられる．しかしながら，計算結果は十分に一致し，ボクセルサイズの変化による計算誤差は十分に小さいと言える．

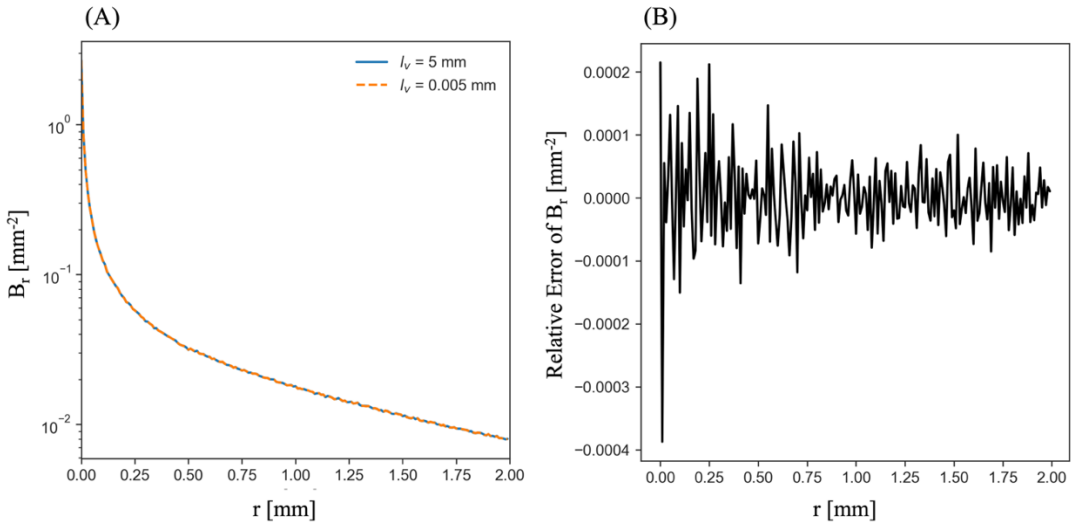


図 3.9　(A) ボクセルサイズ l_v が異なる等価の 2 つの半無限遠懸濁媒体の空間分解された後方散乱光強度分布の比較と (B) 相対的な差.

図 3.9 と同様の光学特性並びに大きさをもつ懸濁媒質について，l_v が計算速度に及ぼす影響について調査した．ここで，計算速度は計算に用いた光子数を時間で除したものである．

シミュレーションでは，500,000個の光子パケットを使用した．CUDAでは，GridとBlockに基づいてThreadを計算する．ここで，Grid数 N_g は光子数Nについて次のように定義した．

$$N_g = \frac{N-1}{128} + 1, N_g \in \mathbb{Z} \tag{3.36}$$

また，Block数 $N_b = 128$ とした．図3.10に l_v を0.005 mmから5 mmに変化させた場合の計算速度の変化を示す．基本的には計算速度は，l_v が小さいほど減少する．これは，ボクセル境界の判別や境界までの距離を計算するのにある程度の計算リソースを消費するためであると考えられる．ただし，図3.10にて $l_v = 1$ mm付近に計算速度のピークが存在するように，l_v が大きなモデルが必ずしも最も早い計算速度を示すわけではないことも推察される．

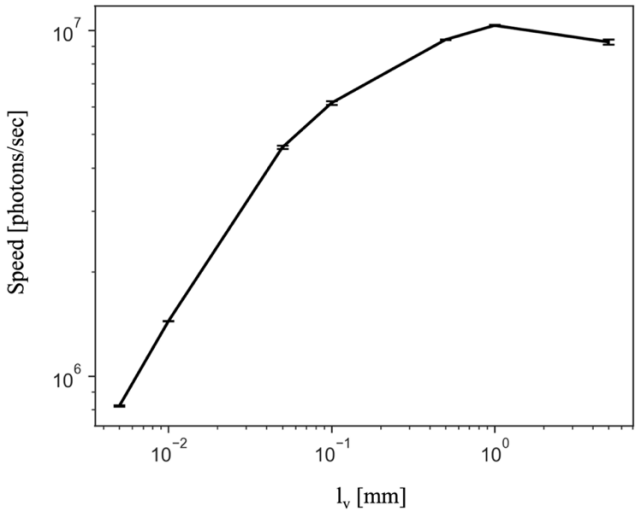


図 3.10　ボクセルサイズ l_v による計算速度の変化

3.3.6　光子数と計算速度

モンテカルロシミュレーションでは，光子数が増大することによって計算時間が増加する．しかしながら，大量の光子を並列で計算する GPU を用いた手法では，光子数と計算速度は一定の関係にはない．本項では，VMC を用いて，光子数が計算速度に及ぼす影響を，次の光学特性を持つ濁った媒体のスラブについて調査した．相対屈折率 $n = 1.5$, $\mu_a = 0.01$ mm^{-1}, $\mu_s = 10$ mm^{-1}，$g = 0.9$，$l_v = 0.5$ mm，厚さ並びに x-y 軸方向サイズ 5 mm とした．シミュレーションでは，10^3 から 10^8 個の光子パケットを使用し，計算は乱数シードを変更して 3 回実施した．図 3.11 (A) に，光子数と計算時間の関係を示す．光子数が増大するごとに計算時間は増

加を示す．ただし，図からも分かるように，光子数が 10^3 個と 10^4 個に計算時間の差はほとんどない．これは，本法にて一括に並行して計算できるスレッドの最大数が 128×128 個であるため，この範囲内であれば計算時間は，モンテカルロのステップ数に依存するためであると考えられる．また，本結果では 10^8 個の光子数による計算時間は 20 秒程度であり，非常に高速であることも注目すべき点である．加えて，CPU と計算時間を比較した場合 10^7 以上の光子数では，GPU による計算時間はおよそ 1/40 倍である．体感的にいうとこれは，CPU では 1 月かかる計算が，GPU では 1 日もかからず終了できることを示している．図 3.11 (B) に，光子数と計算速度の関係を示す．この場合，計算速度は計算効率を意味する．計算速度は，光子数が増加するごとに増加する．そして，光子数 10^7 個ほどで，その増加率は減衰し，計算速度はほぼ一定となる．従って，式 3.36 のように Grid 数並びに Block 数を設定した VMC では，10^7 個以上の光子数で最も計算効率が良いことがわかる．一方，CPU での計算速度は，光子数によらずほとんど変わらず 10^4 個以下の光子数では，むしろ GPU よりも高速である．これは，GPU の性質に起因する．GPU では，大量のプロセッサーを並列に計算させるため，モンテカルロシミュレーションのような繰り返し計算が多いアルゴリズムでは，非常に高速になる．しかしながら，光子数が少ない，つまり繰り返し計算が少ない場合，あまり効率的に計算できないのである．よって，大きな光子数で計算したい場合は GPU を，小さな光子数では CPU の方を使用した方が効率は良いこととなる．

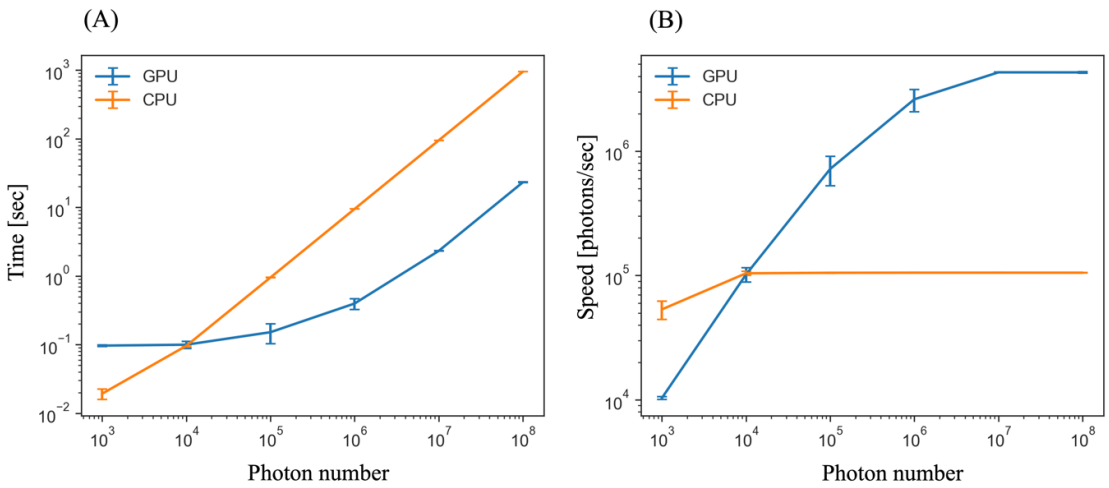


図 3.11　GPU と CPU での計算速度の比較．(A) 光子数の増加に伴う計算時間の変化，(B) 計算スピードの変化

3.3.7　VMC を用いた合成生体組織モデルの計算例

本項では，合成生体組織モデルにおけるモンテカルロシミュレーションの計算例を示す．合成生体組織モデルは，第 2 章で説明した海綿骨組織モデルの周囲に，皮下組織層，真皮層を追加したものである（図 3.12）．合成生体組織モデルに対し，VMC にて空間分解された後方散乱光強度 B_r，側方散乱光強度 L_z，前方散乱光強度 F_r の分布を算出した．なお，合成生体組織モデルは，橈骨最遠位端を前提としており，海綿骨組織は，第 2.3 節で説明したものと同等のものである．

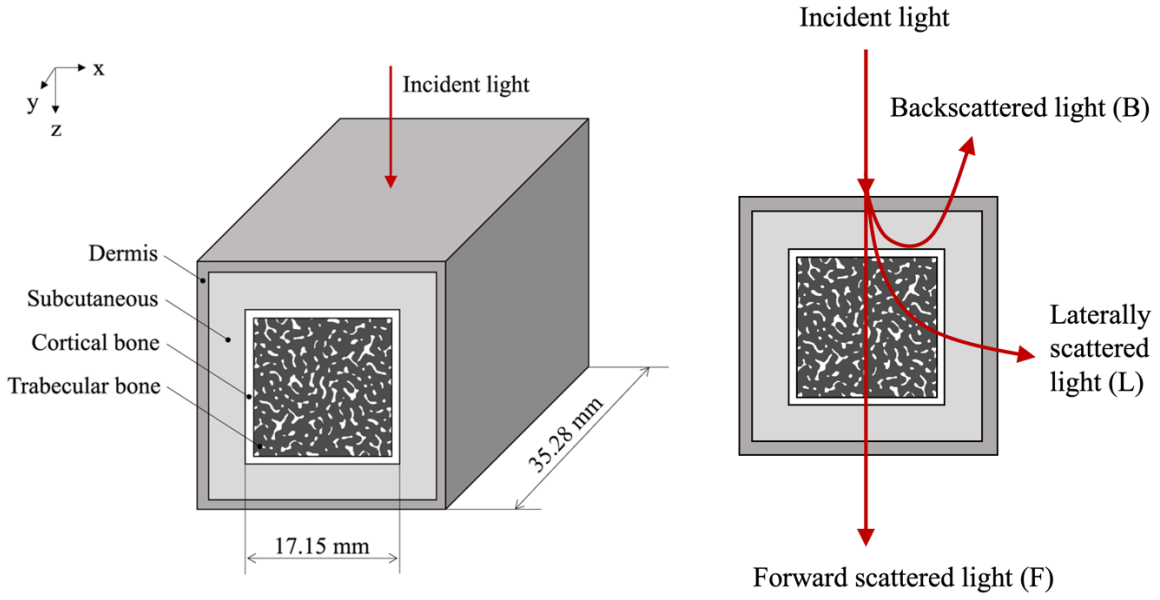


図 3.12　合成生体組織モデル（左）とモンテカルロシミュレーションにおける計測方向．

図 3.13 にモンテカルロシミュレーションで算出された，B_r が試料表面に形成する光強度分布を示す．図 3.13 では，(A) aBMD の変化，(B) 皮膚厚の変化，(C) 真皮の光学特性の変化，(D) 皮下組織の光学特性の変化に対する拡散光強度分布の変化を示している．図 3.12 における (A) から (D) の計算条件は以下の通りである．(A) 真皮の厚さは 1.2 mm，皮下の厚さは 2.0 mm，真皮の μ_s と μ_a はそれぞれ 17.6 mm^{-1} と 0.012 mm^{-1}，皮下の μ_s と μ_a は 11.1 mm^{-1} と 0.009 mm^{-1} であった．(B) aBMD は 0.59 g/cm^2，真皮と皮下の光学特性は (A) と同じであった．(C) の aBMD は，(B) と，皮膚の厚さおよび皮下の光学的性質は (A) と同じであった．(D)の aBMD は(B)と，皮膚の厚さおよび真皮の光学的性質は(A)と同じである．全ての計算におい

て，骨髄空間の光学特性は，皮下組織と同じ値とした．B_r は半径方向 r の増加に従い減少傾向を示した．aBMD の変化に対して，B_r は明らかな違いを示さなかった（図 3.13 (A)）が，B_r は皮膚厚と皮膚の散乱係数に明白な変化を示した．この結果では，作図による視覚化の段階では，B_r は真皮と皮下組織を含む皮膚厚やその光学特性の変化と比較して，骨密度による変化が明らかに小さいことがわかる．従って，B_r のみを用いた aBMD の推定は困難であること推測される．

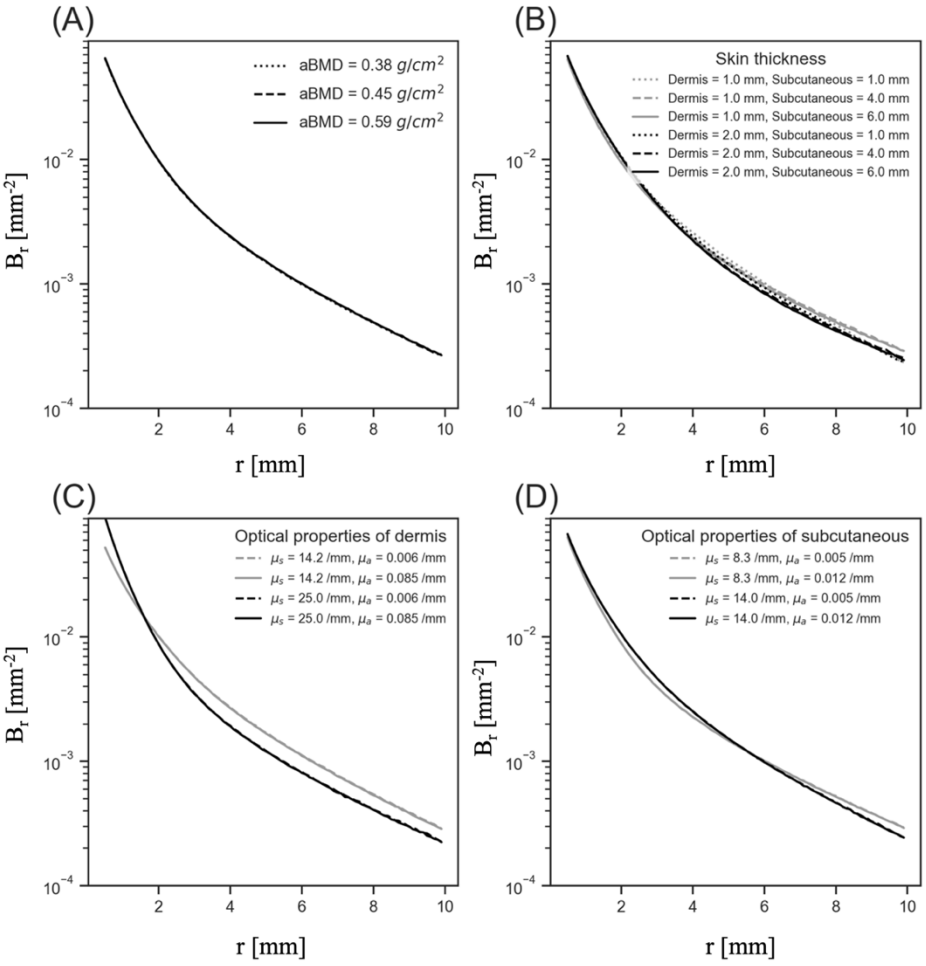


図 3.13　シミュレーション条件による光強度 B の半径方向変化．(A)aBMD に対する変化．(B) Dermis 並びに subcutaneous の厚さに対する変化．(C) Dermis の光学特性に対する変化．(D) Subcutaneous の光学特性に対する変化．

図 3.14 にモンテカルロシミュレーションで算出された，L_z が試料表面に形成する光強度分布を示す．図 3.14 では，(A) aBMD の変化，(B) 皮膚厚の変化，(C) 真皮の光学特性の変化，(D) 皮下組織の光学特性の変化に対する拡散光強度分布の変化を示している．図 3.14 におけ

る (A) から (D) の計算条件は，図 3.12 と同様である．L_z は z の増加に伴い増加し，ピークに達し，その後減少傾向を示す．L_z は，aBMD，皮膚厚，皮膚の光学特性の変化に対し反応を示した．特に，皮膚厚の変化に対する L_z のピーク後の傾きの変化は特徴的である．また，B_r の強度分布が真皮並びに皮下組織の散乱係数等の皮膚の変化に対して，比較的大きな変化を示すのとは異なり，L_z は皮膚の吸収係数に対しても散乱係数と同等のオーダーで変化を示した．L_z は B_r と比較して，骨密度の変化に対する光強度分布の変化が顕著であった．しかしながら，その変化量は，皮膚の高極特性並びに厚さによる変化量と比較して小さいことがわかる．

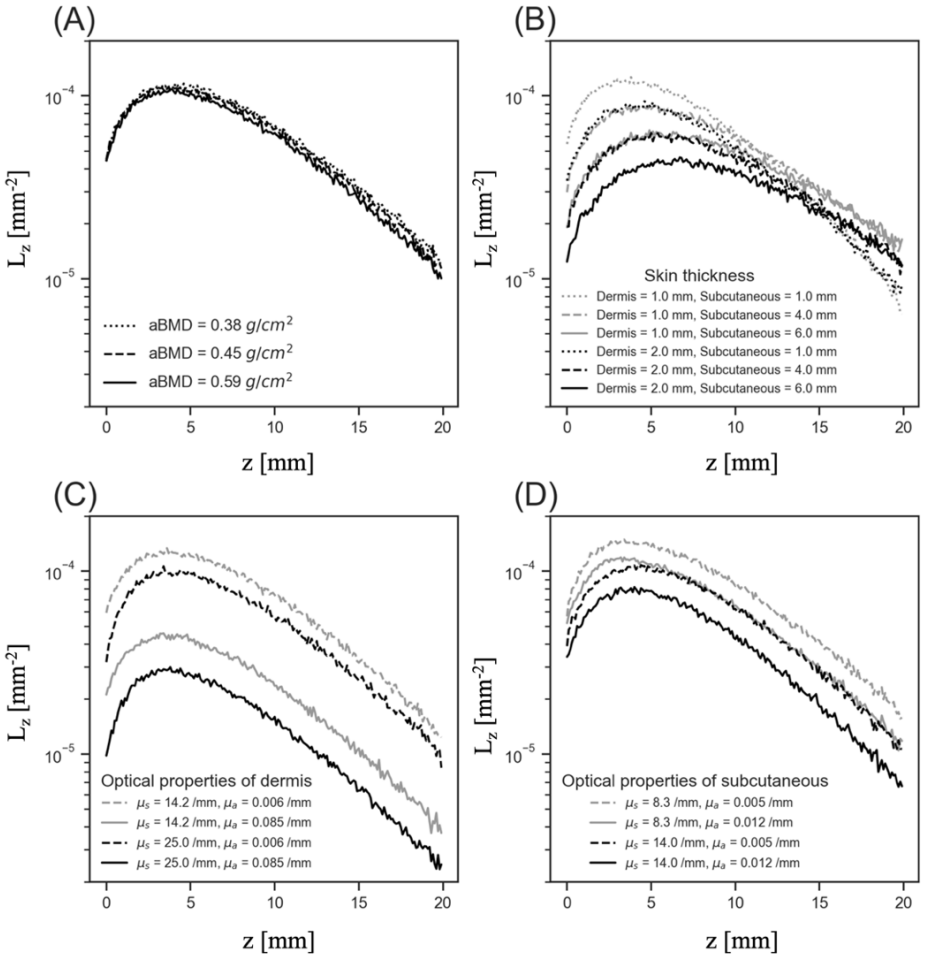


図 3.14　シミュレーション条件による光強度 L の半径方向変化．(A)aBMD に対する変化．(B) Dermis 並びに subcutaneous の厚さに対する変化．(C) Dermis の光学特性に対する変化．(D) Subcutaneous の光学特性に対する変化．

図 3.15 にモンテカルロシミュレーションで算出された，F_rが試料表面に形成する光強度分布を示す．図 3.15 では，(A) aBMD の変化，(B) 皮膚厚の変化，(C) 真皮の光学特性の変化，(D) 皮下組織の光学特性の変化に対する拡散光強度分布の変化を示している．図 3.15 における (A) から (D) の計算条件も図 3.13 と同様である．F_rは，半径方向 r の増加に伴い減少傾向を示すものの，顕著な分布形状を形成しなかった．組織モデルの背面に透過した光は，必ず骨を通過するため，骨密度の増加に従い，光強度は減少傾向を示すことが予想される．予想通り F_r は，aBMD が増大するにつれ，F_r が小さくなる負の関係を示した．しかしながら，F_r は皮膚厚並びに皮膚の光学特性の変化に対しても敏感に反応し，強度分布の変化には特徴的な形は確認されなかった．

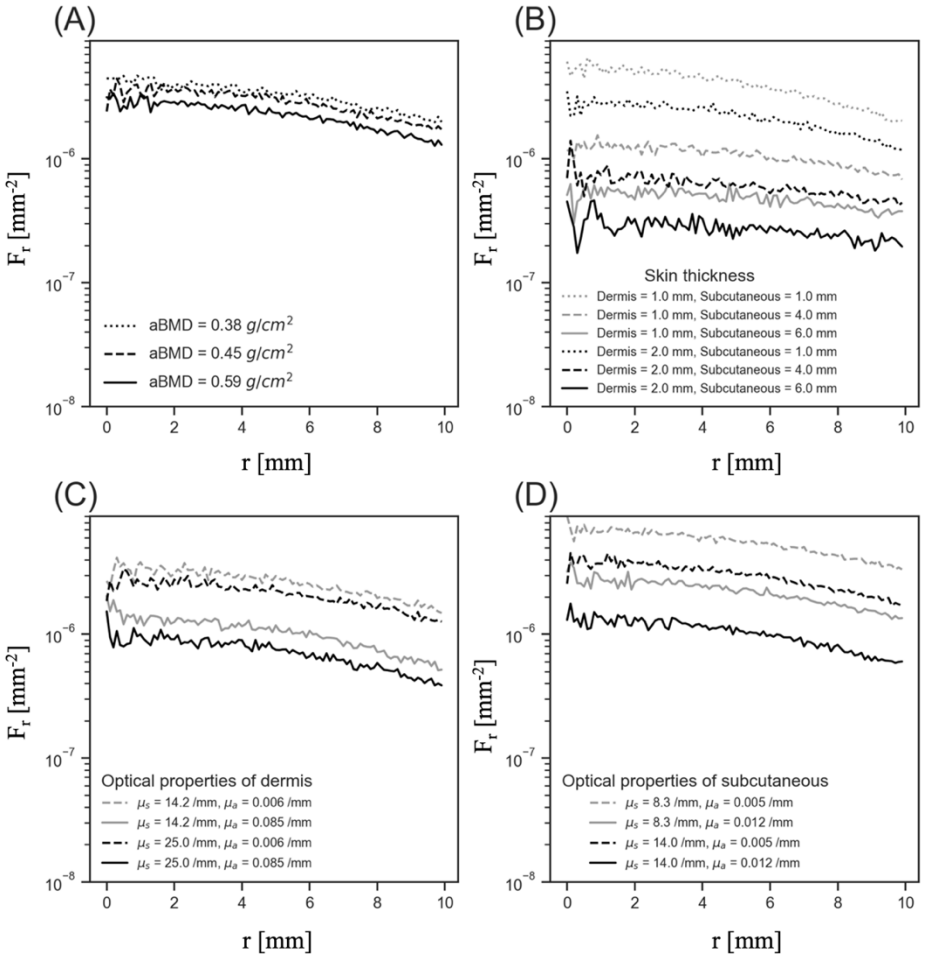


図 3.15　シミュレーション条件による光強度 F の半径方向変化．(A)aBMD に対する変化．(B) Dermis 並びに subcutaneous の厚さに対する変化．(C) Dermis の光学特性に対する変化．(D) Subcutaneous の光学特性に対する変化．

3.3.8　計測方向と各組織から受ける影響の検証

多方向光学式骨密度計測法のコンセプトの 1 つは，光入射に対して，計測対象の後方，側方，前方方向という 3 つの方向から光を取得することで，十分な骨の情報取得することであった．そのため，光を取得する方向によって，どの程度骨や皮膚からの情報が得られるのかを調査する必要がある．本研究では，それぞれの組織からどの程度情報が得られたかを，光子がそれざれの組織からどの程度影響を受けたかという視点で定義した．

モンテカルロシミュレーションでは，光子パケットが相互作用サイトに到達した場合に，相互作用サイトにある組織から吸収ならびに散乱の影響を受ける．相互作用サイトをよりわかりやすく表現すると，光子パケットの無次元ステップサイズ S が 0 になる地点，つまり光子パケットのステップの終端である．従って，各組織に対するステップ数をカウントすることで，ある光子パケットが伝搬経路の中，どの組織からどの程度影響を受けたかが推定できる．本項では，光子パケットのステップ数を利用して，取得される光が各組織から受ける影響度合について考察する．

後方，側方，前方方向から計測された光が，合成生体組織モデルの各組織からどの程度影響を受けているか調査した．本項では，モンテカルロシミュレーションを適用した合成生体組織モデルは，図 3.12 に示したものである．表 3.4 に計算に適用した光学特性並びに組織の構造特性を示した．各組織の光学特性は，Simpson[94] の白人に対する計測値を用い，軟組織の厚さは標準的な体型 [95,96] とした．また，BV/TV は，骨粗鬆症における健常者の平均値であり [83]，骨の光学特性は 12 g/cm^3 の値を用いた [18]．シミュレーションでは，100,000 個の光子パケットを使用した．

表 3.4　合成生体組織の光学特性と構造特性

	n	μ_a [mm^{-1}]	μ_s [mm^{-1}]	g	Thickness [mm]	BV/TB [%]
Dermis	1.4	0.0120	17.64	0.9	1.500	
Subcutaneous	1.4	0.0090	11.13	0.9	2.000	
Cortical bone	1.55	0.0237	20.58	0.9	0.804	
Trabecular bone	1.55	0.0237	20.58	0.9		13.4

図 3.16（A）は，3 方向へ飛び出した全光子パケットが，各組織からどの程度影響を受けているかを示したものである．図3.16の縦軸は，光子パケットの総ステップ数に対して，それぞれの組織内（真皮，皮下組織，骨）のステップ数を百分率で表示したものである．また，箱髭図の箱内の横線は中央値，箱の縦方向両端は第 1 並びに第 3 四分位点，エラーバーは最大最小値をそれぞれ意味する．後方散乱した光子パケットは，真皮に最も強い影響を受け，骨からの影響は大きくないことがわかる．一方，側方ならびに前方散乱した光子パケットは，その伝搬過程において，骨からの影響を強く受けることがわかる．図 3.16 の(B)から(D)は，それぞれの方向から飛び出す光子を空間分解し，それぞれの組織からの影響度合を表現したものである．ここで，エラーバーは，標準偏差を示す．(B) の後方散乱光では，光源付近では，最も表層の真皮による影響が大きく，総ステップ数に占める真皮内でのステップ数はおよそ 90%前後であった．そして，半径方向に向かうに従い，皮下組織並びに骨による影響も徐々に大きくなり，r = 10 mm では，全ての組織におけるステップ数は，全ステップ数のおよそ 40%となる．これは，光源と光子を検出する位置が大きくなるにつれて，より深層の情報が多く得られることを示唆している．(C) の側方散乱では，総ステップ数に占める骨内のステップ数の割合はおよそ 60 %前後であり，皮下組織並びに真皮では 20 %程度であった．このように，後方散乱と比較して，側方散乱では骨の情報が多く得られていることがわかる．また，側方散乱では，z の位置が変化しても，それぞれの組織におけるステップ数は大きく変化しないこともわかる．(D)の前方散乱では，側方散乱よりも，骨内でのステップ数が多く，全体の 70%程度を示す．また，エラーバーも側方散乱と比較し小さい．真皮と皮下組織内でのステップ数は，10 から 20%程度であった．これらの(B)から(D)の図は，それぞれが連続した組織であると考えるとわかりやすい．光入射位置から遠くに離れるに従い，徐々に全ステップ数を占める骨組織内でのステップ数が増大していく．

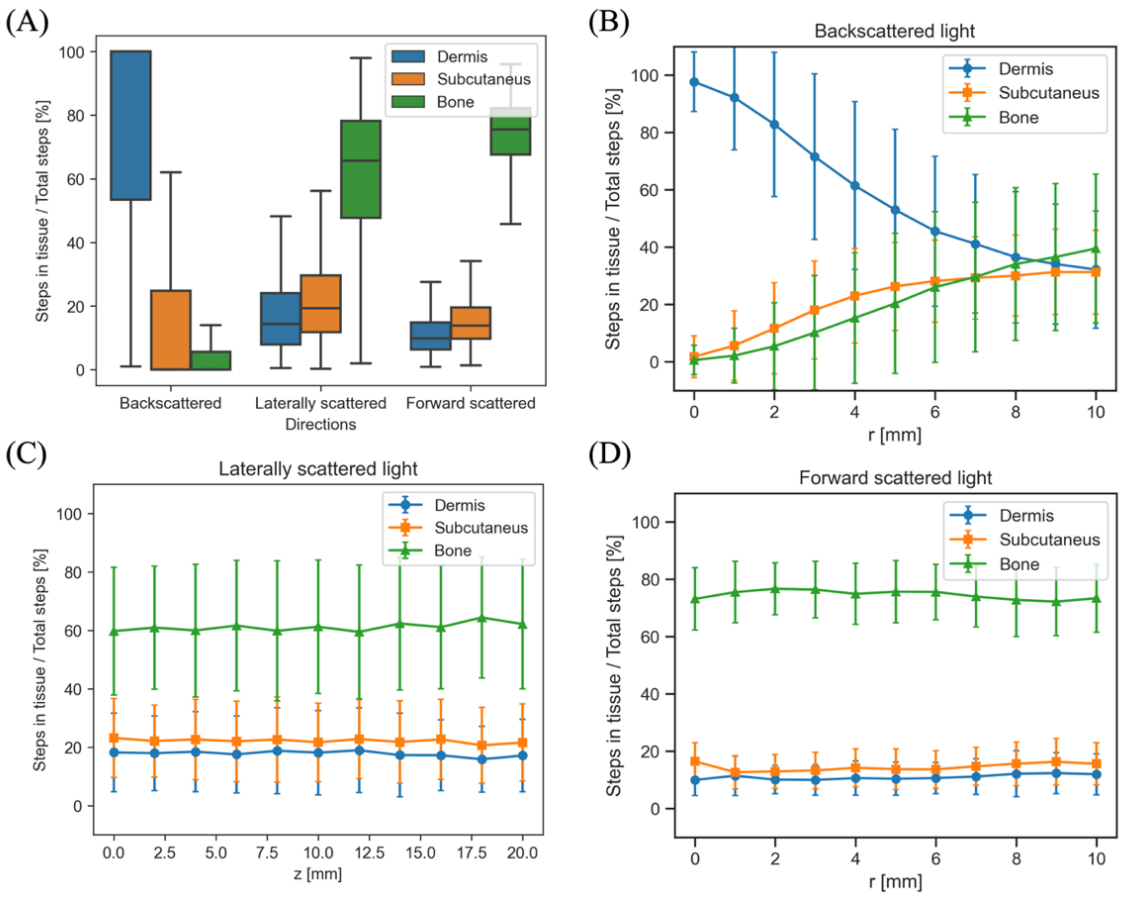


図 3.16　光子パケットの総ステップ数に対する，真皮（1.5 mm），皮下組織（2.0mm），骨内のステップ数．（A）後方，側方，前方方向で取得される全光子パケットの分布．（B）後方方向（C）側方方向（D）前方方向に対する空間分解で得られた光子の平均と分散．組織の光学特性と構造特性のシミュレーション条件は表 3.4 に記載した．

図 3.16 の結果は，標準的な皮膚厚に対して行ったシミュレーションであり，厚い皮膚でも同様の結果が得られるかは不明である．表 3.4 の構造特性に対し，真皮の厚さを 2.0 mm，皮下組織の厚さを 6.0 mm とした場合の総ステップ数に対する，真皮，皮下組織，骨内での光子パケットのステップ数を図 3.17 に示した．図 3.17 は，図 3.16 と比較して，相対的に皮下組織による影響が大きくなり，それに伴って，骨による影響が小さくなっていることがわかる．特に側方方向は，皮下組織による影響が増大していることがわかる（図 3.17（A）（C））．この原因は，図 3.16 と比較して，皮下組織厚さは 3 倍であるためである．一方，前方散乱光では，総ステップ数に占める骨内のステップ数の割合はおよそ 50 から 60 %であり，骨の影響が皮下組織の影響（30 から 40 %）を上回っている．つまり，真皮と皮下組織

を合わせて厚さが 8 mm であったとしても，前方に透過する光からは骨の情報が得られると考えられる．

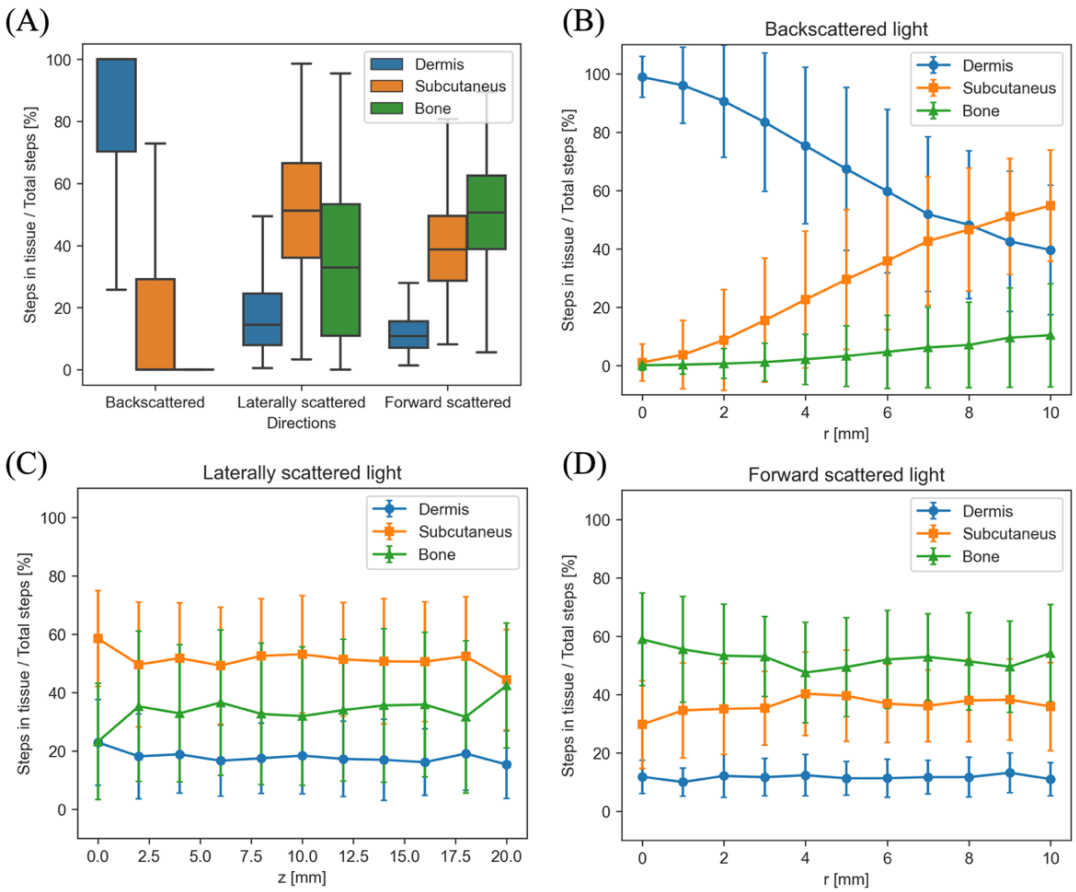


図 3.17　光子パケットの総ステップ数に対する，真皮（2.0 mm），皮下組織（6.0mm），骨内のステップ数．（A）後方，側方，前方方向で取得される全光子パケットの分布．（B）後方方向（C）側方方向（D）前方方向に対する空間分解で得られた光子の平均と分散．真皮と皮下組織厚さ以外の組織の光学特性と構造特性のシミュレーション条件は表 3.4 に記載した．

図 3.16 並びに図 3.17 の結果より，3 方向から光を取得することにより，後方方向のみから取得することと比較して，より多くの骨情報が得られることがわかった．また，どの組織からどの程度の情報が得られるかは，光の照射位置と取得位置の距離に関係しており，基本的に，照射位置から離れた位置では，より深層の情報が多く得られる．ただし，深層の情報が多く得られることは，骨密度が予測できることとは異なることに注意が必要である．

3.4 結言

本章では，まずVMCの計算アルゴリズムを説明した．VMCは，WangらのMCMLを拡張し，計算対象モデルを正6面体のボクセルで構成することで，3次元構造体にもモンテカルロシミュレーションを適用できるようにしたものである．

次に，VMC の計算例を示し，プログラムを検証するため，いくつかの結果を他の理論や他のモンテカルロシミュレーション結果と比較した．検証の結果，VMC は，過去に発表されたモンテカルロシミュレーションと同等の結果が得られることがわかった．また，単位要素であるボクセルのサイズを変更しても，単一媒質で同様の光学特性を持つ場合，ボクセルサイズに依存せず，同様の結果が得られた．また，VMC の計算速度も調査した．ボクセルサイズが小さくなるに従って，VMC の計算速度は急激に低下した．一方，光子数が増加するに従い，計算速度も増加傾向を示した．VMC を用いて第2章で説明した海綿骨組織モデルを皮下組織と真皮で覆った合成生体組織モデルの計算例を示した．全てのシミュレーションで算出された結果は，皮膚の厚さ並びに光学特性の変化に対して，顕著な変化を示した．骨密度の変化に対する算出された光強度分布の変化は，皮膚によるものと比較して微小であった．このように，光入射に対する応答のみから骨密度を推定することは，皮膚の影響を受け困難であることがわかる．後方，前方，側方から出てくる光子パケットに対して，どの組織からどの程度の影響を受けるかも調査した．この調査によると，異なる方向で取得される光は，それぞれの組織に対して異なる影響を受けた光であることがわかる．つまり，これらを組み合わせることで，軟組織厚さと光学特性の変化情報も取得することが可能になり，結果的に，骨密度の推定精度が向上することが考えられる．

第4章では，機械学習を用い，個人によって異なる皮膚の影響にロバストな骨密度計測方法を提案する．

第 4 章

骨密度予測のための機械学習モデルの構築とその推定精度

4.1　緒言

第 2 章と 3 章では，チューリングの反応拡散モデルを用いた海綿骨モデルの生成法，並びにボクセルベースのモンテカルロ法のアルゴリズムについて説明した．本章では，本論の核である，機械学習を用いた骨密度予測法について記述する．

4.2　機械学習を用いた多方向骨密度予測方法

モンテカルロ法で算出した組織表面上に析出する光強度から機械学習技術を用いてこつ密度の予測を行なった．システム自体の概要を図 4.1 に示す．本法では，まず，密度可変の海綿骨モデルに，ランダムに構造特性並びに光学特性を決定した皮膚や皮下組織を定義し，合成した合成生体組織モデルを生成する．次に，そのモデルに対し，モンテカルロシミュレーションを実施し，合成生体組織モデルの表面上に 3 方向から観測される光の強度分布を算出する．そして，モンテカルロシミュレーションの結果から，機械学習に入力するための特徴量ベクトルを生成する．最後に，機械学習モデルを用いて面積骨密度 aBMD を予測する．本セクションでは，システムを構成する 4 つの段階と，光強度と aBMD を関連付ける関数を推論するモジュールについて説明する．

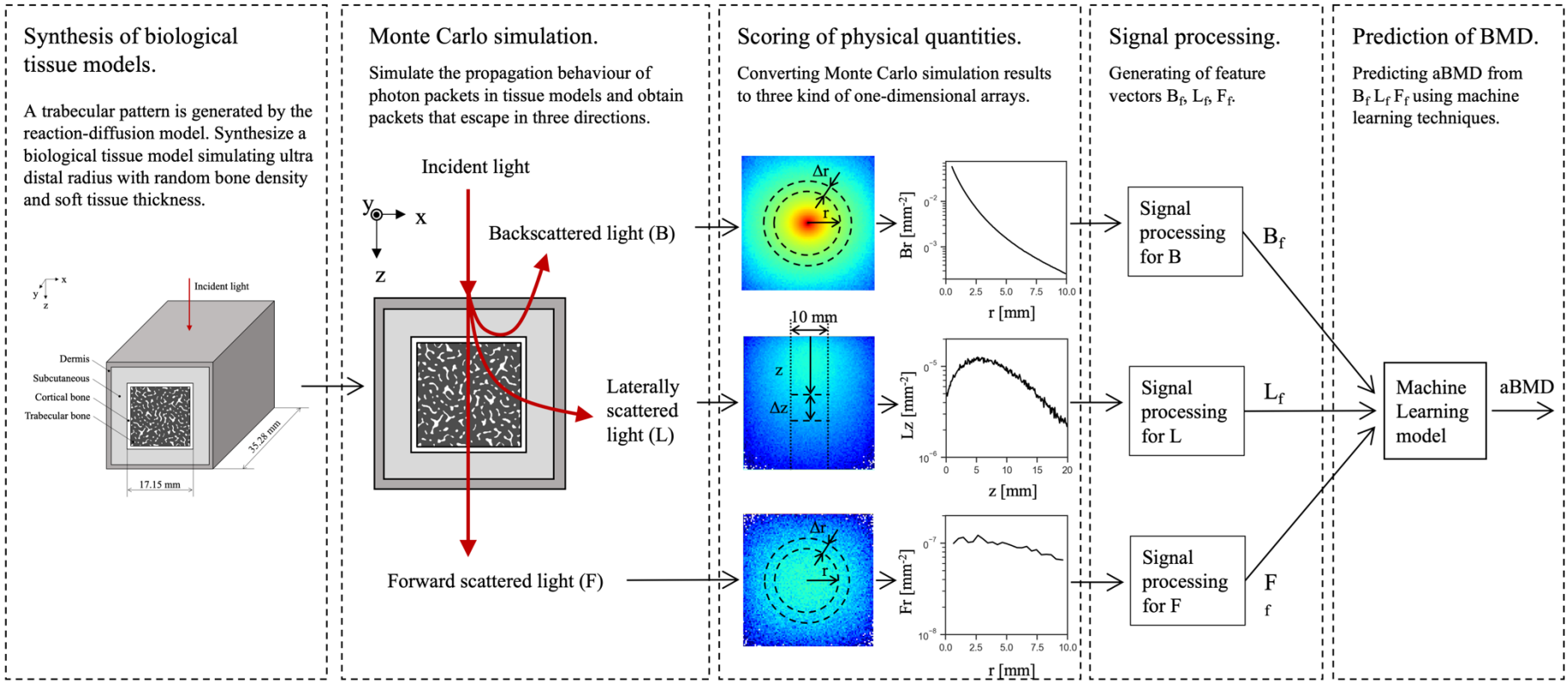


図 4.1　モンテカルロ法によるシミュレーションデータを用いた機械学習モデルによる骨密度予測手順

4.2.1　生体組織モデルの合成

合成生体組織モデルの作成手順を図 4.1 に示す．生体組織モデルは，骨組織，皮下組織，真皮から構成されている．骨組織は，皮質骨，海綿骨，骨髄から構成されている．本節では，生体組織モデルを生成するための 4 つのステッ プを説明する．

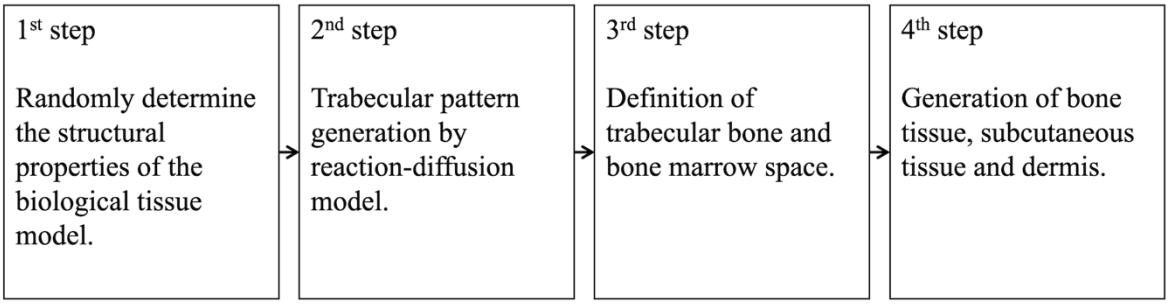


図 4.2　組織モデルの作成手順

ステップ 1，生体組織モデルの構造特性をランダムに決定した．生体組織モデルの構造的特性を表 4.1 に示す．真皮と皮下組織の厚さは，それぞれ 1.0 から 2.0 mm [96]，1.0 から 6.0 mm [95] の範囲で一様な確率の乱数を用いて決定した．これらは，前腕の軟組織厚さとしては，十分な大きさの範囲を仮定している．骨組織の構造特性は，Boutroy (2005) らによる女性の橈骨最遠位端 (UD radius)の測定値を用いた [83]．皮質骨厚（C.Th）と骨量（bone volume (BV) fraction, BV/tissue volume (TV)）は骨粗鬆症の重症度（骨粗鬆症，骨減少症，健常）で異なる．そこで，骨粗鬆症の重症度ごとに平均と分散が異なる正規分布を仮定し，乱数を用いて C.Th と BV/TV を定義した．C.Th と BV/TV の分布はピアソン相関係数 r = 0.54 であった．骨基質量（mBMD）は，完全に石灰化した骨を想定し，1.2 g/cm^3 と仮定した．

表 4.1 合成生体組織モデルの構造特性

	Thickness in z and x axis direction [mm]	BV/TV	mBMD [g/cm^3]	Ref
Dermis	1.0 - 2.0	-	-	Kozarova (2017)[96]
Subcutaneous	1.0 - 6.0	-	-	Hassager (1989)[95]
Cortical bone	Osteoporosis 0.487 ± 0.138, Osteopenia 0.571 ± 0.173, Normal 0.804 ± 0.149.	-	1.2	Boutroy (2005)[83]
Trabecular bone	17.15 minus the cortical bone thickness	Osteoporosis 8.5 ± 2.2, Osteopenia 10.3 ± 3.0, Normal 13.4 ± 2.8.	1.2	Boutroy (2005)[83]

The values showing the range (number - number) have a uniform distribution of probability, and the mean ± standard deviation has a normal distribution. The three states of cortical bone thickness and BV/TV are corresponding to their respective values.

ステップ 2，Alan Turing (1952) の反応拡散モデル [73] を用いて，骨梁パターンを生成した．骨梁パターンの生成方法は第 2 章で既に説明した．式 2.15 の連続支配方程式における反応項は，式 2.21 を用いた．活性化因子と抑制因子の拡散係数 d_u，d_v は，それぞれ 0.0002 ならびに 0.01 とした．また，活性化因子 u と抑制因子 v の初期濃度分布は，活性化因子，抑制因子ともに-0.5 から 0.5 の範囲の一様な乱数とした．グリッド空間のサイズは，ここでは 720×720×720 の 3 次元配列とした．1 グリッドの大きさは，$\frac{1}{64}$ とした．計算試行回数は 100 回とした．以上の条件で活性化因子濃度 u を算出した．

ステップ 3，式 2.23 を用い，ステップ 1 で決定された BV/TV から，骨梁と骨梁間隙の境界を分ける閾値 u_{th} を算出した．また，第 2 章と同様に，ボクセルサイズ lv を 24.5 μm とした．

ステップ 4，UD radius を想定し，骨組織，皮下組織，真皮を定義し，図 3.4 のような合成生体組織モデルを生成した．生体組織モデルは，骨軸方向を y 軸とした．まず，骨組織の大きさを決定した．骨軸方向に十分な大きさを確保するため，生成した海綿骨を y 軸方向にコピーし，接合した．UD radius の断面積から，一辺 17.15 mm の正方形を x-z 平面における外

骨表面（OBS）と定義した（図 4.3）．したがって，骨組織の大きさは，x-z 軸で 17.15 mm，y 軸で 35.28 mm となった．次に，OBS から海綿骨の中心に向かって C.Th の大きさの皮質骨を定義し，骨組織を生成した．骨組織の面積骨密度 aBMD は，式 3.19 を用いて決定した．最後に，図 4.3 に示すように，OBS から骨組織の外側に向かって皮下組織，真皮の順で生成した．以上の処理はすべて Python 3.8 でコード化した．

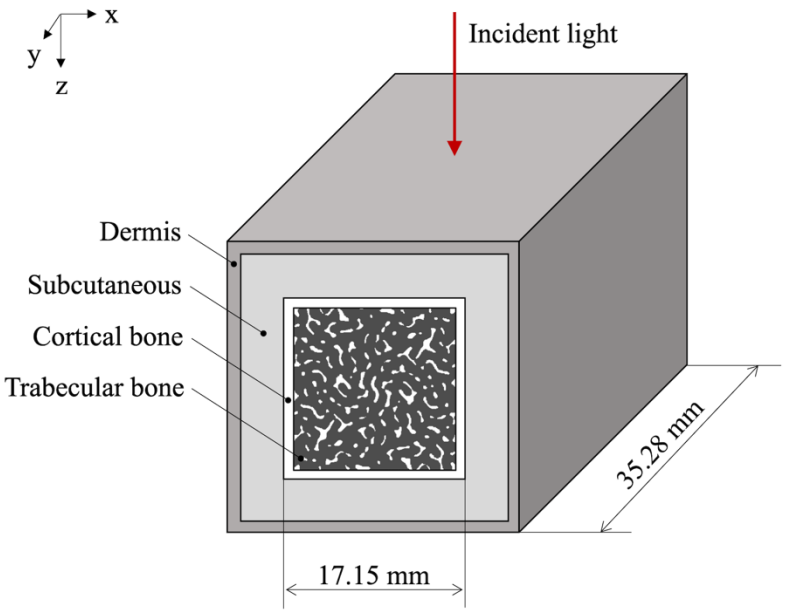


図 4.3 橈骨最遠位端を想定した合成生体組織モデルの外観

4.2.2　モンテカルロシミュレーション

モンテカルロシミュレーションでは，まず，合成生体組織モデルの光学特性がランダムに決定される．モデルの光学特性を表 4.2 に示す通りである．光学特性の範囲は，波長 850 nm の弾道光による測定を想定し，文献値を参考に決定した．軟部組織の吸収係数 μa，散乱係数 μ_s は，Simpson（1998）[94] らの測定値の範囲の一様乱数を用いて決定した．著者らが測定した真皮の光学特性は，黒人と白人のものを含み，真皮と表皮を合わせたものを仮定した．骨の μ_s は Ugryumova（2004）らの式 [18] から求めた．Ugryumova の式では mBMD が湿潤密度

であるため，μ_sは Williams（1996）[97]らのデータから mBMD を湿潤密度に変換した後に算出した．従って mBMD と μ_sの変換式は，次式のようになる．

$$\mu_s = 17.77mBMD - 0.74 \tag{4.1}$$

一般的な説明において骨髄の主な役割は造血である．従って，骨髄中には，赤色の増結細胞が含まれており，近赤外光をよく吸収する．しかしながら，橈骨の骨髄は 20〜30 歳を過ぎるとほとんどが脂肪細胞で構成される [98]．従って，骨髄の光学特性は脂肪細胞を主体とする皮下組織と同じ値とした．

表 4.2　合成生体組織モデルの光学特性

	μ_a [mm^{-1}]	μ_s [mm^{-1}]	g	n	Ref
Dermis	0.0063 - 0.0856	14.20 - 25.06	0.9	1.4	Simpson (1998)[94]
Subcutaneous	0.0049 - 0.0124	8.30 - 13.96	0.9	1.4	Simpson (1998)[94]
Bone	0.0237	20.58	0.9	1.55	μ_a and μ_s, Ugryumova (2004)[18]; n, Ascenzi (1959)[99].

μa, absorption coefficient; μs, scattering coefficient; g, anisotropy coefficient; n, refractive index.

VMC を用いて第 4.2.1 項で生成された合成生体組織モデルに対し光輸送シミュレーションを実施した．シミュレーションは，全光子数 N を 10^7 として行った．VMC は，構造的・光学的特性の異なる 1211 の組織モデルに適用された．各モデルにおいて，構造的・光学的特性は表 4.1 および表 4.2 に示す範囲でランダムに組み合わせられた．合成生体組織モデルからモデル外に飛び出した光子パケットは，パケットの発射方向に対して後方，前方，側方散乱に分類された．側方散乱については，x 軸の正方向のみを考慮した．3 方向に飛び出す光子パケットを後方散乱光（B），前方散乱光（F），側方散乱光（L）と定義した．

4.2.3　物理量のスコアリング

物理量のスコアリング方法の詳細については，第 3 章で説明した．

B_r と F_r では，空間分解のための半径方向のグリッドサイズ $\Delta r = 0.4$mm で，光源から半径 r = 10mm まで計算した．r については，実測を想定した場合，r＝0 での光量の正確な測定が困難であると考えられるため，半径方向の初期位置 r_0 を 0.5 mm とした．

L_z では，空間分解のためのグリッドサイズ $\Delta z = 0.1$ mm で，y 軸方向の幅 $\Delta y = 5$ mm で z = 20 mm までの範囲を計算した．

4.2.4　特徴量の生成

得られた光強度分布 B_r, F_r, L_z のシグナルをそれぞれ異なる加工を施し，特徴ベクトルが生成した．特徴量ベクトル B_f，F_f，L_f は，

$$B_f = [B_r, \ln mB_r, \ln vB_r] \tag{4.2}$$

$$F_f = [\ln mF_r, \ln vF_r] \tag{4.3}$$

$$L_f = [\ln L_z, \ln mL_z, \ln vL_z] \tag{4.4}$$

ここで，mB_r，mF_r，mL_r はそれぞれ B_r，F_r，L_r の平均値，vB_r，vF_r，vL_z は分散を表す．F_r は有効な分布を形成しないため，F_f では使用しなかった．また，L_z の信号はノイズが多いため，1 mm 間隔で移動平均を算出した．そして，L_z の長さを短くするために，2 mm 間隔の平均値を採用し，L_z を長さ 10 の配列とした．各特徴ベクトルの要素は，以下のように，データセットの平均値 0 標準偏差（SD）1 に正規化した．

$$\frac{B_{fi} - \mu_{bi}}{\sigma_{bi}} \rightarrow B_{fi}, i = 1,2,3, \dots, n_b \tag{4.5}$$

$$\frac{F_{fi} - \mu_{fi}}{\sigma_{fi}} \rightarrow F_{fi}, i = 1,2 \tag{4.6}$$

$$\frac{L_{fi} - \mu_{li}}{\sigma_{li}} \rightarrow L_{fi}, i = 1,2,3, \dots, n_l \tag{4.7}$$

ここで，μ と σ は，特徴ベクトルの各要素におけるデータセットの平均値と SD であり，n は要素の総数である．

4.2.5　骨密度の予測

生成した B_f, F_f, L_f から機械学習モデルを用いて aBMD を予測する．モンテカルロ法の計算，並びに B_f, F_f, L_f の生成，機械学習モデルによる aBMD の予測は全て Python 3.8 環境で行なった．

4.2.6　機械学習モジュール

ここでは，ラベル付けされた例から特徴ベクトル（Bf, Ff, Lf）と aBMD を関連付ける関数を推論するモジュールについて説明する．

本論では，リッジ回帰（RR），サポートベクターマシン（SVM），ランダムフォレスト（RF），勾配木ブースティング（GTB）の 4 種類の構造の異なる機械学習モジュールをテストした．機械学習モジュールの選択基準は，データの特殊性に対して十分な安定性と柔軟性を持ち，さらに過剰汎化を抑制する方策を持つことであった．選択された機械学習モジュールの原理について次に示す．

Ridge regressor (RR)

リッジ回帰器(RR)は L2 正則化を施した線形最小二乗法である．この理論は，1970 年に Hoerl と Kennard によって初めて紹介された [100,101]．リッジ回帰のモデル式は以下の通りである．

$$y = \sum_{i=0}^{m} w_i x_i \tag{4.8}$$

ここで，y は従属変数，x は独立変数，w は独立変数の重みである．次に，右辺と左辺の方程式の差を最小にするwを求める必要がある．この問題を解く最も簡単な方法は，最小二乗法である．データ空間において，式 4.8 は次式でベクトル空間として表すことができる．

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1, & x_{11} & \cdots & x_{m1} \\ & \vdots & \ddots & \vdots \\ 1, & x_{1n} & \cdots & x_{mn} \end{pmatrix} \begin{pmatrix} w_0 \\ \vdots \\ w_m \end{pmatrix} \tag{4.9}$$

$$\hat{\boldsymbol{y}} = X\boldsymbol{w} \tag{4.10}$$

最小二乗法は，次の式で表される損失関数 L の最小化問題を解くものである．

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{4.11}$$

式 4.11 は，ベクトルの内積として以下のように表される．

$$L = (\boldsymbol{y} - X\boldsymbol{w})^T (\boldsymbol{y} - X\boldsymbol{w}) \tag{4.12}$$

$$L = \boldsymbol{w}^T X^T X\boldsymbol{w} - 2\boldsymbol{y}^T X\boldsymbol{w} + y^T y \tag{4.13}$$

式 4.12 を微分して，一次導関数が 0 ベクトルになる w を求める．

$$\frac{\partial L}{\partial \boldsymbol{w}} = 2X^T X\boldsymbol{w} - 2X^T\boldsymbol{y} = \boldsymbol{0} \tag{4.14}$$

ここで，$X^T X$は正方行列であり，逆行列が存在するため，w は次のように表される．

$$w = (X^T X)^{-1} X^T \boldsymbol{y} \tag{4.15}$$

$X^T X$ の逆行列が存在しない場合，解を求めることができない．これを多重共線性問題という．最小二乗法には多重共線性以外にもオーバーフィッティングの問題がある．入力 x ベクトルが多い場合，重み w がデータに対して過剰に汎化されてしまうことがある．RR では，L2 正則化と呼ばれる二乗（L2 ノルム）ペナルティを重みに与えることで，フィッティング時のオーバーフィッティングを回避している．Ridge におけるコスト関数は以下のように表される．

$$L = (\boldsymbol{y} - X\boldsymbol{w})^T (\boldsymbol{y} - X\boldsymbol{w}) + \lambda \|w\|_2 \tag{4.16}$$

ここで，λ は正則化項の影響の度合いを定義するためのハイパーパラメータである．式 4.14 と同様に，式 4.16 を微分して 0 ベクトルを求めると，w は次式となる．

$$w = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y} \tag{4.17}$$

ここで，I は恒等式行列である．

本研究では，正則化項の係数は 10-5 から 10-1 の範囲で変化させ，テストを行った．Python 3.8 の scikit-learn 0.24.2 の Ridge モジュールを使用した．

Support vector machine (SVM)

サポートベクター シン (SVM) は分類と回帰のための一般的な機械学習ツールで，V. Vapnik (1992)らが提案した．SVM は，プシロン不感応帯を用いることでノイズの影響を受けにくい性質を持つ．また，SVM 回帰はカーネル関数に依存するため，ノンパラメトリックな手法と考えられており，非線形な関数にも導入可能である．SVM の説明については，V. Kecman (2005)と A. J. Smola (2004)の解説を参考にした [102].

SVM の回帰分析手法は，基本的にリッジ回帰と同様に線形回帰手法である．そのため，あるサンプルの目的変数の推定値は，式 4.10 と同様に次の様に表される．

$$f(X_i) = X_i \boldsymbol{w} \tag{4.18}$$

リッジ回帰と同様に，重み w を誤差関数に加えて，過学習を防止する．リッジ回帰と異なる点は，誤差に不感帯を設けることで，ノイズの影響を受けにくくしている．また，線形回帰モデルを高次元空間で写像することで，非線形な問題にも対応可能である．SVM の回帰では，次を最小化する凸最適化問題として定式化される．

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} h\big(y_i - f(X_i)\big) \tag{4.19}$$

ここで，y はサンプルのラベルであり，h は誤差関数である．h は，次の様に定義される．

$$h\big(y_i - f(X_i)\big) = \max(0, |y_i - f(X_i)| - \varepsilon) \tag{4.20}$$

つまり，式 4.19 には，全ての残差が ε より小さいという条件があり，次式のように表される．

$$|y_i - f(X_i)| \leq \varepsilon \tag{4.21}$$

全ての点についてこれらの制約を満たす関数 f(X)は存在しない可能性がある．実行不可能な制約に対処するため，各点にスラック変数 ξ_i，ξ^*_i を導入する．スラック変数は必要な条件を

満たしたまま，ξ_iとξ^*_iまで回帰誤差を許容する．スラック変数を含めると，目的関数は次のようになる．

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{4.22}$$

そして，式 4.21 には次の条件が適用される．

$$y_i \le f(X_i) + \varepsilon + \xi_i, \qquad y_i \le f(X_i) - \varepsilon - \xi_i^*, \qquad \xi_i \ge 0, \qquad \xi_i^* \ge 0 \tag{4.23}$$

ここで，C はボックス制約で，ε の範囲外にあるサンプルに課されるペナルティを制御する値であり，過度な一般化を防止するのに役立つ．

$\frac{\|\boldsymbol{w}\|^2}{2}$を最小化するとマージン（ε）の範囲内に侵入するサンプルが増加し，$C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$が増加する．これは双極問題と呼ばれており，この最適化問題を解くには，相反する 2 つの項のバランスをとりながら最小化する必要がある．この様な問題を解くには，一般的にはラグランジュ未定乗数法を用いる．ラグランジュ未定乗数法は制約つき最適化問題の代表的な手法である．目的関数 f(x)を n 個の不等式制約条件 g(x)i≤0, i=1,2,3,…n の条件にもとで最小化するときを考える．次の様にラグランジュ関数を定義する．

$$L(x,\alpha) = f(x) + \sum_{i=1}^{n}\alpha_i g_i(x) \tag{4.24}$$

ここで，x の微分が極小解であるならば，α_iが存在して，ラグランジュ関数について次の 4 つの条件が成り立つ．

$$\frac{\partial L(x,\alpha)}{\partial x} = 0 \tag{4.25}$$

$$\frac{\partial L(x,\alpha)}{\partial \alpha_i} = g_i(x) \le 0, \qquad (i = 1,2,\cdots,n) \tag{4.26}$$

$$\alpha_i g_i(x) = 0, \qquad (i = 1,2,\cdots,n) \tag{4.27}$$

$$\alpha_i \ge 0, \qquad (i = 1,2,\cdots,n) \tag{4.28}$$

極大の場合は，α_iに関する不等式が逆になる．次に，式 4.21 にラグランジュ未定乗数法を適用する．ここで，ラグランジュ定数は，$\alpha_i, \alpha_i^*, \beta_i, \beta_i^*$とした．

$$L(w,c,\xi_i,{\xi_i}^*,\alpha_i,{\alpha_i}^*,\beta_i,{\beta_i}^*)=\frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=1}^{n}(\xi_i+{\xi_i}^*)-\sum_{i=1}^{n}\left(\beta_i\xi_i+{\beta_i}^*{\xi_i}^*\right)$$
$$-\sum_{i=1}^{n}\alpha_i(\varepsilon+\xi_i+f(X_i)-y_i+b)-\sum_{i=1}^{n}{\alpha_i}^*(\varepsilon+{\xi_i}^*-f(X_i)+y_i+b) \tag{4.29}$$

上式を$w,c,\xi_i,{\xi_i}^*$で偏微分すると次の式が得られる．

$$w=\sum_{i=1}^{n}(\alpha_i-{\alpha_i}^*)\emptyset(X_i)^T \tag{4.30}$$

$$\sum_{i=1}^{n}(\alpha_i-{\alpha_i}^*)=0 \tag{4.31}$$

$$\alpha_i+\beta_i=C,\qquad (i=1,2,\cdots,n) \tag{4.32}$$

$$\alpha_i{}^*+\beta_i{}^*=C,\qquad (i=1,2,\cdots,n) \tag{4.33}$$

ここで，ϕは関数 f の写像である．これらを用いて，L を変形すると，次の 2 次計画問題となる．

$$maximize\left\{L(\alpha)=\begin{array}{c}\sum_{i=1}^{n}(\alpha_i-{\alpha_i}^*)y_i-\varepsilon\sum_{i=1}^{n}(\alpha_i+{\alpha_i}^*)\\ -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=i}^{n}(\alpha_i-{\alpha_i}^*)\left(\alpha_j-{\alpha_j}^*\right){X_i}^TX_j\end{array}\right\},$$
$$subject\ to\ \sum_{i=1}^{n}(\alpha_i-{\alpha_i}^*)=0\ and\ \alpha_i,{\alpha_i}^*\in[0,C] \tag{4.34}$$

この問題を解くために，SVM では，カーネル法が用いられる．線形回帰関数は，非線形の問題に対して回帰能力に限界がある．そこで，カーネル法では，元の空間より高次元に写像してデータの分離を試みる．高次元空間のデータは入力空間のデータ${x_i}^Tx_j$を関数$\emptyset(x)$に写像したものであるので，高次元空間の最適化問題は次式となる．

$$maximize\left\{L(\alpha)=\begin{array}{c}\sum_{i=1}^{n}(\alpha_i-{\alpha_i}^*)y_i-\varepsilon\sum_{i=1}^{n}(\alpha_i+{\alpha_i}^*)\\ -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=i}^{n}(\alpha_i-{\alpha_i}^*)\left(\alpha_j-{\alpha_j}^*\right)\emptyset(X)_i{}^T\emptyset(X)_j\end{array}\right\} \tag{4.35}$$

しかしながら，ここで特徴量空間が高次元になればなるほど，$\emptyset(X)_i{}^T\emptyset(X)_j$の計算量が増大する．この項を簡単に計算する方法がカーネルトリックと呼ばれる方法である．次の様にカーネル関数を定義する．

$$K(X_i, X_j) = \emptyset(X)_i{}^T \emptyset(x)_j \tag{4.36}$$

この，カーネル関数を用いることで，$\emptyset(x)$を直接計算する必要がなくなる．実際に問題を解く際は，次のような 3 つのカーネル関数がよく用いられる．

$$K(X_i, X_j) = X_i{}^T X_j \tag{4.37}$$

$$K(X_i, X_j) = exp\left\{-\frac{\|X_i - X_j\|}{2\sigma^2}\right\} = \exp\left(-\gamma\|X_i - X_j\|^2\right) \tag{4.38}$$

$$K(X_i, X_j) = \left(X_i{}^T X_j + c\right)^d \tag{4.39}$$

ここで，γ, dは，ハイパーパラメーターであり，それぞれの関数は，線形カーネル，ガウシアンカーネル，多項式カーネルと呼ばれる．最後に，式4.30の 2 次計画問題を解くと，次式が導かれる．

$$\boldsymbol{w} = \sum_{i=1}^{n}(\alpha_i - \alpha_i{}^*)\emptyset(X_i)^T, \qquad Thus\ f(X_j) = \sum_{i=1}^{n}(\alpha_i - \alpha_i{}^*)K(X_i, X_j) + b \tag{4.40}$$

これは，サポートベクター展開と呼ばれており，w はトレーニングパターン X_iの線形結合として完全に記述できる．従って，SVM を用いた回帰では，ハイパーパラメーターとして，C と ε，カーネル関数でガウシアンカーネルを使用する場合は γ の決定が必要である．

本研究では，カーネルには，ガウシアンカーネルを選択し，カーネル係数 γ は，2e-3 から 1 の範囲で変化させテストした．また，ソフトマージン C とチューブ ε も変化させ，予測値に対し最も高い決定係数 r^2 が得られる組み合わせを採用した．SVM 回帰関数は，Python 3.8 の scikit-learn 0.24.2 の SVR モジュールを用いた．

Random Forest (RR)

ランダムフォレストは，Breiman (2001)によって考案された [103]．RF は決定木を弱学習器とするアンサンブル学習アルゴリズムである．そのため，本項では，まず決定木(Decision tree)を説明し，次に RF の説明をする．決定木の説明は，Quinlan (1986)の説明を参考にした [104,105]．

決定木は，対となる質問を繰り返し行うことにより最終的な判断を実現させる方法である．その構造は，木（Tree）の枝分かれのように見えるため（図 4.4）決定木と呼ばれる．機械

学習では，この枝分かれを決定する方法として，ID3, C4.5, CART が有名である．本項では，まず，ID3 について説明する．

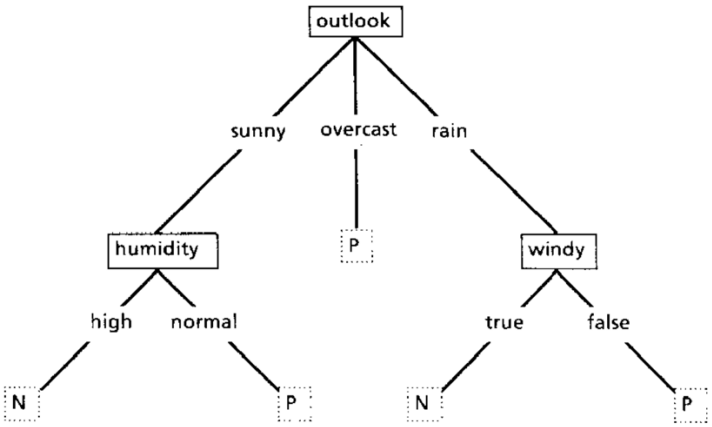


図 4.4　決定木の概略図 [105]

ID3 は，J.R. Quinlan によって 1979 年に考案された．図 4.4 に示すように，決定木の葉（leaf）は，クラス名であり，他のノードは，考えられる結果ごとに分岐する属性ベースのテストを表す．オブジェクトを分類するために，tree の根幹から開始し，テストを評価して，結果に適した枝分かれを取得する．このプロセスは，leaf が検出されるまで続く．leaf が検出されると，オブジェクトは leaf によって指定されたクラスに属することが決定される．ID3 は，tree の枝別れを前後の情報エントロピー差で定義する．決定木の枝分かれ前後のエントロピー差 I_G は，情報ゲインと呼ばれ，次の式で表される．

$$I_G(D_p, f) = I_H(D_p) - \sum_{j=1}^{C} \frac{N_j}{N_P} I_H(D_j) \tag{4.41}$$

ここで，左辺の第 1 項は，親ノードの情報エントロピーであり，第 2 項は，子ノードのエントロピーである．加えて，N_j は子ノード内のサンプル数，N_p は親ノード内のサンプル数を示す．情報エントロピーは，ノード内の情報の不純度を示す．つまり，情報エントロピー差のである I_G が大きければ，親ノードと個ノード間で情報の乱雑性が大きく減少したことを示す．従って，ID3 は，この I_G の最大化させる条件を探索するアルゴリズムである．

決定木のノード内の情報の不純度は，情報エントロピー I_H として次の式のように表される．

$$I_H(t) = -\sum_{i=1}^{n} P(i|t)\log_2 P(i|t), \qquad P(i|t) = \frac{n_i}{n} \tag{4.42}$$

ここで，n_i はクラス i に属するサンプル数を示す．では，情報の不純度はどのように情報エントロピーで説明されるのだろうか．これは，情報量（または自己エントロピー）で説明される．情報量は，事象の起こりにくさを表す尺度であり，ありふれた事象は情報を持っておらず，珍しい事象は情報を持っていると考える．ここに，情報量の性質を整理する．(1) 情報量 f(p)は，単調減少する関数である;起きる確率が大きいほど情報量は小さい．(2) f(p)は連続する．(3) f(p,q) = f(p) + f(q)の性質を示す；これは，p と q が独立なとき，「事象 p と q を同時に観測したときに得る情報量」と「事象 p を観測して得た情報量と事象 q を観測して得た情報量の和」は等しいことを示す．(4)確率 1/2 の自称の情報量は 1bit である．これら(1)から(4)の条件を満たす関数を考える．(2)と(3)より，g(x) = f(α^x)と置くと次式が得られる．

$$g(x+y) = f(\alpha^{x+y}) = f(a^x) + f(a^y) = g(x) + g(y) \tag{4.43}$$

コーシーの函数方程式より

$$g(x) = bx \tag{4.44}$$

$$f(x) = g(\log_a x) = b\log_a x = C\log x \tag{4.45}$$

以上の条件より，情報量は次式で定義される．

$$f(x) = -log_2 x \tag{4.46}$$

情報エントロピーは，各事象の選択情報量-logP の期待値である．

C4.5 は，ID3 の後継アルゴリズムである．ID 3 では現実の問題を扱うのは困難であり，C4.5 では主に次のような改良がなされた．

1, 特徴量がカテゴリーでなければならないという制限を取り除いた．これは，連続変数を離散的な間隔のセットに分割する離散属性を同的に定義することで実現した．

2, オーバーフィッティングを防止する機能が追加された．この機能は，プルーニング(Pruning, 刈り込み)と呼ばれる．Pruning は，ルールの前提条件を削除しない方が精度向上する場合に，その前提条件を削除することで行われる．

3, Missing value への対処が可能.

C4.5では，属性が欠損している場合，欠損した属性値は，ゲインとエントロピーの計算に使われない．

CART は，C4.5 と非常に似ているが，回帰をサポートし，ルールセットを計算しないという点で異なる．CART は，各ノードで最大の情報ゲインをもたらす機能と閾値を使用して，バイナリーツリーを構築する．また，CART では，情報エントロピーの他に，ジニ不純度と呼ばれる指標も利用可能である．ジニ不純度 I_G を次に示す．

$$I_G(t) = 1 - \sum_{i=1}^{C} P(i|t)^2, \qquad P(i|t) = \frac{n_i}{n} \tag{4.47}$$

ジニ不純度では不純度が大きくなると I_G が 1 に漸近する．我々が本研究で用いる scikit-learn では，この CART が採用されている．次に RF について説明する．

決定木は特徴のスケーリングやその他様々な変換に対して不変であり，無関係な特徴の混入に対してもロバストである．その結果，決定木は様々な学習課題で利用されているが欠点もある．特に，非常に深く成長した木は，学習集合に過剰にフィットし，不規則なパターンを学習する傾向がある．つまり，バイアスは低いが，分散が非常に大きくなる．RF は，図 4.5 に示すように，分散を減らす目的で，同じ訓練データセットの異なる部分で訓練した複数の深い決定木を平均化する．この平均化はバイアスのわずかな増加と解釈可能な損失を犠牲にするが，一般に最終モデルの性能を大幅に向上させる．RF アルゴリズムは分散を減らすために，樹形学習器にバギングという一般的な方法を適用する．応答 y を持つ訓練集合 X が与えられたとき，バギングを繰り返し（B 回），訓練集合の置換付きランダムサンプルを選択し，これらのサンプルに木を当てはめる．

For b = 1, 2, …, B:

1. Sample n training example from X, **y**; call X_b, $\mathbf{y}_b$.
2. Train a regression or classification tree f_b on X_b $\mathbf{y}_b$.

学習後，未知のサンプル X_b に対する予測は，X_b に対する個々の回帰木の予測値をすべて平均化することで行うことができる．

$$f(X) = \frac{1}{B}\sum_{b=1}^{N} f_b(X_b) \tag{4.48}$$

ここで，分類木の場合，多数決が行われる．この手順はブートストラップ集計としても知られ，バイアスを増加させることなくモデルの分布を減少させる．つまり，1 本の木の予測はその訓練セットのノイズに非常に敏感であるが，多くの木の平均は，木が相関していない限り，敏感ではない．ブートストラップサンプリングは，異なる訓練セットを表示することによって，木の相関を装飾する方法である．

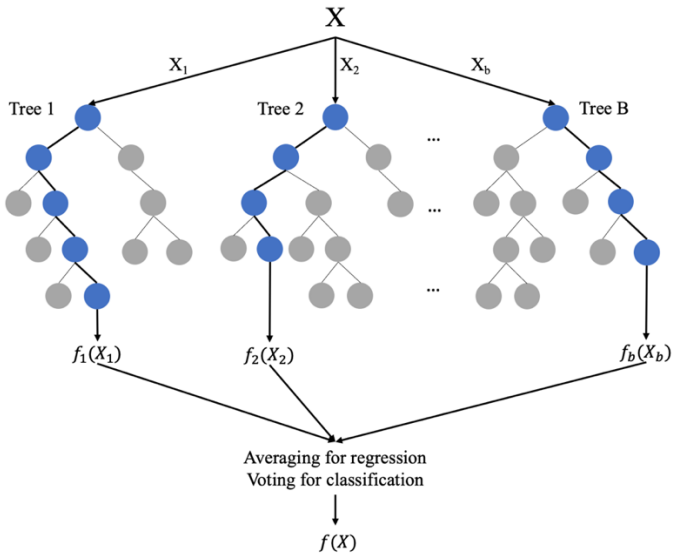


図 4.5　ランダムフォレスト回帰の概略図.

Breiman (2001)[103] を引用して，RF の理論的背景を説明する．分類器のアンサンブル $h_1(x)$, …, $h_k(x)$が与えられ，ベクトル X, **y** の分布からランダムに学習集合を引いたとき，マージン関数 *mg(X, **y**)* を以下のように定義する．

$$mg(X, \boldsymbol{y}) = av_k I(h_k) - \max_{j \neq \boldsymbol{y}} av_k I(h_k(X) = j) \tag{4.49}$$

ここで，$I(\cdot)$は指標関数である．マージンは，X, **y** における右クラスの平均投票数が，他のクラスの平均投票数をどの程度上回っているかを測定するものである．マージンが大きいほど，分類に対する信頼性が高い．汎化誤差は次式で与えられる．

$$PE^* = P_{X,\boldsymbol{y}}(mg(X, \boldsymbol{y}) < 0) \tag{4.50}$$

ここで，添え字の X, y は X, y 空間での確率であることを示す．RF では，$h_k(X) = h(X, \Theta_k)$ である．木の数が多い場合，大数の強法則と木構造から次のようになる．

定理：木の数が増えると，ほぼ確実にすべての配列$\theta_1 \ldots PE*$について，次式のように収束する．

$$P_{X,y}\left(P_\Theta(h(X,\Theta)=\boldsymbol{y})-\max_{j\neq \boldsymbol{y}} P_\Theta(h(X,\Theta)=j)<0\right) \tag{4.51}$$

この結果は，RF が木を増やしても過大汎化せず，汎化誤差の限界値を出すことを説明するものである．

RF では，次の 3 つのパラメータを変化させて調節した[106]; アンサンブル内の決定木の数，ノードがリーフとみなされるための最小サンプル数，最適なノード分割を計算する際に考慮する機能の数．RF は，python の scikit-learn 0.24.2 に含まれるものを用いた．

Gradient Tree Boosting (GTB)

Gradient Boosting は，回帰および分類問題の機械学習手法であり，弱学習器のアンサンブル形式の予測モデルを生成する．そして，弱学習器として決定木を使用したものを特に Gradient Tree Boosting （GTB）と呼び，通常は RF よりも優れている．他のブースティング手法と同様に段階的にモデルを構築し，任意の微分可能損失関数の最適化を可能とすることでモデルを一般化する．勾配ブースティングのアイデアは，ブースティングが適切なコスト関数最適化アルゴリズムとして解釈できるという Breiman の観察に由来している．その後，J.H. Friedman によって回帰勾配ブースティングアルゴリズムが開発された[107,108]．

Friedman が考案した GTB のアルゴリズムを図 4.6 に示す．GTB のアイデアは，残差を修正し最小化することであった．イメージとしては，まず弱い木 1 で大雑把な学習を行い，次に異なる木 2 で木 1 から得られた予測値とターゲットとの残差を学習する．この様に残差を学習することで，木 1 並びに木 2 から得られた予測値の合計は，ターゲットの値に近づき，その合計値の残差は，木 1 の予測から得られる残差よりも小さくなるはずである．このような操作を繰り返すことで，GTB は残差を最小化させてゆく．このアイデアを定式化すると次のようになる．まず，あるトレーニングサンプル$\{y_i, X_i\}_1^N$ を (y, X) としたとき， モデル F_1 をデータにフィットさせ，次に別のモデル h_1 を残差にフィットさせる．そして，そこから新たなモデル F_2 とつくることを考えると次の様になる．

$$F_1(X)=y, \qquad h_1(X)=\ y-F_1(X), \qquad F_2(X)=F_1(X)+h_1(X) \tag{4.52}$$

前のモデルの残差を修正するモデルをさらに挿入することで，このアイデアを次の様に一般化できる．

$$F(X) = F_1(X) \mapsto F_2(X) = F_1(X) + h_1(X) \cdots \mapsto F_M(X) = F_{M-1}(X) + h_{M-1}(X) \tag{4.53}$$

ここで，$F_1(X)$は y にフィットさせた最初のモデルである．この作業の最初に $F_1(X)$をフィットさせているので，各ステップでやるべきことは，h フィットさせることである．言い換えると，$h_m(X) = y - F_m(X)$となる h_mを見つける．次にこのアイディアを実装に近づけて一般化する．まず単一の予測値でモデル F_0 を初期化する．ここでのタスクは損失関数 L(y,γ)の最小化することである．この誤差関数が二乗の場合，F_0 は訓練データの平均値で初期化され，次のようになる．

$$F_0(X) = \arg\min_{\gamma} \sum_{i=1}^{n} \psi(y_i, \gamma) = \arg\min_{\gamma} \sum_{i=1}^{n} (\gamma - y_i)^2 = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{4.54}$$

ここで，損失関数 L は，図 4.6 のψであり，式 4.54 は 1 行目に当たる．つまり，ここから後続の F_m を再起的に求めていく．

$$F_m(X) = F_{m-1}(X) + h_{m-1}(X; p_m), \qquad for\ m \geq 0 \tag{4.55}$$

ここで，p_m はモデル h のパラメーターである．ハイパーパラメーターm は，使用するモジュールによって異なるが，一般的には交差検定で最適値が求められる．式4.54 は二乗誤差を用いたが，誤差を最小化には，勾配降下法を使用捨こともできる．勾配降下法を用いることで，任意の損失関数でパラメータを最適化可能である．

Algorithm 1: Gradient_TreeBoost

1 $F_0(\mathbf{x}) = \arg\min_{\gamma} \sum_{i-1}^{N} \Psi(y_i, \gamma)$.
2 For $m = 1$ to M do:
3 $\quad \tilde{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \ i = 1, N$
4 $\quad \{R_{lm}\}_1^L = L - \text{terminal node } tree(\{\tilde{y}_{im}, \mathbf{x}_i\}_1^N)$
5 $\quad \gamma_{lm} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{lm}} \Psi(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
6 $\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot \gamma_{lm} 1(\mathbf{x} \in R_{lm})$
7 endFor.

図 4.6　Friedman (2002) による gradient tree boosting のアルゴリズム [108] .

勾配降下の説明については，Friedman (2001)[107] の説明を引用する．パラメータ化されたモデル F(X; P)は，関数最適化問題を 1 つのパラメタ最適化に変更する．

$$\begin{aligned} P^* &= \arg\min_P \Phi(P) \\ \Phi(P) &= E_{y,x}(y, F(X;P)) \\ F^*(X) &= F(X;P) \end{aligned} \tag{4.56}$$

ほとんどの F(X; P)と L については，解くために数値最適化手法を適用する必要がある．これは多くの場合, パラメータに対する解を，式 4.56 で表現することになる．

$$P^* = \sum_{m=0}^{M} p_m \tag{4.57}$$

ここで，p_0 は初期値，$\{p_m\}_1^M$は連続した増分であり，それぞれ前のステップのシーケンスに基づく．勾配降下法もしくは急降下法は，数値最小化法の中で最も単純なものの一つである．これは，式 4.57 の増分を次のように定義する．まず，現在の勾配 g_m が計算される．

$$g_m = \{g_{jm}\} = \left\{ \left[\frac{\partial L(P)}{\partial P_j} \right]_{P = P_{m-1}} \right\}$$

$$P_{m-1} = \sum_{i=0}^{m-1} p_i.$$

$$P_m = -\rho_m g_m$$

$$\rho_m = \arg\min_\rho \Phi(P_{m-1} - \rho g_m). \tag{4.58}$$

負の勾配-gm は最も急な勾配方向を定義する．

勾配降下のアルゴリズムへの応用を説明する．出発点は式 4.54 で示される $F_0(X)$である．M＝1 回目において，$F_0(X)$における損失関数 L の勾配は，以下のように擬似残差$\tilde{y}_{im}$として計算される．

$$\tilde{y}_{im} = -\left[\frac{\partial L\big(y_i, F(X_i)\big)}{\partial F(X_i)} \right]_{F(X) = F_{m-1}(X)} \quad for\ i = 1, \dots, n \tag{4.59}$$

そして，h(X; pm)が与えられると，係数 β_m の最適値が決定される．

$$\beta_m = \arg\min_\beta \sum_{i=1}^{n} L\big(y_i, F_{m-1}(X_i) + \beta h(X_i; p_m)\big). \tag{4.60}$$

GTB はこのアプローチを，ベース学習器 h(X; p)が K 個のターミナル・ノードの回帰木である場合に特化したものである．各反復 m において，回帰木は X 空間を K 個の不連続な領域 $\{R_{km}\}_{k=1}^{K}$ に分割し，それぞれの領域で別々の定数値を予測する．

$$h\left(X;\{R_{km}\}_{k=1}^{K}\right)=\sum_{k=1}^{K}\bar{y}_{km}1(X\in R_{km}) \tag{4.61}$$

ここで，$\bar{y}_{km}=mean_{X_i\in R_{km}}(\tilde{y}_{im})$は各領域 R_{km} における式 4.59 の平均値である．回帰木を用いると，式 4.60 を各 R_{km} で個別に解くことができる．式 4.60 で示される木は，各領域 R_{km} で一定の値$\bar{y}_{km}$を予測するので，式 4.60 の解は，損失関数 L に基づく単純な位置推定に帰着する．

$$\gamma_{km}=\arg\min_{\gamma}\sum_{X_i\in R_{km}}L(y_i,F_{m-1}(X_i)+\gamma) \tag{4.62}$$

ここで，γ_{km} はモデルを更新する際のステップ倍率を意味する．また近似値 $F_{m-1}(X)$は，対応する各領域で個別に更新される．

$$F_m(X)=F_{m-1}(X)+\nu\cdot\gamma_{km}1(X\in R_{km}) \tag{4.63}$$

ここで ν は収縮パラメータであり，0 < ν ≦ 1 であれば学習率を制御することができる．これにより，決定木の一般化ブースティングでは，図 4.6 に示すようなアルゴリズムが導かれる．

本研究では，GTB モジュールとして，XGboost 1.4.2 を Python3.8 で使用した．GTB では，次の 3 つのパラメータを変化させた; アンサンブル内の決定木の数，オーバーフィッティングを防ぐために更新時に使用されるステップサイズの縮退，ツリーの最大深度，子に必要な最小のインスタンスウェイトの合計，トレーニングインスタンスのサブサンプル比率，各ツリーを構築する際のカラムのサブサンプル比率.

4.3 検証方法

各機械学習モジュールに最適な構造とパラメータを選択するため，全データの 80%をトレーニングおよび検証用データセット（trDataset）として使用した．残りの 20%のデータセッ

ト(tsDataset)はテストと比較のために使用された．各アルゴリズムの構造とパラメータセットを変更し，trDataset においてグリッドサーチによる 10 交差検定を行い，最も性能の良いものを選択した．10 交差検定（図 4.7）では，データセットの 90%をランダムに選択して学習に使用し，残りを検証に使用した．これはデータセットを回転させながら 10 回行った．また，クロスバリデーションにより，機械学習技術の安定性を確認した．最適な構成が得られた後，tsDataset を用いて性能評価を行った．性能評価の指標として決定係数(r^2)を用いた．

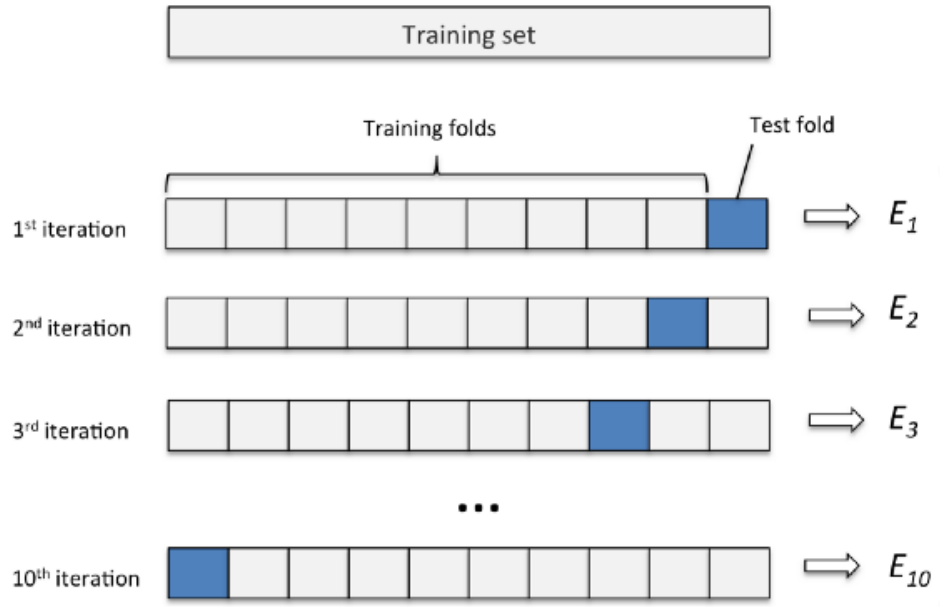


図 4.7　10 分割交差検定

4.4　結果

4.4.1　アルゴリズムごとの予測精度の比較

tsDataset は元データセットの 20 % (242 個)をランダムに抽出したもので，機械学習アルゴリズムの選択基準は tsDataset の決定係数 r^2 値であった．残りの 80 % (969) を含む trDataset はアルゴリズムの学習に使用された．機械学習アルゴリズムは trDataset において，10 分割交

差検定で最高の性能を達成するようにチューニングされた．最適なパラメータと構造のセットが見つかった後，機械学習アルゴリズムは trDataset で再トレーニングされ，tsDataset を用いて性能がテストされた．図 4.8 は，各アルゴリズムの決定係数の比較を示している．SVM 回帰は最も高い決定係数を示した（$r^2 = 0.757$）．他の機械学習アルゴリズムの決定係数は，RR で $r^2 = 0.572$，GTB で $r^2 = 0.451$，RF で $r^2 = 0.252$ であった．未知データに対するアルゴリズムの安定性を判断するために，trDataset における 10 分割交差検定間の r^2 の標準偏差を計算した．SVM の 10 分割交差検定間における決定係数の標準偏差は 0.056 であり，十分許容できる値であった．

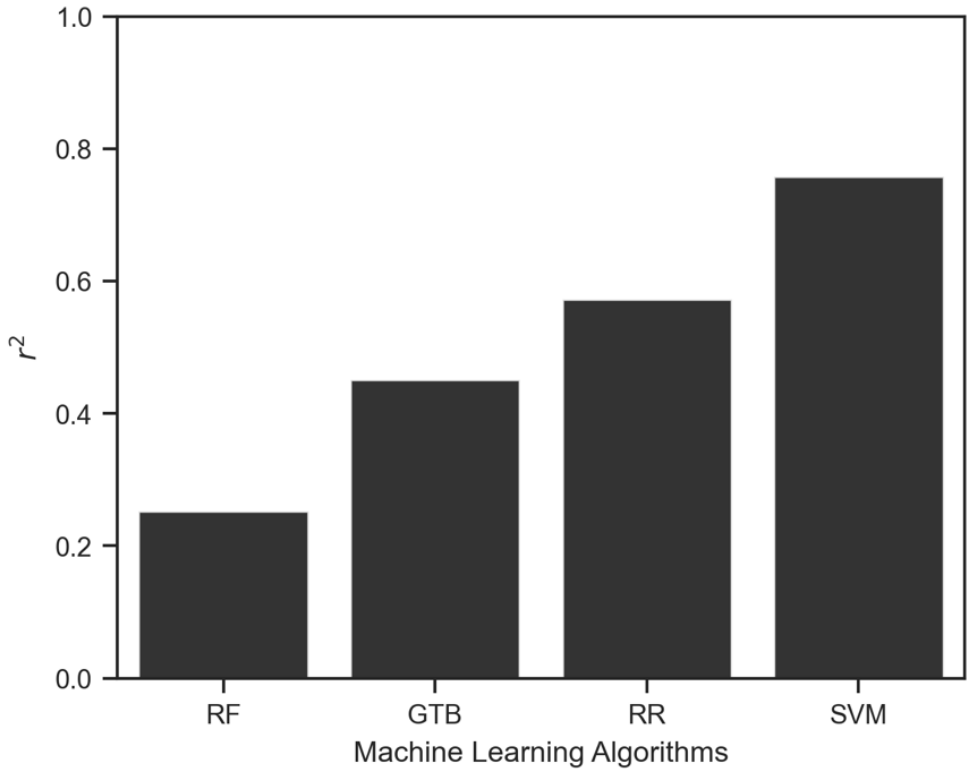


図 4.8　機械学習アルゴリズムの違いによる aBMD 予測性能の比較．SVM, Support vector machine, RR, Ridge regressor, GTB, gradient tree boosting, RF, random forest.

4.4.2　特徴量の組み合わせによる予測精度の比較

SVM を用いた aBMD の予測では，B_f，L_f，F_fのすべての組み合わせについて，tsDataset 上の決定係数 r^2 が計算された（図 4.9）．この実験の目的は，特徴ベクトルとその組み合わせがどの程度 aBMD に関係するかを調べることである．パラメータは trDataset で 10 分割交差検定を行い，すべての特徴ベクトルの組み合わせについてチューニングを行った．すべての特徴ベクトルを組み合わせた予測は，最も高い決定係数を示した．逆に，B_f，L_f，F_f 単独からの aBMD の予測は，最も低い決定係数を示した（$r^2 \leq 0.302$）．これらの結果から，1 つの特徴ベクトルでは aBMD の予測は困難であるが，特徴ベクトルを組み合わせることで精度の高い予測が可能になることが示された．

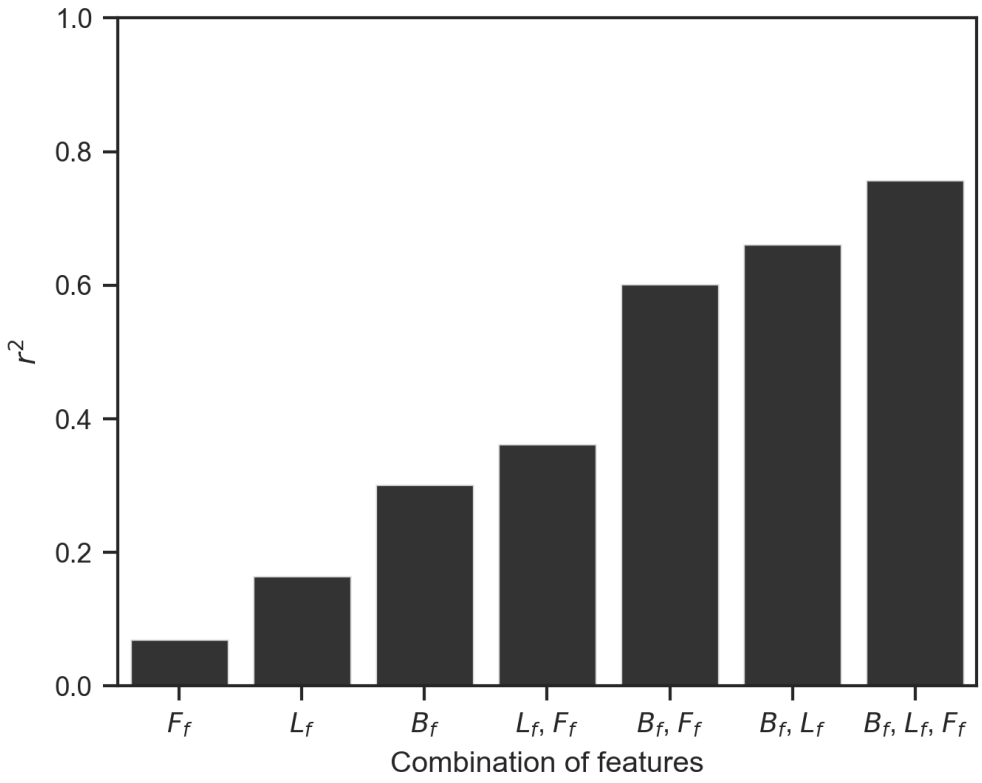


図 4.9　特徴ベクトルの組み合わせによる SVM 回帰の決定係数（r^2）の違い．B_f：後方散乱光からの特徴ベクトル，L_f：側方散乱光からの特徴ベクトル，F_f：前方散乱光からの特徴ベクトル．

4.4.3　最終結果

最終的に，1211 件の全データに対して，SVM 方式でシステムの性能を検証した．SVM は aBMD の予測に最も高い r^2 値を与えたため選択された．性能は，全データセットに対して 10 分割交差検定を用いて評価された．aBMD の予測値と基準値の関係を図 4.10 に示す．予測値と参照値の aBMD の線形回帰は r^2 値 0.760 をもたらし，妥当な一致を示した．図 4.11 に Bland-Altman（BA）プロットを示す．BA プロットは，2 つの測定方法の一致と系統誤差を確認するために用いられる方法である [109]．BA プロットは，わずかな比例バイアスを伴う 0.22 という中程度の相関係数 r を示す．この比例的な偏りは，実際には問題にならないかもしれない．予測値と基準値の平均差は 0.00 であり，固定バイアスはなかった．予測値の一致限界 (limit of agreement) は ± 0.124 g/cm^2 であった．これらの結果から，軟部組織の厚みや光学的性質にばらつきがあっても，この方法を用いることで高い精度で BMD を予測できることが示唆された．

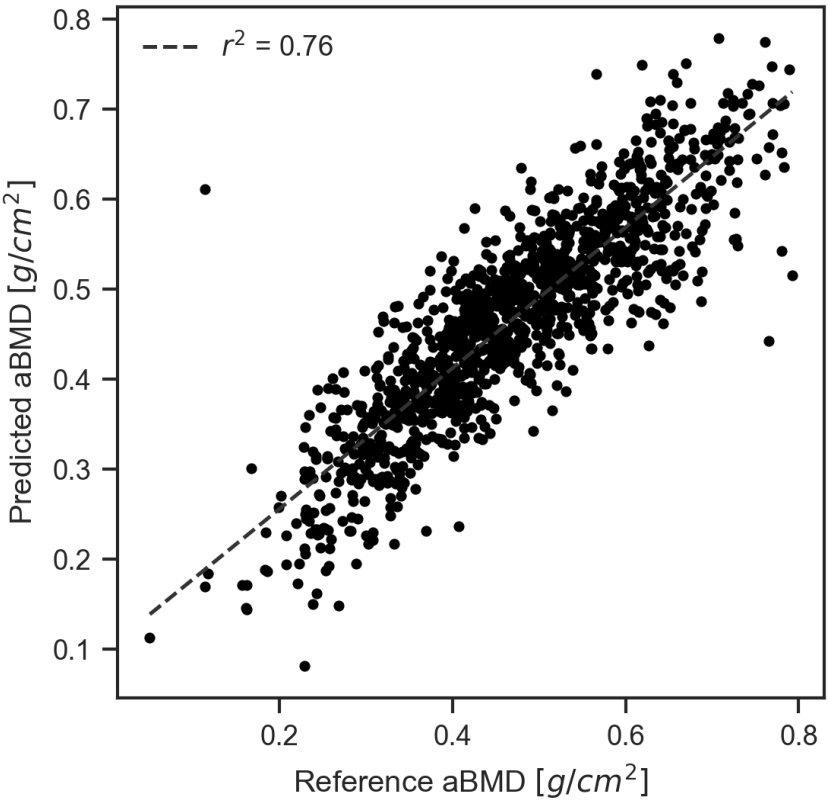


図 4.10　予測値と参照値の aBMD の関係

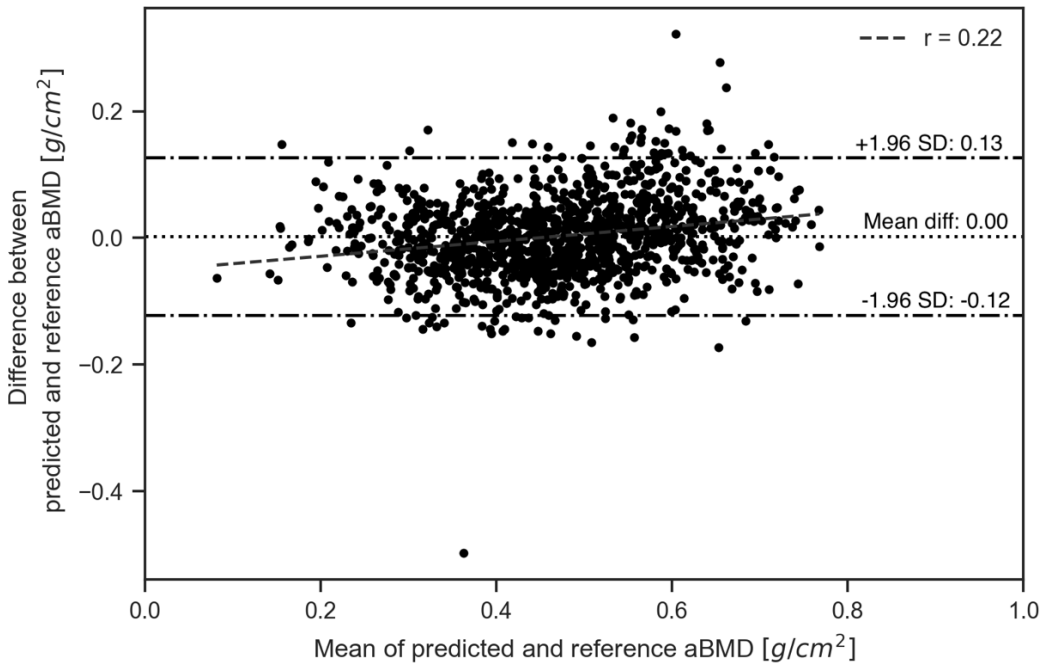


図 4.11　予測値と基準値 aBMD の Bland-Altman プロット

4.4.4　円柱モデルへの適用

本論は全編を通して，立方体のモデルを橈骨最遠位端として採用している．しかしながら，実際の橈骨最遠位端は，正確な立方体ではなく，円柱のような丸みも含んだ複雑な形状をしている．従って本項では，図4.12に示すような円柱モデルを生成し，立方体モデルと遜色ない骨密度の予測精度が得られるのか確認する．

円柱モデルのデータセットは，立方体モデルと同様の方法で生成された．円柱モデルでは，外骨表面 OBS までの半径距離 r_{obs} は，9.14 mm とした．ここで r_{obs} は，橈骨最遠位端における骨面積 [83] から算出された．円柱モデルの aBMD は次に示す式で導出した．

$$aBMD = mBMD[{r_{obs}}^2 - (1 - BV/TV)(r_{obs} - C.Th)^2]\,\pi \qquad (4.64)$$

データセットは，1000 パターンの円柱モデルに対して VMC によるシミュレーションで構成された．

図 4.11 と同様に，SVM が機械学習モジュールとして選択された．性能は，全データセットに対して 10 分割交差検定を用いて評価された．円柱モデルにおける aBMD の予測値と基

準値の関係を図 4.13 に示す．予測値と参照値の aBMD の線形回帰は r^2 値 0.626 をもたらし，妥当な一致を示した．この結果は，四角柱モデルと比較すると決定係数は 0.134 低い．しかしながら，この結果はモデルが円柱状であったとしても，多方向光学式骨密度計測法は，BMD の高い予測精度を有することを示唆する結果である．

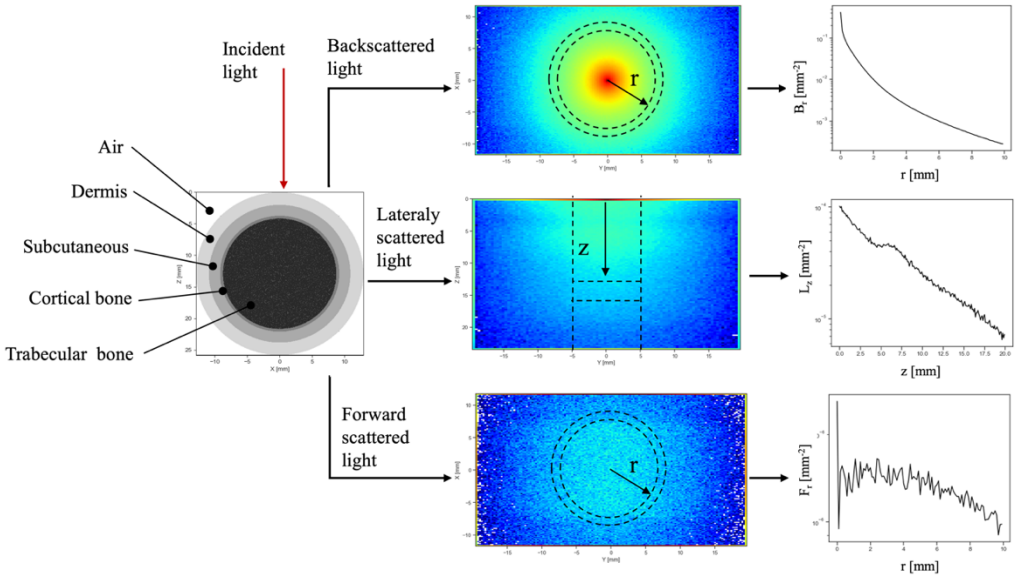


図 4.12　円柱モデルにおけるシミュレーション手順

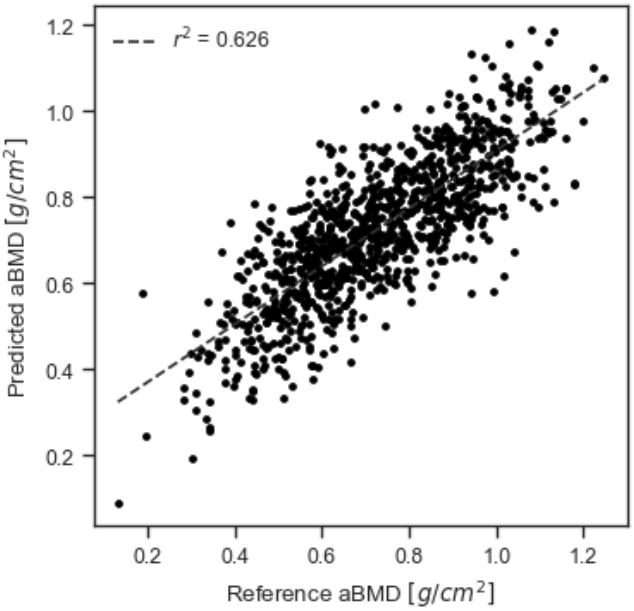


図 4.13　円柱モデルにおける予測値と参照値の aBMD の関係

4.5 考察

本研究の目的は，軟部組織の構造的・光学的特性の個人差に影響されない光学的骨密度測定法を開発し，その実現可能性を検証することである．提案手法では，モンテカルロ法によりシミュレーションした 3 方向（光照射方向に対して後方，前方，側方）の空間分解された拡散光を取得し，これらのデータから機械学習法により BMD を予測することとした．クロスバリデーションの結果，提案手法はシミュレーションにおいて高精度かつ低誤差で BMD を予測できることが実証された．この結果は，提案手法が軟組織の変動に頑健な光学的骨密度測定に適用できることを示唆している．

本研究で合成した生体組織モデルは，生体が示し得る範囲に基づいてランダムに構築されているため，モンテカルロシミュレーションで得られたデータは，十分な分散を持つ母集団を反映しているものと考えられる．軟組織の物性範囲は，測定値に基づいてランダムに決定した．特に真皮の光学特性は有色人種から白人まで広い範囲をカバーしている [94]．骨組織では，海綿骨に BMD 値を代表する単一の散乱係数を割り当てることが困難であるため，反応拡散モデルにより海綿骨パターンを生成した．Ugryumova の式 [18] では，海綿骨の BMD の範囲に対して負の散乱係数が得られている．また，Pifferi らは，TSR を使用した踵骨の測定において，散乱係数の年齢による変化がないことを発見した [21]．全体として，海綿体パターンの独特で不規則な形状は，均質な媒体と比較して異なる光散乱プロセスをもたらす可能性がある．本研究で生成した海綿状パターンは，外観，BV/TV，および定量的な幾何学構造の点で実骨のものと類似していた．さらに，BMD は重度の骨粗鬆症から健常者の範囲までランダムに調整された．さらに，モンテカルロ法は組織内の光輸送をモデリングするためのゴールドスタンダードである [71]．したがって，本研究でシミュレーションした拡散光は，十分な分散のある集団に対して生体組織の構造および光学特性を表していると考えることができる．

図 4.9 に示した結果は，Chung らの報告と一見矛盾しているように見える [16]．著者らは，波長 850nm の光を用いた超遠心半径の測定において，透過光強度と aBMD が強く相関していることを実証した．しかし，我々の結果では，透過光のみから推定される aBMD は高い決定係数を示さない．この矛盾は，母集団の分散に起因していると思われる．Chung らは，10 名という限られたサンプルサイズに着目した．透過光のみによる BMD 測定は，おそらく軟

部組織組成のばらつきが小さい集団に限られる．それにもかかわらず，F_f は，R_f，L_f と組み合わせた場合，決定係数を明らかに増加させた．この結果は，異なる方向から測定した光を組み合わせることで，個体差のある軟部組織からの誤差を低減できることを示唆している．

BMD と複数方向から観測される空間分解された拡散光との間には，非線形な関係が存在する可能性がある．図 4.8 に示すように，SVM は RR よりも高い予測性能を示した．RR は線形重回帰に基づくアルゴリズムである [100,101]，一方 SVM は RBF などの非線形カーネル関数を用いたマッピングにより非線形データに適用できるアルゴリズムである [102]，つまり aBMD の RR と SVM 予測の決定係数差は非線形性への対応力の差によるものと思われる．GTB や RF も非線形なアルゴリズムであるが，決定木ベースのアルゴリズムであるため，光源からの距離によって連続的な変化を示す拡散光への適用には適さないと思われる．したがって，我々のデータから BMD を予測する場合，SVM のような非線形データ用のアルゴリズムが必要と考えられる．

図 4.13 に示すように、骨形状が円柱状でも BMD の高い予測精度を示すことがわかる。本研究は、主に四角柱形状の組織モデルを中心に議論しているが、形状が変わっても、対称な形状であれば、同様の結果が得られることを示唆している。今後、非対称や複雑形状を持つモデルを対象にしたシミュレーションを行う必要がある。

モンテカルロシミュレーションのデータから SVM を用いて予測した aBMD は，決定係数が高く，誤差も少なかった（図 4.10，4.11）．このように，本手法は，骨の周囲に厚みや光学的性質の個体差のある軟組織が存在する場合でも，高い予測性能で BMD を評価できることが実証された．

4.6　本研究の制限と今後の課題

この研究には，いくつかの限界がある．まず，シミュレーションはあくまで理論的な検証に留まることである．しかしながら，シミュレーションは拡散光に影響を与える様々な組織について，理論的かつ明確な洞察を与えてくれる．また，モンテカルロシミュレーションと

機械学習技術の組み合わせは，光拡散理論を用いた非侵襲的医療計測の開発において，理論的検証を超えたいくつかの示唆を与えてくれると考えている．十分な分散と大量のデータで構築された機械学習モデルは優れた汎化性能を持つが，医療計測の分野では，症例数の制限や侵襲的な計測など，データ取得に倫理的な壁や困難があることが多い．計算機資源さえあればほぼ無尽蔵にデータを生成できるシミュレーションは，そのような問題に対する有力な解決策となり得る．第二に，VMC は 6 方向にしか境界を持たないため，より自由な境界を表現できるメッシュベースの手法 [71,110–112] と比較すると，明らかな制約がある．しかし，複雑な構造を持つ生体組織のような光散乱媒体の内部では，散乱によって光が平均化される．したがって，海綿状構造の曲率を過度に正確に表現することは，おそらく実用的でない．第三に，本研究では単純な直方体の生体組織モデルを採用した．このモデルでは，軟組織，血管，複雑な骨構造における不均質性を考慮していない．また，この生体組織モデルは，橈骨最遠位端を想定したものであるが，本来腕の軟組織厚さは掌側と背側で異なっており，骨自体の大きさも個人によって異なる．これらの情報は，CT や MRI の技術を用いてモデルに実装することが可能である．しかしながら，このモデルの潜在的な重要性は，光学式骨密度測定法の検証において，軟部組織の厚さと光学特性のランダムな変化を簡単かつ的を絞って議論することができる点である．このモデルは，拡散光を用いた測定における骨と軟部組織の間の相互作用について有用な情報を提供するかもしれない．ここで述べたすべての限界は，生体内での光の拡散現象とその出力を表現するために，実際の生体組織をどの程度モデルに想定すべきかが不明であること，そもそも仮想的な数値シミュレーションがどの程度現実に起こる現象を再現しているか不明であるとの 2 点に起因している．

今後の課題をいくつかのアイデアと共に説明する．

1 つ目は，バーチャルで生成した生体組織モデルの形状に関する検証である．本研究では，シンプルな直方体の生体組織を仮定したが，実際の骨は，さらに複雑な形態を持ち，軟組織の厚さも掌側と背側で異なっている．つまり，このような形態をもつ組織でも，本法が利用可能であるかは不明である．従って，このフェーズでは，幾つかの幾何学形状および，実際の CT データから生体組織モデルを生成し，本研究と同様の結果が得られるか調査する．

2 つ目は，本研究で提案された方法を in vitro で検証することである．具体的には，図 4.3 に示すような海綿骨と模擬皮膚から構成されるファントムを作成し，本研究で生成した機械学習モデルが実際に BMD を予測可能か検証する．実験的な検証には，図 4.14 に示す光学系

を提案する．本論では，光は 3 個方向から取得されるが，それぞれの方向に対してカメラを用意することは，実験精度面や経済的な面でもあまり現実的ではない．従って，提案された光学系は，１つのカメラと３つのコヒーレントレーザーで構成される．光は３つのレーザーから順番に照射され，その都度ファントム表面に形成される光強度の分布をカメラで取得する．この場合，カメラは，近赤外光データを取得可能なものを用い，図4.14のように斜めから撮影する．斜め方向からの撮影する理由は，カメラ自身がレーザーと干渉してしまうためであり，本来望ましいものではない．画像自体に斜め方向撮影に対する補正をかける，もしくは，光学系にハーフミラーを導入するという対策が必要である．このフェーズでは，理論値と実験値の定量的な一致が必要であるからであため，この他にも PC 上の仮想的な世界と現実世界を結びつけるための，何らかの工夫が必要になるかもしれない．

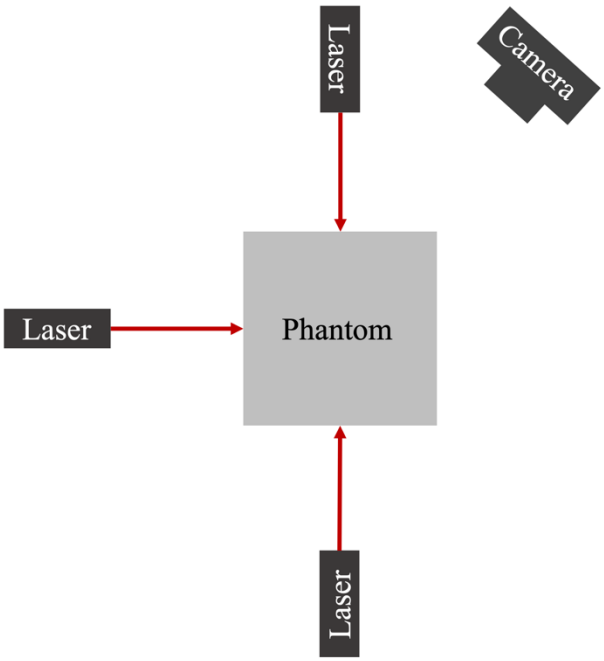


図 4.14　Phantom を用いた検証における光学系の案

３つ目は，組織の不均一性が考慮され，かつ安定的に光データが取得可能な方法を提案する必要がある．図 4.15 に示す，橈骨最遠位端の CT 断層画像のように，橈骨の形状は左右上下非対称であり，周囲の軟組織厚さは均一ではない．また，実際の組織を対象とした場合，計測の位置精度も重要となり，正確に安定して同様の計測が実施できることが求められる．

図4.14に示される光学系は，本論に多方向光学式骨密度計測法の検証では，使用できるものの，実際の組織を仮定した計測に対する適用はおそらく困難である．そこで，図4.16に示すような光学系を提案する．これは，光源と検出器を組織の周囲に交互に配置し，光を順番に入射させる．全ての方向から光を入射することで，組織の非対称性や，不均一性による影響を軽減することが可能かもしれない．また，図4.16は，検出器を組織に接触させている．このように配置することで，非接触型の計測法と比較して，安定した計測が可能となる．そして，おそらく，光入射時の皮膚表面における拡散反射の影響も軽減可能であると考えられる．

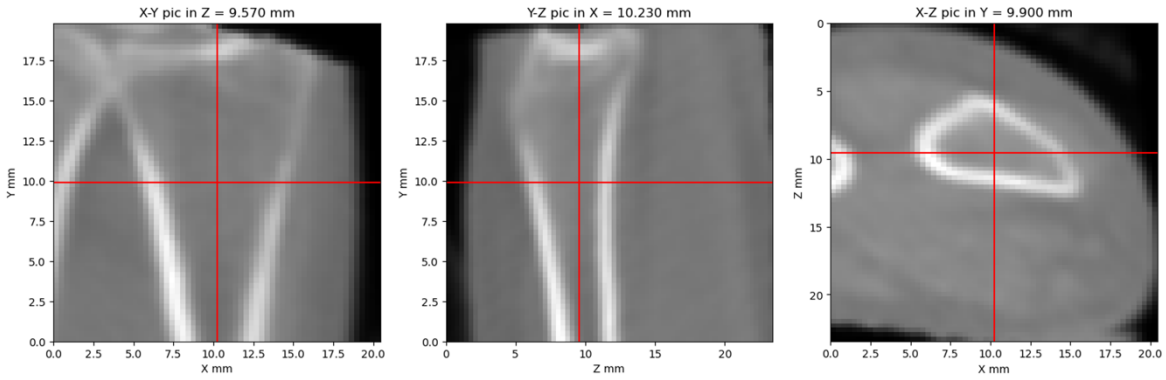


図 4.15　橈骨最遠位端の CT 断層画像．米国国立図書館 Visible Human Dataset より引用.

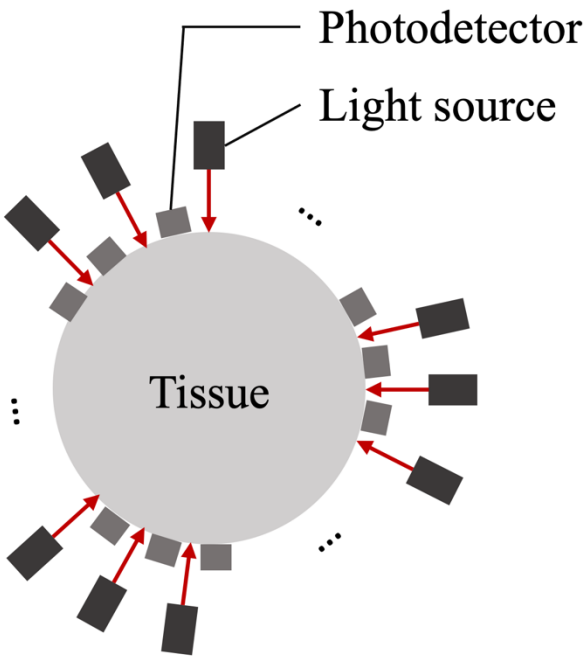


図 4.16　多方向光学式骨密度計測法の橈骨最遠位端計測のための光学系案.

最後に，多少飛躍した話になるが，臨床試験により，生成された機械学習モデルが，どの程度実際の人の BMD を予測するか調査することである．これは，どのようなモデルが実際の計測において，実質的に BMD を表現し得るか不明であるからである．本研究は，これらの一連の研究に対する，定量的な光学式 BMD 計測法の実現可能性を示す最初の研究に位置すると考えられる.

4.7 結言

本章では，モンテカルロシミュレーションと機械学習技術を用いた光学的骨密度測定法を開発し，クロスバリデーションによる検証を行った．得られた結果から，生体組織モデルの後方，前方，側方平面で取得した空間分解定常拡散光データから予測される aBMD は，軟組織層の厚さや光学特性の違いにロバストであると結論づけた.

第 5 章

結論

骨粗鬆症の早期発見のため，光学系を用いたいくつかの簡便な骨密度測定法が提案されてきた．しかしながら，軟部組織の構造や光学特性には個人差があり，その影響は定量的な骨密度測定を困難なものにしていた．そのため，軟部組織のばらつきに頑健な光学式骨密度測定法を開発することは，骨粗鬆症の早期発見のために重要である．本研究の目的は，モンテカルロシミュレーションと機械学習の手法を用いて，軟組織の影響を受けにくい光学式骨密度計を開発し，その実現可能性を検証することであった．

この目的を達成するため，本論では主に 3 つの課題に取り組んだ． 1 つ目は，骨梁パターンを持つ仮想的な海綿骨モデルを開発することである．これは，海綿骨特有の複雑構造によって生じる物理的な光子の挙動を再現するために必須であった．2 つ目は，骨梁の 3 次元構造を適用するための，モンテカルロシミュレーションモデルの開発である．このモンテカルロモデルは VMC（Voxel-based Monte Carlo）と名付けられ，立方体要素で構成された光拡散媒質モデル内の光子の伝搬挙動を計算することができる．3 つ目は，VMC で

本論の各章で得られた結果を以下に要約する．

第 2 章では，チューリングモデルを用いて生成した仮想海綿骨モデルの検討を行った．まず，Miura と Maini の論文 [79] を元に，チューリングモデルを用いた骨梁パターン生成のための，支配方程式について説明した．支配方程を解くために，フーリエ変換による陰解法を用いた．また，チューリングモデルを用いた周期的なパターン生成に関する計算例も示した．次に，チューリングモデルを用いた海綿骨モデルの生成を行った．海綿骨モデルの生成では，骨梁と骨梁間隙である骨梁間隙部分，骨のスケール，海綿骨部を覆う皮質骨を定義した．最後に，生成された海綿骨組織モデルが，定性的また定量的な観点から，どの程度実際の海綿骨に近いのか評価した．生成されたモデルは，ウシ海綿骨の μCT 画像に酷似しており，形態計測による定量的な評価においても，実際のヒト橈骨の海綿骨に似た構造を有していた．

第 3 章では，ボクセルベースのモンテカルロ法 (VMC) について述べた．まず，本論で開発された VMC のアルゴリズムについて説明した．VMC では，計算を高速化するため，GPU を用いた超並列計算を適用しており，これについても説明した．次に，過去に他の研究者によって実施されたモンテカルロ法と VMC を比較し，コードの妥当性を検証した．その結果，VMC の計算結果は，過去の研究結果と一致した．また，GPU を用いた計算により，どの程度高速化されるかも調査した．GPU の計算は，並列計算の数，つまりは光子数が増加することで，より高速になった．また，CPU での計算速度との比較では，40 倍ほど高速であった．これは体感的には，1 月以上かかる計算が，1 日足らずで終了する程の高速化である．

第 4 章では，モンテカルロ法でシミュレートされたデータを用いて，機械学習モデルを作成する方法並びに，生成された機械学習モデルの BMD 推定精度について述べた．まず，シミュレーションにより機械学習モデル訓練用のデータを生成する方法について説明した．シミュレーションデータは，第 2 章で説明した海綿骨モデルから合成生体組織モデルを生成し，それに対し第 3 章で説明した VMC を実施することで生成された．次に，シミュレーションデータと BMD を結びつけるための機械学習モデルについて説明した．機械学習モデルは，4 つの異なる構造をもつモデルを訓練し，最も予測性能が高いものを採用した．その結果，サポートベクターマシンが最も高い予測性能を示した．加えて，3 つの方向で取得されるデータとその組み合わせが骨密度の予測精度を向上させるか調査した．その結果，全ての方向で取得されるデータの組み合わせが最も高い精度で骨密度を予測した．また，各方向単体のデータでは，十分な予測精度が得られないことも判明した．最後に，本論で提案する多方向光学式骨密度計測法でどの程度の予測性能が得られるのかを検証した．その結果，予測値と参照値の骨密度の線形回帰は $r^2 = 0.760$ をもたらし，妥当な一致を示した．

本論は，モンテカルロシミュレーションと機械学習技術を用いた光学的骨密度測定法を開発し，その理論的な妥当性を検証した．得られた結果から，合成生体組織モデルの後方，前方，側方平面で取得した空間分解定常拡散光データから軟組織層の厚さや光学特性の違いにロバストな，高精度な骨密度予測が可能であると結論づけた．

参考文献

1. 骨粗鬆症の予防と治療ガイドライン作成委員会. 骨粗鬆症の予防と治療ガイドライン2015年版.
2. 骨粗鬆症の予防と治療のガイドライン作成委員会（日本骨粗鬆症学会 骨粗鬆症代謝学会 骨粗鬆症財団). 骨粗鬆症の予防と治療のガイドライン2011年度版.
3. NIH Consensus Development Panel on Osteoporosis Prevention, Diagnosis, and T. Osteoporosis Prevention, Diagnosis, and Therapy. *JAMA J. Am. Med. Assoc.* **285**, 785–795 (2001).
4. Heaney, R. P. *et al.* Peak bone mass. *Osteoporos. Int.* **11**, 985–1009 (2000).
5. Oleksik, A. *et al.* Health-related quality of life in postmenopausal women with low BMD with or without prevalent vertebral fractures. *J. Bone Miner. Res.* **15**, 1384–1392 (2000).
6. Hagino, H. *et al.* Sequential change in quality of life for patients with incident clinical fractures: A prospective study. *Osteoporos. Int.* **20**, 695–702 (2009).
7. Nguyen, N. D., Center, J. R., Eisman, J. A. & Nguyen, T. V. Bone loss, weight loss, and weight fluctuation predict mortality risk in elderly men and women. *J. Bone Miner. Res.* **22**, 1147–1154 (2007).
8. Ensrud, K. E. *et al.* Prevalent vertebral deformities predict mortality and hospitalization in older women with low bone mass. *J. Am. Geriatr. Soc.* **48**, 241–249 (2000).
9. Mafi Golchin, M., Heidari, L., Ghaderian, S. M. H. & Akhavan-Niaki, H. Osteoporosis: A Silent Disease with Complex Genetic Contribution. *J. Genet. Genomics* **43**, 49–61 (2016).
10. El Maghraoui, A. & Roux, C. DXA scanning in clinical practice. *Qjm* **101**, 605–617 (2008).
11. Kanis, J. A. & Glüer, C. C. An update on the diagnosis and assessment of osteoporosis with densitometry. *Osteoporosis International* vol. 11 192–202 (2000).
12. Cummings, S. R. *et al.* Bone density at various sites for prediction of hip fractures. *Lancet* **341**, 72–75 (1993).
13. Marshall, D., Johnell, O. & Wedel, H. Meta-analysis of how well measures of bone mineral density predict occurrence of osteoporotic fractures. *Bmj* **312**, 1254–1259 (1996).
14. Njeh, C. F., Kuo, C. W., Langton, C. M., Atrah, H. I. & Boivin, C. M. Prediction of human femoral bone strength using ultrasound velocity and BMD: An in vitro study. *Osteoporos. Int.* **7**, 471–477 (1997).
15. Miura, K., Matsubara, H. & Tanaka, S. M. Development of optical bone densitometry using near-infrared light. *J. Mech. Eng.* **5**, 60–67 (2018).
16. Chung, C., Chen, Y. P., Leu, T. H. & Sun, C. W. Near-infrared bone densitometry: A feasibility study on distal radius measurement. *J. Biophotonics* **11**, 1–5 (2018).
17. Anderson, R. R. & Parrish, J. . The Optics of Human Skin. *The Journal of investigative dermatology* vol. 77 13–9 (1981).
18. Ugryumova, N., Matcher, S. J. & Attenburrow, D. P. Measurement of bone mineral density via light scattering. *Phys. Med. Biol.* **49**, 469–483 (2004).
19. Ugryumova, N., Matcher, S. J. & Attenburrow, D. P. Optical studies of changes in bone mineral density. *Opt. Diagnostics Sens. Biomed. III* **4965**, 77 (2003).
20. Takeuchi, A. *et al.* A new method of bone tissue measurement based upon light scattering. *J.*

Bone Miner. Res. **12**, 261–266 (1997).

21. Pifferi, A. *et al.* Optical biopsy of bone tissue: a step toward the diagnosis of bone pathologies. *J. Biomed. Opt.* **9**, 474 (2004).
22. 内閣府. 平成29年版高齢社会白書 (2016).
23. Ministry of Health, L. a. . The Outline of the White Paper on the Labour Economy 2016. 1–519 (2016).
24. Kanis, J. A. & Kanis, J. A. Assessment of fracture risk and its application to screening for postmenopausal osteoporosis: Synopsis of a WHO report. *Osteoporos. Int.* **4**, 368–381 (1994).
25. Songpatanasilp, T. *et al.* Thai Osteoporosis Foundation (TOPF) position statements on management of osteoporosis. *Osteoporos. Sarcopenia* **2**, 191–207 (2016).
26. Parfitt, A. M. The cellular basis of bone remodeling: The quantum concept reexamined in light of recent advances in the cell biology of bone. *Calcif. Tissue Int.* **36**, (1984).
27. Parfitt, A. M. Osteonal and hemi-osteonal remodeling: The spatial and temporal framework for signal traffic in adult human bone. *J. Cell. Biochem.* **55**, 273–286 (1994).
28. Seeman, E. & Delmas, P. D. Bone quality--the material and structural basis of bone strength and fragility. *N. Engl. J. Med.* **354**, 2250–2261 (2006).
29. Parfitt, A. M. The coupling of bone formation to bone resorption: A critical analysis of the concept and of its relevance to the pathogenesis of osteoporosis. *Metab. Bone Dis. Relat. Res.* **4**, 1–6 (1982).
30. 学会編. 原発性骨粗鬆症の診断基準 （2012年度改訂版）. *Osteoporos. Japan* **21**, (2013).
31. Sinaki, M. *et al.* Stronger back muscles reduce the incidence of vertebral fractures: A prospective 10 year follow-up of postmenopausal women. *Bone* **30**, 836–841 (2002).
32. Gardner, M. M., Robertson, M. C. & Campbell, A. J. Exercise in preventing falls and fall related injuries in older people: A review of randomised controlled trials. *Br. J. Sports Med.* **34**, 7–17 (2000).
33. Karlsson, M. K., Vonschewelov, T., Karlsson, C., CÃster, M. & Rosengen, B. E. Prevention of falls in the elderly: A review. *Scand. J. Public Health* **41**, 442–454 (2013).
34. Howe, T. E. *et al.* Exercise for preventing and treating osteoporosis in postmenopausal women. *Cochrane Database Syst. Rev.* **2011**, 1–167 (2011).
35. Yamazaki, S., Ichimura, S., Iwamoto, J., Takeda, T. & Toyama, Y. Effect of walking exercise on bone metabolism in postmenopausal women with osteopenia/osteoporosis. *J. Bone Miner. Metab.* **22**, 500–508 (2004).
36. Asikainen, T. M., Kukkonen-Harjula, K. & Miilunpalo, S. Exercise for health for early postmenopausal women: A systematic review of randomised controlled trials. *Sport. Med.* **34**, 753–778 (2004).
37. Pfeifer, M. *et al.* Musculoskeletal rehabilitation in osteoporosis: A review. *J. Bone Miner. Res.* **19**, 1208–1214 (2004).
38. Bolland, M. J. *et al.* Vascular events in healthy older women receiving calcium supplementation: Randomised controlled trial. *Bmj* **336**, 262–266 (2008).
39. Bolland, M. J., Grey, A., Avenell, A., Gamble, G. D. & Reid, I. R. Calcium supplements with or without vitamin D and risk of cardiovascular events: Reanalysis of the Women's Health Initiative limited access dataset and meta-analysis. *Bmj* **342**, (2011).
40. Khosla, S. *et al.* Bisphosphonate-associated osteonecrosis of the jaw: Report of a Task Force of the American Society for Bone and Mineral Research. *J. Bone Miner. Res.* **22**, 1479–1491 (2007).

41. 米田俊之 *et al.* ビスフォスフォネート関連顎骨壊死に対するポジションペーパー. (ビスフォス フォネート関連顎骨壊死検討委員会 2012).

42. Galante, J., Rostoker, W. & Ray, R. D. Physical Properties of Trabecular Bone. *Calc Tiss Res* **5**, 236–246 (1970).

43. Arnold, J. S. Quantitation of mineralization of bone as an organ and tissue in osteoporosis. (1960).

44. Nayak, S. *et al.* Meta-analysis: Accuracy of quantitative ultrasound for identifying patients with osteoporosis. *Annals of Internal Medicine* vol. 144 832–841 (2006).

45. Hamdy, R. C., Petak, S. M. & Lenchik, L. Which central dual X-ray absorptiometry skeletal sites and regions of interest should be used to determine the diagnosis of osteoporosis? *J. Clin. Densitom.* **5**, (2002).

46. Miyamura, S. *et al.* Utility of Distal Forearm DXA as a Screening Tool for Primary Osteoporotic Fragility Fractures of the Distal Radius. *JBJS Open Access* **5**, e0036 (2020).

47. Wasnich, R. D., Ross, P. D., Davis, J. W. & Vogel, J. M. A comparison of single and multi-site BMC measurements for assessment of spine fracture probability. *J. Nucl. Med.* **30**, 1166–1171 (1989).

48. Marshall, D., Johnell, O. & Wedel, H. Meta-analysis of how well measures of bone mineral density predict occurrence of osteoporotic fractures. *Br. Med. J.* **312**, 1254–1259 (1996).

49. Cummings, S. R. *et al.* Risk factors for hip fracture in white women. *N. Engl. J. Med.* **332**, 767–774 (1995).

50. Pongchaiyakul, C., Panichkul, S., Songpatanasilp, T. & Nguyen, T. V. A nomogram for predicting osteoporosis risk based on age, weight and quantitative ultrasound measurement. *Osteoporos. Int.* **18**, 525–531 (2007).

51. Overman, R. A. *et al.* DXA Utilization Between 2006 and 2012 in Commercially Insured Younger Postmenopausal Women. *J. Clin. Densitom.* **18**, 145–149 (2015).

52. Elliot-Gibson, V., Bogoch, E. R., Jamal, S. A. & Beaton, D. E. Practice patterns in the diagnosis and treatment of osteoporosis after a fragility fracture: A systematic review. *Osteoporos. Int.* **15**, 767–778 (2004).

53. Holmberg, T. *et al.* Socioeconomic status and risk of osteoporotic fractures and the use of DXA scans: data from the Danish population-based ROSE study. *Osteoporos. Int.* **30**, 343–353 (2019).

54. Boonen, S. *et al.* Identifying postmenopausal women with osteoporosis by calcaneal ultrasound, metacarpal digital X-ray radiogrammetry and phalangeal radiographic absorptiometry: A comparative study. *Osteoporos. Int.* **16**, 93–100 (2005).

55. Adams, J. E. Quantitative computed tomography. *Eur. J. Radiol.* **71**, 415–424 (2009).

56. Michalski, A. S., Besler, B. A., Burt, L. A. & Boyd, S. K. Opportunistic CT screening predicts individuals at risk of major osteoporotic fracture. *Osteoporos. Int.* (2021) doi:10.1007/s00198-021-05863-0.

57. O'Malley, C. D. *et al.* Trends in Dual-Energy X-Ray Absorptiometry in the United States, 2000-2009. *J. Clin. Densitom.* **14**, 100–107 (2011).

58. Wall, A. & Board, T. The Compressive Behavior of Bone as a Two-Phase Porous Structure. in *Classic Papers in Orthopaedics* 457–460 (Springer London, 2014). doi:10.1007/978-1-4471-5451-8_116.

59. Baroncelli, G. I. Quantitative ultrasound methods to assess bone mineral status in children: Technical characteristics, performance, and clinical application. *Pediatr. Res.* **63**, 220–228 (2008).

60. Prins, S. H., Jørgensen, H. L., Jørgensen, L. V. & Hassager, C. The role of quantitative ultrasound in the assessment of bone: A review. *Clin. Physiol.* **18**, 3–17 (1998).

61. Patterson, M. S., Chance, B. & Wilson, B. C. Time resolved reflectance and transmittance for the noninvasive measurement of tissue optical properties. *Appl. Opt.* **28**, 2331 (1989).

62. Chaichanakol, S., Tanaka, S. M. & Khantachawana, A. QUANTITATIVE DETECTION OF CALCIUM USING NEAR- INFRARED SPECTROSCOPY FOR APPLY IN BONE. *Int. J. Mech. Prod. Eng.* **4**, 90–92 (2016).

63. 田中 茂雄, 野川 雅道, 山越 憲一 & 辻本 敏行. 近赤外光を利用した新規骨密度計測装置. 日本臨床バイオメカニクス学会誌 , Proc. ... Annu. Meet. Japanese Soc. Clin. Biomech. Relat. Res. **28**, 35–40 (2007).

64. Miura, K., Akae, H. & Tanaka, S. 光を用いた骨密度計測装置の開発とモンテカルロ法による検証. in 日本機械学会［No.167-1］北陸信越支部 第53期総会・講演会 講演論文集 [2016.3.5長野県長野市] 2–6 (2016).

65. Miura, K., Matsubara, H. & Tanaka, S. 近赤外光を用いた骨粗鬆症スクリーニング用簡易型骨密度計の開発. in 日本機械学会北陸信越支部 第54期総会・講演会 講演論文集 [2017.3.9石川県金沢市] (2017).

66. Bashkatov, A. N., Genina, E. A. & Tuchin, V. V. Optical properties of skin, subcutaneous, and muscle tissues: A review. *J. Innov. Opt. Health Sci.* **4**, 9–38 (2011).

67. Bashkatov, A. N., Genina, E. A., Kochubey, V. I. & Tuchin, V. V. Optical properties of human skin, subcutaneous and mucous tissues in the wavelength range from 400 to 2000 nm. *J. Phys. D. Appl. Phys.* **38**, 2543–2555 (2005).

68. 三浦要,柳瀬義寛, カンタチャワナアナック &田中茂雄. 光骨密度データを利用した機械学習技術による骨粗鬆症予測. in 日本機械学会北陸信越支部 第 58 期総会・講演会 講演論文集 (2021).

69. Chance, B., Dait, M. T., Zhang, C., Hamaoka, T. & Hagerman, F. Recovery from exercise-induced desaturation in the quadriceps muscles of elite competitive rowers. *Am. J. Physiol. - Cell Physiol.* **262**, (1992).

70. Mohri, M., Rostamizadeh, A. & Talwalkar, A. *Foundations of machine learning*. *The MIT Press* (2018).

71. Zhu, C. & Liu, Q. Review of Monte Carlo modeling of light transport in tissues. *J. Biomed. Opt.* **18**, 050902 (2013).

72. Burt, L. A., Liang, Z., Sajobi, T. T., Hanley, D. A. & Boyd, S. K. Sex- and Site-Specific Normative Data Curves for HR-pQCT. *J. Bone Miner. Res.* **xx**, 1–7 (2016).

73. Turing, A. M. The chemical basis of morphogenesis. *Philos. Trans. R. Soc. Lond. B. Biol. Sci.* **237**, 37–72 (1952).

74. Miura, T. & Shiota, K. Extracellular matrix environment influences chondrogenic pattern formation in limb bud micromass culture: Experimental verification of theoretical models. *Anat. Rec.* **258**, 100–107 (2000).

75. Yochelis, A., Tintut, Y., Demer, L. L. & Garfinkel, A. The formation of labyrinths, spots and stripe patterns in a biochemical approach to cardiovascular calcification. *New J. Phys.* **10**, (2008).

76. Rosen, E. O., McNamara, E. A., Whittaker, L. T. G., Malabanan, A. O. & Rosen, H. N. Effect of Positioning of the ROI on BMD of the Forearm and Its Subregions. *J. Clin. Densitom.* **21**, 529–533 (2018).

77. Wang, L., Jacquesa, S. L. & Zhengb, L. MCML - Monte Carlo modeling of light transport in

multi-layered tissues. *Biomedicine* **2607**, (1995).

78. Wang, L. & Jacques, S. L. Monte Carlo Modeling of Light Transport in Multi-layered Tissues in Standard C Monte Carlo Modeling of Light Transport in Multi-layered Tissues in Standard C. 173 (1992).

79. Miura, T. & Maini, P. K. Periodic pattern formation in reaction-diffusion systems: An introduction for numerical simulation. *Anat. Sci. Int.* **79**, 112–123 (2004).

80. Murray, J. D. *Mathematical Biology biomedical applications*. doi:10.1007/b98869.

81. 三浦岳. 発生の数理. (京都大学学術出版会, 2015).

82. Gierer, A. & Meinhardt, H. A theory of biological pattern formation. *Kybernetik* **12**, 30–39 (1972).

83. Boutroy, S., Bouxsein, M. L., Munoz, F. & Delmas, P. D. In vivo assessment of trabecular bone microarchitecture by high-resolution peripheral quantitative computed tomography. *J. Clin. Endocrinol. Metab.* **90**, 6508–6515 (2005).

84. Fazzalari, N. L. & Parkinson, I. H. Fractal dimension and architecture of trabecular bone. *J. Pathol.* **178**, 100–105 (1996).

85. Alberich-Bayarri, A. *et al.* Assessment of 2D and 3D fractal dimension measurements of trabecular bone from high-spatial resolution magnetic resonance images at 3 T. *Med. Phys.* **37**, 4930–4937 (2010).

86. Zhou, B. *et al.* High-resolution peripheral quantitative computed tomography (HR-pQCT) can assess microstructural and biomechanical properties of both human distal radius and tibia: Ex vivo computational and experimental validations. *Bone* **86**, 58–67 (2016).

87. Hildebrand, T. & Rüegsegger, P. Quantification of bone microarchitecture with the structure model index. *Comput. Methods Biomech. Biomed. Engin.* **1**, 15–23 (1997).

88. Odgaard, A. & Gundersen, H. J. . Quantification of connectivity with special emphasis on 3D reconstractions. *Bone* **14**, 173–182 (1993).

89. Wilson, B. C. & Adam, G. A Monte Carlo model for the absorption and flux distributions of light in tissue. *Med. Phys.* **10**, 824–830 (1983).

90. Wang, L., Jacques, S. L. & Zheng, L. Monte Carlo modeling of light transport in multi-layered tissues in standard C. *Comput. Methods Programs Biomed.* **47**, 131–146 (1995).

91. Carter, L. L. & Cashwell, E. D. *Particle-transport simulation with the monte carlo method*. *Development* (1975).

92. Prahl, S. A. A Monte Carlo model of light propagation in tissue. in *The International Society for Optical Engineering* (eds. Mueller, G. J., Sliney, D. H. & Potter, R. F.) vol. 5 102–111 (1989).

93. Wyman, D. R., Patterson, M. S. & Wilson, B. C. Similarity relations for anisotropic scattering in Monte Carlo simulations of deeply penetrating neutral particles. *J. Comput. Phys.* **81**, 137–150 (1989).

94. Simpson, C. R., Kohl, M., Essenpreis, M. & Cope, M. Near-infrared optical properties of ex vivo human skin and subcutaneous tissues measured using the Monte Carlo inversion technique. *Phys. Med. Biol.* **43**, 2465–2478 (1998).

95. Hassager, C., Borg, J. & Christiansen, C. Measurement of the subcutaneous fat in the distal forearm by single photon absorptiometry. *Metabolism* **38**, 159–165 (1989).

96. Kozarova, A., Kozar, M., Minarikova, E. & Pappova, T. Identification of the Age Related Skin Changes Using High-Frequency Ultrasound. *Acta Medica Martiniana* **17**, 15–20 (2017).

97. Williams, P. A. & Saha, S. The electrical and dielectric properties of human bone tissue and their relationship with density and bone mineral content. *Ann. Biomed. Eng.* **24**, 222–233 (1996).

98. Burkhardt, R. *et al.* Changes in trabecular bone, hematopoiesis and bone marrow vessels in aplastic anemia, primary osteoporosis, and old age: A comparative histomorphometric study. *Bone* **8**, 157–164 (1987).

99. Ascenzi, A. & Fabry, C. Technique for dissection and measurement of refractive index of osteones. *J. Biophys. Biochem. Cytol.* **6**, 139–142 (1959).

100. Hoerl, A. E. & Kennard, R. W. Ridge Regression: Applications to Nonorthogonal Problems. *Technometrics* **12**, 69–82 (1970).

101. Hoerl, A. E. & Kennard, R. W. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics* **12**, 55–67 (1970).

102. Lipo Wang. *Support Vector Machines: Theory and Applications*. *Studies in Fuzziness and Soft Computing* vol. 177 (Springer Berlin Heidelberg, 2005).

103. Breiman, L. Random forests. *Mach. Learn.* **45**, 5–32 (2001).

104. Quinlan, J. R. J. Ross Quinlan_C4.5_ Programs for Machine Learning. *Morgan Kaufmann* vol. 5 302 (1993).

105. Quinlan, J. R. Induction of decision trees. *Mach. Learn.* **1**, 81–106 (1986).

106. Olson, R. S., La Cava, W., Mustahsan, Z., Varik, A. & Moore, J. H. Data-driven Advice for Applying Machine Learning to Bioinformatics Problems. *Pacific Symp. Biocomput.* **0**, 192–203 (2017).

107. Friedman, J. H. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* **29**, (2001).

108. Friedman, J. H. Stochastic gradient boosting. *Comput. Stat. Data Anal.* **38**, 367–378 (2002).

109. Martin Bland, J. & Altman, D. Statistical Methods for Assessing Agreement Between Two Methods of Clinical Measurement. *Lancet* **327**, 307–310 (1986).

110. Yan, S., Tran, A. P. & Fang, Q. Dual-grid mesh-based Monte Carlo algorithm for efficient photon transport simulations in complex three-dimensional media. *J. Biomed. Opt.* **24**, 1 (2019).

111. Fang, Q. & Boas, D. A. Monte Carlo Simulation of Photon Migration in 3D Turbid Media Accelerated by Graphics Processing Units. **23**, 1–7 (2008).

112. Margallo-Balbás, E. & French, P. J. Shape based Monte Carlo code for light transport in complex heterogeneous Tissues. *Opt. Express* **15**, 14086 (2007).

付 録

A1 シミュレーションコード

シミュレーションコードの構造図を示す（図 A1.1）．本コード全体は，pyMonteOpt と呼ばれる名前がついている．シミュレーションの各要素は，Pymopt と呼ばれるモジュールで定義される．骨組織モデルのモデリングは，Pymopt 中の modeling および modeling_gpu 内の _classes.py ファイル内に記述されており，modeling は CPU，modeling_gpu は GPU での計算に対応している．次にモンテカルロシミュレーションは，voxel および voxel_gpu 内に記述されており，voxel は CPU，voxel_gpu は GPU での計算に対応している．シミュレーション全体の動作を記述しているのは，virtualOBD.py で，このファイルは，ipython notebook 用ファイルである vOBD.ipynb から動かす．次にこれらのファイルの詳細を記述する．

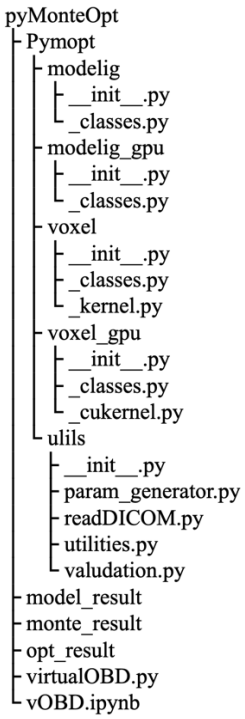


図 A1.1 シミュレーションコードのディレクトリ図

A1.1 骨組織モデリング用モジュール

__init__.py は，モジュール検索のためのマーカーであり，メインのコードは，_classes.py に記述した.

A1.1.1 modeling

pyMonteOpt/ Pymopt/ modeling/__init__.py

```
# -*- coding: utf-8 -*-
from ._classes import TuringPattern
__all__ = [
'TuringPattern',
]
```

pyMonteOpt/ Pymopt/ modeling/_classes.py

```
# -*- coding: utf-8 -*-
## *** All parameters should be defined in millimeters ***

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("ticks", {'grid.linestyle': '--'})
import gc
import datetime,time
import pandas as pa
from tqdm import tqdm
import sys
from scipy.signal import argrelmax
import json
from ..utils.utilities import set_params,ToJsonEncoder
```

```
from scipy.fftpack import fftn, ifftn

import pydicom
from pydicom.uid import ImplicitVRLittleEndian
from pydicom.dataset import Dataset, FileDataset

__all__ = [
'TuringPattern',
]

class TuringPattern:
    def __init__(self):
        self.dtype = 'float32'
        self.save_dicom=False
        self.params ={
        'grid':40,
        'dx':1/40,
        'dt':1,
        'du':0.0002,
        'dv':0.01,
        'length':13,
        'repetition':100,
        'bv_tv':0.138,
        'voxelsize':0.0295,
        'seed':False,
        'ct_coef':4.5e4,
        'tile_num_xz':1,
        'tile_num_y':1,
        }
        self.coef = np.array([-7.67681281,1.65199492,-0.45314158,0.60424417])
        self.keys_params  = list(self.params.keys())
        self._setatter_params()
        self.f = lambda u,v: u+self.dt*(0.6*u-v-u**3)
        self.g = lambda u,v: v+self.dt*(1.5*u-2*v)

    def modeling(self,path,save_dicom=False):
        self.save_dicom=save_dicom
        self._calc_kukv()
        u,v = self._get_inital_vector()
        for i in tqdm(range(self.repetition)):
            u,v = self._calc_onestep(u,v)
        self.model_shape=u.shape
        print("Model Size: %s Mb"%(sys.getsizeof(u)/1e6))
```

```
    del v
    gc.collect()
    if save_dicom:
        self._save_dicom(u,path)
    u = self._adjust_vbtv(u)
    self._calc_microarchitecture(u)
    self._save_info(path)
    u = self._model_binarization(u)
    if self.tile_num_xz != 0:
        u = np.tile(u, (self.tile_num_xz,self.tile_num_y,self.tile_num_xz))
    return u

def set_threshold_func_coef(self,coef):
    self.coef = np.array(coef)

def set_params(self,*initial_data, **kwargs):
    set_params(self.params,self.keys_params,*initial_data, **kwargs)
    self._setatter_params()

def set_inhibitor_function(self,g):
    self.g = g

def set_activator_function(self,f):
    self.f = f

def _setatter_params(self):
    for k in self.params:
        setattr(self,k,self.params[k])
    self.gridsize=self.length*self.grid

def _kernel_vector3(self,dd):
    ke = np.float32(-self.dt*dd/self.dx**2)
    kernel = np.zeros((self.gridsize,self.gridsize,self.gridsize)).astype(self.dtype)
    kernel[0,0,0] = np.float32(1+6*self.dt*dd/self.dx**2)
    kernel[1,0,0] = ke
    kernel[-1,0,0] = ke
    kernel[0,-1,0] = ke
    kernel[0,1,0] = ke
    kernel[0,0,-1] = ke
    kernel[0,0,1] = ke
    return kernel

def _calc_kukv(self):
```

```
    ku = self._kernel_vector3(self.du)
    kv = self._kernel_vector3(self.dv)
    ku = 1/(fftn(ku).real.astype('float32'))
    kv = 1/(fftn(kv).real.astype('float32'))
    self.ku = ku
    self.kv = kv

def _calc_onestep(self,u,v):
    a = ifftn(self.ku*fftn(self.f(u,v)).real.astype(self.dtype)).real.astype(self.dtype)
    b = ifftn(self.kv*fftn(self.g(u,v)).real.astype(self.dtype)).real.astype(self.dtype)
    return a,b

def _get_inital_vector(self):
    gs = self.gridsize
    if self.seed:
        np.random.seed(seed=int(self.seed))
    ui = (1-np.random.rand(gs,gs,gs).astype(self.dtype))
    if self.seed:
        np.random.seed(seed=int(self.seed*2))
    vi = (1-np.random.rand(gs,gs,gs).astype(self.dtype))
    return ui,vi

def _get_threshold(self):
    x = self.bv_tv
    self.threshold = np.poly1d(self.coef)(x)

def _adjust_vbtv(self,u):
    self._get_threshold()
    ind = np.where(u<=self.threshold)
    u[ind[0],ind[1],ind[2]] = 0
    self.bv_tv_real = 1-ind[0].shape[0]/(u.shape[0]*u.shape[1]*u.shape[2])
    print("BV/TV = %.4f"%self.bv_tv_real)
    return u

def _calc_microarchitecture(self,u):
    def count_peak_par_axis(sub_pic):
        a = 0
        for i in sub_pic:
            a += argrelmax(i,order = 5)[0].shape[0]
        return a/sub_pic.shape[0]
    self.num_peak = 0
    for i in u:
        self.num_peak+=count_peak_par_axis(i)
```

```
    tbn = self.num_peak/(self.voxelsize*self.gridsize**2)

    self.microarchitecture = {
    'BV/TV':self.bv_tv,
    'BV/TV real':self.bv_tv_real,
    'Tb.N* [/mm]':tbn,
    'Tb.Th [mm]':(self.bv_tv_real/tbn),
    'Tb.Sp [mm]':((1-self.bv_tv_real)/tbn),
    'Dtorb [mg/cm3]':(self.bv_tv_real*1200),
    }

def _model_binarization(self,u):
    ind = np.where(u>0)
    u[ind[0],ind[1],ind[2]] = 1
    return u.astype("int8")

def _save_info(self,path,coment=""):
    info = {
        'Date':datetime.datetime.now().isoformat(),
        'coment':coment,
        'model_params':self.params,
        'microarchitecture':self.microarchitecture,
        'model_shape':self.model_shape,
    }
    save_name = path+"/_info.json"
    with open(save_name, 'w') as fp:
        json.dump(info,fp,indent=4,cls= ToJsonEncoder)

def _save_dicom(self,pixel_array,path):
    pixel_array = (pixel_array.copy()*self.ct_coef).astype("uint16")
    for i in range(0, np.shape(pixel_array)[2]):
        self._write_dicom(pixel_array[:,:,i], i,path,self.voxelsize)
    del pixel_array
    gc.collect()

def _write_dicom(self,pixel_array, level,path,voxelsize):
    suffix=  f'{level:04}' + ".dcm"

    filename_endian= path+"/" + suffix
    file_meta= Dataset()
    file_meta.TransferSyntaxUID= ImplicitVRLittleEndian
    file_meta.MediaStorageSOPClassUID = pydicom.uid.generate_uid()
    file_meta.MediaStorageSOPInstanceUID = pydicom.uid.generate_uid()
```

```
    file_meta.ImplementationClassUID = pydicom.uid.generate_uid()

    ds = FileDataset(filename_endian, {}, file_meta= file_meta, preamble= b"\0"*128)
    ds.Modality = 'WSD'
    ds.ContentDate = str(datetime.date.today()).replace('-','')
    ds.ContentTime = '010101.000000' #milliseconds since the epoch
    ds.StudyInstanceUID =  pydicom.uid.generate_uid()
    ds.SeriesInstanceUID = pydicom.uid.generate_uid()
    ds.SOPInstanceUID =    pydicom.uid.generate_uid()
    ds.SOPClassUID = pydicom.uid.generate_uid()
    #ds.SecondaryCaptureDeviceManufctur = 'Python 3.6.5'

    ## These are the necessary imaging components of the FileDataset object.
    ds.SamplesPerPixel = 1
    ds.PhotometricInterpretation = "MONOCHROME2"
    ds.PixelRepresentation = 0
    ds.HighBit = 15
    ds.BitsStored = 16
    ds.BitsAllocated = 16
    #ds.SmallestImagePixelValue = pixel_array.min()
    #ds[0x00280106].VR= 'US'
    ds.LargestImagePixelValue = pixel_array.max()
    ds[0x00280107].VR= 'US'
    ds.Columns = pixel_array.shape[0]
    ds.Rows = pixel_array.shape[1]
    ds.PixelSpacing = [voxelsize, voxelsize]
    """if pixel_array.dtype != np.uint16:
       pixel_array = pixel_array.astype(np.uint16)"""
    ds.PixelData = pixel_array.tobytes()

    ds.save_as(filename_endian)



def linear_stability_analysis(self):
    dp = self.du
    dq = self.dv

    import sympy
    p = sympy.Symbol('p')
    q = sympy.Symbol('q')
    intialUV = sympy.solve([self.f(p,q),self.g(p,q)])

    print('initial value')
```

```
print(intialUV)
try:
   intialUV = intialUV[0]
except:
   pass

fu = sympy.diff(f(p,q),p).subs([(p, intialUV[p])])
fv = sympy.diff(f(p,q),q).subs([(q, intialUV[q])])
gu = sympy.diff(g(p,q),p).subs([(p, intialUV[p])])
gv = sympy.diff(g(p,q),q).subs([(q, intialUV[q])])
print()
print("fu = %.3f"%fu)
print("fv = %.3f"%fv)
print("gu = %.3f"%gu)
print("gv = %.3f"%gv)

la = sympy.Symbol('la')
k = sympy.Symbol('k')
dispersion_relation = sympy.solve((la-fu+dp*k**2)*(la-gv+dq*k**2)-fv*gu,la)
print()
print("Solution of λ")
print(dispersion_relation[0])
print(dispersion_relation[1])

wave_range =np.arange(100)
growth_rate = []
for i in wave_range:
   ans0 = dispersion_relation[0].subs([(k, i)])
   ans1 = dispersion_relation[1].subs([(k, i)])
   growth_rate.append(np.array([ans0,ans1]).astype(float))
growth_rate = np.array(growth_rate).T

fig, axis = plt.subplots(ncols=2,sharex=True,figsize=(10,4), dpi = 100)
axis[0].plot(wave_range/(2*np.pi),growth_rate[0],c = 'k')
axis[0].plot(np.array([wave_range[0],wave_range[-1]])/(2*np.pi),[0,0],'--',c = 'k')
axis[0].set_title("Top : %.3f" %(wave_range[growth_rate[0].argmax()]/(2*np.pi)))
axis[0].set_xlabel('Number of repetition per unit length')
axis[1].plot(wave_range/(2*np.pi),growth_rate[1],c = 'k')
axis[1].plot(np.array([wave_range[0],wave_range[-1]])/(2*np.pi),[0,0],'--',c = 'k')
axis[1].set_title("Top : %.3f" %(wave_range[growth_rate[1].argmax()]/(2*np.pi)))
axis[1].set_xlabel('Number of repetition per unit length')
plt.show()
```

A1.1.2 modeling_gpu

pyMonteOpt/ Pymopt/ modelig/__init__.py

```
# -*- coding: utf-8 -*-
from ._classes import TuringPattern
__all__ = [
'TuringPattern',
]
```

pyMonteOpt/ Pymopt/ modeling_gpu/_classes.py

```
# -*- coding: utf-8 -*-
## *** All parameters should be defined in millimeters ***

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("ticks", {'grid.linestyle': '--'})
import gc
import datetime,time
import pandas as pa
from tqdm import tqdm
import sys
from scipy.signal import argrelmax
import json
from ..utils.utilities import set_params,ToJsonEncoder
import cupy as cp

import pydicom
from pydicom.uid import ImplicitVRLittleEndian
from pydicom.dataset import Dataset, FileDataset

__all__ = [
'TuringPattern',
]
```

```
class TuringPattern:
    def __init__(self):
        self.dtype = 'float32'
        self.save_dicom=False
        self.params ={
        'grid':40,
        'dx':1/40,
        'dt':1,
        'du':0.0002,
        'dv':0.01,
        'length':13,
        'repetition':100,
        'bv_tv':0.138,
        'voxelsize':0.0295,
        'seed':False,
        'ct_coef':4.5e4,
        'tile_num_xz':1,
        'tile_num_y':1,
        }
        self.coef = np.array([-7.67681281,1.65199492,-0.45314158,0.60424417])
        self.keys_params  = list(self.params.keys())
        self._setatter_params()
        self.f = lambda u,v: u+self.dt*(0.6*u-v-u**3)
        self.g = lambda u,v: v+self.dt*(1.5*u-2*v)

    def modeling(self,path,save_dicom=False):
        self.save_dicom=save_dicom
        mempool = cp.get_default_memory_pool()
        pinned_mempool = cp.get_default_pinned_memory_pool()
        mempool.free_all_blocks()
        pinned_mempool.free_all_blocks()
        self._calc_kukv()
        u,v = self._get_inital_vector()
        for i in tqdm(range(self.repetition)):
            u,v = self._calc_onestep(u,v)
        self.model_shape=u.shape
        print("Model Size: %s Mb"%(sys.getsizeof(u)/1e6))
        U = cp.asnumpy(u)
        del self.ku, self.kv, u,v
        gc.collect()
        mempool.free_all_blocks()
        pinned_mempool.free_all_blocks()
        if save_dicom:
```

```
            self._save_dicom(U,path)
        U = self._adjust_vbtv(U)
        self._calc_microarchitecture(U)
        self._save_info(path)
        U = self._model_binarization(U)
        if self.tile_num_xz != 0:
            U = np.tile(U, (self.tile_num_xz,self.tile_num_y,self.tile_num_xz))
        return U

    def set_threshold_func_coef(self,coef):
        self.coef = np.array(coef)

    def set_params(self,*initial_data, **kwargs):
        set_params(self.params,self.keys_params,*initial_data, **kwargs)
        self._setatter_params()

    def set_inhibitor_function(self,g):
        self.g = g

    def set_activator_function(self,f):
        self.f = f

    def _setatter_params(self):
        for k in self.params:
            setattr(self,k,self.params[k])
        self.gridsize=self.length*self.grid

    def _kernel_vector3(self,dd):
        ke = np.float32(-self.dt*dd/self.dx**2)
        kernel = np.zeros((self.gridsize,self.gridsize,self.gridsize)).astype(self.dtype)
        kernel[0,0,0] = np.float32(1+6*self.dt*dd/self.dx**2)
        kernel[1,0,0] = ke
        kernel[-1,0,0] = ke
        kernel[0,-1,0] = ke
        kernel[0,1,0] = ke
        kernel[0,0,-1] = ke
        kernel[0,0,1] = ke
        return kernel

    def _calc_kukv(self):
        self.ku = cp.asarray(self._kernel_vector3(self.du))
        self.kv = cp.asarray(self._kernel_vector3(self.dv))
        self.ku = 1/(cp.fft.fftn(self.ku).real.astype('float32'))
```

```
    self.kv = 1/(cp.fft.fftn(self.kv).real.astype('float32'))

def _calc_onestep(self,u,v):
    a = cp.real(cp.fft.ifftn(self.ku*cp.real(cp.fft.fftn(self.f(u,v)))))
    b = cp.real(cp.fft.ifftn(self.kv*cp.real(cp.fft.fftn(self.g(u,v)))))
    return a,b

def _get_inital_vector(self):
    gs = self.gridsize
    if self.seed:
        np.random.seed(seed=int(self.seed))
    ui = (1-np.random.rand(gs,gs,gs).astype(self.dtype))
    if self.seed:
        np.random.seed(seed=int(self.seed*2))
    vi = (1-np.random.rand(gs,gs,gs).astype(self.dtype))
    return cp.asarray(ui),cp.asarray(vi)

def _get_threshold(self):
    x = self.bv_tv
    self.threshold = np.poly1d(self.coef)(x)

def _adjust_vbtv(self,u):
    self._get_threshold()
    ind = np.where(u<=self.threshold)
    u[ind[0],ind[1],ind[2]] = 0
    self.bv_tv_real = 1-ind[0].shape[0]/(u.shape[0]*u.shape[1]*u.shape[2])
    print("BV/TV = %.4f"%self.bv_tv_real)
    return u

def _calc_microarchitecture(self,u):
    def count_peak_par_axis(sub_pic):
        a = 0
        for i in sub_pic:
            a += argrelmax(i,order = 5)[0].shape[0]
        return a/sub_pic.shape[0]
    self.num_peak = 0
    for i in u:
        self.num_peak+=count_peak_par_axis(i)
    tbn = self.num_peak/(self.voxelsize*self.gridsize**2)

    self.microarchitecture = {
    'BV/TV':self.bv_tv,
    'BV/TV real':self.bv_tv_real,
```

```
    'Tb.N* [/mm]':tbn,
    'Tb.Th [mm]':(self.bv_tv_real/tbn),
    'Tb.Sp [mm]':((1-self.bv_tv_real)/tbn),
    'Dtorb [mg/cm3]':(self.bv_tv_real*1200),
    }

def _model_binarization(self,u):
    ind = np.where(u>0)
    u[ind[0],ind[1],ind[2]] = 1
    return u.astype("int8")

def _save_info(self,path,coment=""):
    info = {
        'Date':datetime.datetime.now().isoformat(),
        'coment':coment,
        'model_params':self.params,
        'microarchitecture':self.microarchitecture,
        'model_shape':self.model_shape,
    }
    save_name = path+"/_info.json"
    with open(save_name, 'w') as fp:
        json.dump(info,fp,indent=4,cls= ToJsonEncoder)

def _save_dicom(self,pixel_array,path):
    pixel_array = (pixel_array.copy()*self.ct_coef).astype("uint16")
    for i in range(0, np.shape(pixel_array)[2]):
        self._write_dicom(pixel_array[:,:,i], i,path,self.voxelsize)
    del pixel_array
    gc.collect()

def _write_dicom(self,pixel_array, level,path,voxelsize):
    suffix=  f'{level:04}' + ".dcm"

    filename_endian= path+"/" + suffix
    file_meta= Dataset()
    file_meta.TransferSyntaxUID= ImplicitVRLittleEndian
    file_meta.MediaStorageSOPClassUID = pydicom.uid.generate_uid()
    file_meta.MediaStorageSOPInstanceUID = pydicom.uid.generate_uid()
    file_meta.ImplementationClassUID = pydicom.uid.generate_uid()

    ds = FileDataset(filename_endian, {}, file_meta= file_meta, preamble= b"\0"*128)
    ds.Modality = 'WSD'
    ds.ContentDate = str(datetime.date.today()).replace('-','')
```

```
    ds.ContentTime = '010101.000000' #milliseconds since the epoch
    ds.StudyInstanceUID =  pydicom.uid.generate_uid()
    ds.SeriesInstanceUID = pydicom.uid.generate_uid()
    ds.SOPInstanceUID =    pydicom.uid.generate_uid()
    ds.SOPClassUID = pydicom.uid.generate_uid()
    #ds.SecondaryCaptureDeviceManufctur = 'Python 3.6.5'

    ## These are the necessary imaging components of the FileDataset object.
    ds.SamplesPerPixel = 1
    ds.PhotometricInterpretation = "MONOCHROME2"
    ds.PixelRepresentation = 0
    ds.HighBit = 15
    ds.BitsStored = 16
    ds.BitsAllocated = 16
    #ds.SmallestImagePixelValue = pixel_array.min()
    #ds[0x00280106].VR= 'US'
    ds.LargestImagePixelValue = pixel_array.max()
    ds[0x00280107].VR= 'US'
    ds.Columns = pixel_array.shape[0]
    ds.Rows = pixel_array.shape[1]
    ds.PixelSpacing = [voxelsize, voxelsize]
    """if pixel_array.dtype != np.uint16:
        pixel_array = pixel_array.astype(np.uint16)"""
    ds.PixelData = pixel_array.tobytes()
    ds.save_as(filename_endian)

def linear_stability_analysis(self):
    dp = self.du
    dq = self.dv

    import sympy
    p = sympy.Symbol('p')
    q = sympy.Symbol('q')
    intialUV = sympy.solve([self.f(p,q),self.g(p,q)])

    print('initial value')
    print(intialUV)
    try:
        intialUV = intialUV[0]
    except:
        pass

    fu = sympy.diff(f(p,q),p).subs([(p, intialUV[p])])
```

```
fv = sympy.diff(f(p,q),q).subs([(q, intialUV[q])])
gu = sympy.diff(g(p,q),p).subs([(p, intialUV[p])])
gv = sympy.diff(g(p,q),q).subs([(q, intialUV[q])])
print()
print("fu = %.3f"%fu)
print("fv = %.3f"%fv)
print("gu = %.3f"%gu)
print("gv = %.3f"%gv)

la = sympy.Symbol('la')
k = sympy.Symbol('k')
dispersion_relation = sympy.solve((la-fu+dp*k**2)*(la-gv+dq*k**2)-fv*gu,la)
print()
print("Solution of λ")
print(dispersion_relation[0])
print(dispersion_relation[1])

wave_range =np.arange(100)
growth_rate = []
for i in wave_range:
    ans0 = dispersion_relation[0].subs([(k, i)])
    ans1 = dispersion_relation[1].subs([(k, i)])
    growth_rate.append(np.array([ans0,ans1]).astype(float))
growth_rate = np.array(growth_rate).T

fig, axis = plt.subplots(ncols=2,sharex=True,figsize=(10,4), dpi = 100)
axis[0].plot(wave_range/(2*np.pi),growth_rate[0],c = 'k')
axis[0].plot(np.array([wave_range[0],wave_range[-1]])/(2*np.pi),[0,0],'--',c = 'k')
axis[0].set_title("Top : %.3f" %(wave_range[growth_rate[0].argmax()]/(2*np.pi)))
axis[0].set_xlabel('Number of repetition per unit length')
axis[1].plot(wave_range/(2*np.pi),growth_rate[1],c = 'k')
axis[1].plot(np.array([wave_range[0],wave_range[-1]])/(2*np.pi),[0,0],'--',c = 'k')
axis[1].set_title("Top : %.3f" %(wave_range[growth_rate[1].argmax()]/(2*np.pi)))
axis[1].set_xlabel('Number of repetition per unit length')
plt.show()
```

A1.2　VMC モジュール

_classes.py では，合成生体組織を定義し，モンテカルロシミュレーションの初期値を決定，並びにシミュレーション計算用のカーネルの制御を記述した．モンテカルロシミュレーションのコードは，kernel.py ファイルに記載されている．

A1.2.1 voxel

pyMonteOpt/ Pymopt/ voxel/__init__.py

```
from ._kernel import vmc_kernel
from ._classes import VoxelPlateModel
from ._classes import VoxelTuringModel

__all__ = [
'vmc_kernel',
'VoxelPlateModel',
'VoxelTuringModel',
]
```

pyMonteOpt/ Pymopt/ voxel/_classes.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from ._kernel import vmc_kernel

import numpy as np
import os
import pydicom
from scipy import stats
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import colors
from abc import ABCMeta, abstractmethod
```

```
import datetime,time
import json,pickle,bz2

from ..utils import readDicom,reConstArray_8,reConstArray
from ..utils.validation import _deprecate_positional_args
from ..fluence import Fluence2D,Fluence3D
from ..utils.utilities import calTime,set_params,ToJsonEncoder
from ..optics._classes import Grass
import gc
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
#os.system("taskset -p 0xff %d" % os.getpid())

__all__ = [
'VoxelPlateModel','VoxelTuringModel'
]

# ====================================================================
# Base solid model
# ====================================================================

class BaseVoxelMonteCarlo(metaclass = ABCMeta):
    #@_deprecate_positional_args
    @abstractmethod
    def __init__(self,*,nPh,model,dtype_f=np.float32,dtype=np.int32,
                 beam_type = 'TEM00',w_beam = 0,
                 beam_angle = 0,initial_refrect_by_angle = False,
                 first_layer_clear = False,
                 ):

        def __check_list_name(name,name_list):
            if not(name in name_list):
                raise ValueError('%s is not a permitted for factor. Please choose from %s.'%(name,name_list))

        self.beam_type_list=['TEM00',False]
        __check_list_name(beam_type,self.beam_type_list)
        self.beam_type = beam_type

        self.dtype = dtype
        self.dtype_f = dtype_f
        self.nPh = nPh
        self.w_beam = w_beam
```

```
        self.initial_refrect_by_angle = initial_refrect_by_angle
        self.beam_angle = beam_angle

        self.model = model
        self.first_layer_clear=first_layer_clear

    def start(self):
        self.nPh = int(self.nPh)
        self._reset_results()
        self._generate_initial_coodinate(self.nPh)

        self.add = self.add.astype(np.int32)
        self.p = self.p.astype(np.float32)
        self.v = self.v.astype(np.float32)
        self.w = self.w.astype(np.float32)
        print("")
        print("###### Start ######")
        print("")
        start_ = time.time()
        self.add,self.p,self.v,self.w = vmc_kernel(
            self.add, self.p,self.v, self.w,
            self.model.ma, self.model.ms, self.model.n, self.model.g,
            self.model.voxel_model, self.model.voxel_space,
            np.int32(self.nPh), np.int8(self.model.end_point)
        )

        self._end_process()
        print("###### End ######")
        self.getRdTtRate()
        calTime(time.time(), start_)
        #del func
        return self

    def _end_process(self):#書き換え
        #index = np.where(~np.isnan(self.w))[0]
        self.v_result = self.v#[:,index]
        self.p_result = self.p#[:,index]
        self.add_result = self.add#[:,index]
        self.w_result = self.w#[index]

    def _reset_results(self):
```

```
    self.v_result = np.empty((3,1)).astype(self.dtype_f)
    self.p_result = np.empty((3,1)).astype(self.dtype_f)
    self.add_result = np.empty((3,1)).astype(self.dtype)
    self.w_result = np.empty(1).astype(self.dtype_f)
    return self

def get_voxel_model(self):
    return self.model.voxel_model

def _generate_initial_coodinate(self,nPh):
    self._set_inital_add()
    self._set_beam_distribution()
    self._set_inital_vector()
    self._set_inital_w()



def _set_inital_add(self):
    if self.beam_type == 'TEM00':
        self.add =  np.zeros((3, self.nPh),dtype = self.dtype)
    self.add[0] = self._get_center_add(self.model.voxel_model.shape[0])
    self.add[1] = self._get_center_add(self.model.voxel_model.shape[1])
    if self.first_layer_clear:
        self.add[2] = self.model.get_second_layer_addz()
    else:
        self.add[2] = 1

def _get_center_add(self,length):
    #add の中心がローカル座標（ボックス内）の中心となるため,
    #ボクセル数が偶数の時は，1/2 小さい位置を中心とし光を照射し,
    #逆変換時（_encooder）も同様に 1/2 小さい位置を中心として元のマクロな座標に戻す.
    return int((length-1)/2)

def _set_inital_vector(self):
    if self.beam_type == 'TEM00':
        self.v = np.zeros((3,self.nPh)).astype(self.dtype_f)
        self.v[2] = 1
        if self.beam_angle!=0 and self.w_beam==0:
            #ビーム径がある場合はとりあえず無視
            #角度は rad 表記
            ni = self.model.n[-1]
            nt = self.model.n[0]
            ai = self.beam_angle
```

```
            at = np.arcsin(np.sin(ai)*ni/nt)
            self.v[0] = np.sin(at)
            self.v[2] = np.cos(at)
            if self.initial_refrect_by_angle:
                Ra = ((np.sin(ai-at)/np.sin(ai+at))**2\
                +(np.tan(ai-at)/np.tan(ai+at))**2)/2

                self.inital_del_num = np.count_nonzero(Ra>=np.random.rand(self.nPh))
                self.v = np.delete(self.v, np.arange(self.inital_del_num), 1)
                self.p = np.delete(self.p, np.arange(self.inital_del_num), 1)
                self.add = np.delete(self.add, np.arange(self.inital_del_num), 1)
                sub_v = np.zeros((3,self.inital_del_num)).astype(self.dtype_f)
                sub_v[0] = np.sin(ai)
                sub_v[2] = -np.cos(ai)
                self.v_result = np.concatenate([self.v_result,
                sub_v],axis = 1)
                self.p_result = np.concatenate([self.p_result,
                self.p[:,:self.inital_del_num]],axis = 1)
                self.add_result = np.concatenate([self.add_result,
                self.add[:,:self.inital_del_num]],axis = 1)
        else:
            print("ビームタイプが設定されていません")

    def _set_inital_w(self):
        if self.beam_type == 'TEM00':
            self.w = np.ones(self.nPh).astype(self.dtype_f)
            Rsp = 0
            n1 = self.model.n[-1]
            n2 = self.model.n[0]
            if n1 != n2:
                Rsp = ((n1-n2)/(n1+n2))**2
                if self.beam_angle!=0 and self.w_beam==0:
                    ai = self.beam_angle
                    at = np.arcsin(np.sin(ai)*n1/n2)
                    Rsp = ((np.sin(ai-at)/np.sin(ai+at))**2\
                    +(np.tan(ai-at)/np.tan(ai+at))**2)/2
                elif self.first_layer_clear:
                    n3=self.model.n[1]
                    r2 = ((n3-n2)/(n3+n2))**2
                    Rsp = Rsp+r2*(1-Rsp)**2/(1-Rsp*r2)
                self.w -= Rsp
```

```
        if self.beam_angle!=0 and self.w_beam==0:
            if self.initial_refrect_by_angle:
                self.w[:] = 1
                self.w = np.delete(self.w, np.arange(self.inital_del_num), 0)
                self.w_result = np.concatenate([self.w_result,
                self.w[:self.inital_del_num]],axis = 0)
    else:
        print("ビームタイプが設定されていません")

def _set_beam_distribution(self):
    if self.beam_type == 'TEM00':
        self.p = np.zeros((3,self.nPh)).astype(self.dtype_f)
        self.p[2] = -self.model.voxel_space/2
        if self.w_beam!= 0:
            print("%s を入力"%self.beam_type)
            #ガウシアン分布を生成
            gb = np.array(self.gaussianBeam(self.w_beam)).astype(self.dtype_f)
            #ガウシアン分布を各アドレスに振り分ける

            l = self.model.voxel_space
            pp = (gb/l).astype("int16")
            ind = np.where(gb<0)
            pp[ind[0].tolist(),ind[1].tolist()] -= 1
            pa = gb - (pp+1/2)*l
            ind = np.where((np.abs(pa)>=l/2))
            pa[ind[0].tolist(),ind[1].tolist()] = \
                np.sign(pa[ind[0].tolist(),ind[1].tolist()])*(l/2)
            pa += l/2
            self.add[:2] = self.add[:2] + pp
            self.p[:2] = pa.astype(self.dtype_f)
    else:
        print("ビームタイプが設定されていません")

def _get_beam_dist(self,x,y):
    fig = plt.figure(figsize=(10,6),dpi=70)
    ax = fig.add_subplot(111)
    ax.set_aspect('equal')
    H = ax.hist2d(x,y, bins=100,cmap="plasma")
    ax.set_title('Histogram for laser light intensity')
    ax.set_xlabel('X [mm]')
    ax.set_ylabel('Y [mm]')
    fig.colorbar(H[3],ax=ax)
```

```
    plt.show()

def gaussianBeam(self,w=0.54):
    #TEM00 のビームを生成します
    r = np.linspace(-w*2,w*2,100)
    #Ir = 2*np.exp(-2*r**2/(w**2))/(np.pi*(w**2))
    Ir = np.exp(-2*r**2/(w**2))
    normd = stats.norm(0, w/2)
    x = normd.rvs(self.nPh)
    y = normd.rvs(self.nPh)
    #z = np.zeros(self.nPh)

    fig, ax1 = plt.subplots()
    ax1.set_title('Input laser light distribution')
    ax1.hist(x, bins=100, color="C0")
    ax1.set_ylabel('Number of photon')
    ax2 = ax1.twinx()
    ax2.plot(r, Ir, color="k")
    ax2.set_xlabel('X [mm]')
    ax2.set_ylabel('Probability density')
    plt.show()
    self._get_beam_dist(x,y)
    return x,y

def get_result(self):
    encoded_position = self._encooder(self.p_result,self.add_result)
    df_result = {
        'p':encoded_position,
        'v':self.v_result,
        'w':self.w_result,
        'nPh':self.nPh
    }
    return df_result

def get_model_params(self):
    return self.model.get_params()

def _encooder(self,p,add):
    space = self.model.voxel_space
    center_add_x = self._get_center_add(self.model.voxel_model.shape[0])
    center_add_y = self._get_center_add(self.model.voxel_model.shape[1])
    encoded_position = p.copy()
```

```
    encoded_position[0] = space*(add[0]-center_add_x)+p[0]
    encoded_position[1] = space*(add[1]-center_add_y)+p[1]
    encoded_position[2] = np.round(space*(add[2]-1)+p[2]+space/2,6)
    return encoded_position

def set_monte_params(self,*,nPh,model, dtype_f=np.float32, dtype=np.int32,w_beam = 0):
    self.dtype_f = dtype_f
    self.dtype = dtype
    self.nPh = nPh
    self.w_beam = w_beam
    self.model = model

def build(self,*initial_data, **kwargs):
    if initial_data == () and kwargs == {}:
        pass
    else:
        self.model.set_params(*initial_data, **kwargs)
    self.model.build()

def getRdTtRate(self):
    self.Tt_index = np.where(self.v_result[2]>0)[0]
    self.Rd_index = np.where(self.v_result[2]<0)[0]
    self.Rdw = self.w_result[self.Rd_index].sum()/self.nPh
    self.Ttw = self.w_result[self.Tt_index].sum()/self.nPh
    print('######')
    print('Mean Rd %0.6f'% self.Rdw)
    print('Mean Td %0.6f'% self.Ttw)
    print()

def save_result(self,fname,coment=''):
    start_ = time.time()

    res = self.get_result()
    save_name = fname+"_LID.pkl.bz2"
    with bz2.open(save_name, 'wb') as fp:
        fp.write(pickle.dumps(res))
    print("Monte Carlo results saved in ")
    print("-> %s" %(save_name))
    print('')
    info = self._calc_info(coment)
    save_name = fname+"_info.json"
    with open(save_name, 'w') as fp:
        json.dump(info,fp,indent=4,cls= ToJsonEncoder)
```

```
        print("Calculation conditions are saved in")
        print("-> %s" %(save_name))
        print('')

        calTime(time.time(), start_)

    def _calc_info(self,coment=''):
        _params = self.model.get_params()
        calc_info = {
            'Date':datetime.datetime.now().isoformat(),
            'coment':coment,
            'number_of_photons':self.nPh,
            'calc_dtype':"32 bit",
            'model':{
                'model_name':self.model.model_name,
                'model_params':_params,
                'model_voxel_space':self.model.voxel_space,
                'model_xy_size':self.model.xy_size,
            },
            'w_beam':self.w_beam,
            'beam_angle':self.beam_angle,
            'initial_refrect_mode':self.initial_refrect_by_angle,
            'beam_mode':'TEM00',
            'fluence_mode':self.fluence_mode,
        }
        return calc_info
# ======================================================================
# Modeling class
# ======================================================================
class VoxelModel:
    def build(self):
        pass
    def set_params(self):
        pass

    def getModelSize(self):
        print("Memory area size for voxel storage: %0.3f Mbyte" % (self.voxel_model.nbytes*1e-6))

class PlateModel(VoxelModel):
    @_deprecate_positional_args
    def __init__(self):
        self.model_name = 'PlateModel'
        self.dtype_f = np.float32
```

```
    self.dtype = np.int8
    self.params = {
      'x_size':40,'y_size':40,#X-Y は intralipid の領域を示す
      'voxel_space':0.1,
      'thickness':[1.3,40],
      'n':[1.5,1.4],
      'n_air':1.,
      'ma':[1e-5,0.02374],
      'ms':[1e-5,0.02374],
      'g':[0.9,0.9],
      }
    self.keys = list(self.params.keys())
    self._param_instantiating()
    self.voxel_model = np.zeros((3,3,3),dtype = self.dtype)

def _param_instantiating(self):
    f = self.dtype_f
    self.thickness = self.params['thickness']
    self.n =np.array(self.params['n']+[self.params['n_air']]).astype(f)
    self.ms = np.array(self.params['ms']).astype(f)
    self.ma = np.array(self.params['ma']).astype(f)
    self.g = np.array(self.params['g']).astype(f)
    self.voxel_space = self.params['voxel_space']
    self.x_size = np.int32(round(self.params['x_size']/self.params['voxel_space']))
    self.y_size = np.int32(round(self.params['y_size']/self.params['voxel_space']))
    self.z_size = np.int32(round(np.array(self.thickness).sum()/self.params['voxel_space']))

def build(self):
    #thickness,xy_size,voxel_space,ma,ms,g,n,n_air
    del self.voxel_model
    gc.collect()
    self._make_voxel_model()
    self.getModelSize()

def set_params(self,*initial_data, **kwargs):
    set_params(self.params,self.keys,*initial_data, **kwargs)
    self._param_instantiating()

def _make_voxel_model(self):

    self.voxel_model = np.empty((
        self.x_size+2,
```

```
            self.y_size+2,
            self.z_size+2
        )).astype(self.dtype)

        val = 1
        for n_,i in enumerate(self.thickness):
            val_ = round(i/self.voxel_space)
            self.voxel_model[:,:,val:val_+val] = np.int8(n_)
            val+=val_

        self.end_point = np.int8(np.array(self.thickness).size)
        self.voxel_model[0,:,:] = self.end_point
        self.voxel_model[-1,:,:] = self.end_point
        self.voxel_model[:,0,:] = self.end_point
        self.voxel_model[:,-1,:] = self.end_point
        self.voxel_model[:,:,0] = self.end_point
        self.voxel_model[:,:,-1] = self.end_point

    def get_params(self):
        return {
                'th':self.thickness,
                'ms':self.ms,
                'ma':self.ma,
                'n':self.n,
                'g':self.g
                }

class PlateExModel(VoxelModel):
    #ガラスとイントラリピッドの２層構造のみを対象とする
    #ガラスはイントラリピッドを取り囲んでいるものとする
    def __init__(self):
        self.model_name = 'PlateExModel'
        self.dtype = 'int8'
        self.dtype_f = 'float32'
        self.grass_num=0
        self.intra_num=1
        self.end_point = 2
        self.params = {
            'x_size':40,'y_size':40,#X-Y は intralipid の領域を示す
            'voxel_space':0.1,
            'thickness':[1.3,40],
            'n':[1.5,1.4],
```

```
        'n_air':1.,
        'ma':[1e-5,0.02374],
        'ms':[1e-5,0.02374],
        'g':[0.9,0.9],
        }
    self.keys = list(self.params.keys())
    self._param_instantiating()
    self.voxel_model = np.zeros((3,3,3),dtype = self.dtype)
    self.model_shape = (3,3,3)

def _param_instantiating(self):
    f = self.dtype_f
    self.thickness = self.params['thickness']
    self.n =np.array(self.params['n']+[self.params['n_air']]).astype(f)
    self.ms = np.array(self.params['ms']).astype(f)
    self.ma = np.array(self.params['ma']).astype(f)
    self.g = np.array(self.params['g']).astype(f)
    self.voxel_space = self.params['voxel_space']
    self.x_size = int(self.params['x_size']/self.params['voxel_space'])
    self.y_size = int(self.params['y_size']/self.params['voxel_space'])
    self.z_size = int(self.params['thickness'][1]/self.params['voxel_space'])

def build(self):
    #thickness,xy_size,voxel_space,ma,ms,g,n,n_air
    del self.voxel_model
    gc.collect()
    self._make_voxel_model()
    self.getModelSize()

def set_params(self,*initial_data, **kwargs):
    set_params(self.params,self.keys,*initial_data, **kwargs)
    self._param_instantiating()

def _make_voxel_model(self):
    #まずイントラリピッドを定義し，その後，ガラス面を定義する
    self.voxel_model = np.ones((self.x_size,self.y_size,self.z_size),dtype = self.dtype)
    self.num_pix = int(self.params['thickness'][0]/self.params['voxel_space'])

    #z方向の追加
    ct = np.ones((
    self.voxel_model.shape[0],self.voxel_model.shape[1],self.num_pix+1),
    dtype = self.dtype)*self.grass_num
```

```
        self.voxel_model = np.concatenate((ct,self.voxel_model),2)
        self.voxel_model = np.concatenate((self.voxel_model,ct),2)
        #y 方向の追加
        ct = np.ones((
        self.voxel_model.shape[0],self.num_pix+1,self.voxel_model.shape[2]
        ),dtype = self.dtype)*self.grass_num
        self.voxel_model = np.concatenate((self.voxel_model,ct),1)
        self.voxel_model = np.concatenate((ct,self.voxel_model),1)
        #x 方向の追加
        ct = np.ones((
        self.num_pix+1,self.voxel_model.shape[1],self.voxel_model.shape[2]
        ),dtype = self.dtype)*self.grass_num
        self.voxel_model = np.concatenate((self.voxel_model,ct),0)
        self.voxel_model = np.concatenate((ct,self.voxel_model),0)

        # end cooding
        self.voxel_model[0,:,:] = self.end_point
        self.voxel_model[-1,:,:] = self.end_point
        self.voxel_model[:,0,:] = self.end_point
        self.voxel_model[:,-1,:] = self.end_point
        self.voxel_model[:,:,0] = self.end_point
        self.voxel_model[:,:,-1] = self.end_point

    def get_second_layer_addz(self):
        return self.num_pix+1

class TuringModel_Rectangular(VoxelModel):
    def __init__(self):
        self.model_name = 'TuringModel_Rectangular'
        self.dtype_f = np.float32
        self.dtype = np.int8
        self.ct_num=2
        self.subc_num=3
        self.skin_num=4
        self.end_point = 5
        self.voxel_model = np.zeros((3,3,3),dtype = self.dtype)

        self.params = {
            'xz_size':17.15,'voxel_space':0.0245,'dicom_path':False,'bv_tv':0.138,
            'th_cortical':1.,'th_subcutaneus':2.6,'th_dermis':1.4,
            'n_space':1.,'n_trabecular':1.4,'n_cortical':1.4,'n_subcutaneus':1.4,'n_dermis':1.4,'n_air':1.,
            'ma_space':1e-8,'ma_trabecular':0.02374,'ma_cortical':0.02374,'ma_subcutaneus':0.011,'ma_dermis':0.037,
```

```
        'ms_space':1e-8,'ms_trabecular':20.54,'ms_cortical':17.67,'ms_subcutaneus':20,'ms_dermis':20,
        'g_space':0.90,'g_trabecular':0.90,'g_cortical':0.90,'g_subcutaneus':0.90,'g_dermis':.90,
        }
    self.keys = list(self.params.keys())
    self._make_model_params()
    self.voxel_space = self.params['voxel_space']

def build(self,bone_model):
    #thickness,xy_size,voxel_space,ma,ms,g,n,n_air
    del self.voxel_model
    gc.collect()
    self.voxel_model = bone_model

    self.voxel_space = self.params['voxel_space']
    self._make_voxel_model()
    self.getModelSize()

def set_params(self,*initial_data, **kwargs):
    set_params(self.params,self.keys,*initial_data, **kwargs)
    self._make_model_params()

def _make_model_params(self):
    # パラメーターは，[骨梁間隙, 海綿骨, 緻密骨, 皮下組織, 皮膚, 外気]のように設定されています.
    name_list = ['_space','_trabecular','_cortical','_subcutaneus','_dermis']
    _n = []; _ma = []; _ms = []; _g = []
    for i in name_list:
        _n.append(self.params['n'+i])
        _ma.append(self.params['ma'+i])
        _ms.append(self.params['ms'+i])
        _g.append(self.params['g'+i])
    _n.append(self.params['n_air'])
    self.n = np.array(_n).astype(self.dtype_f)
    self.ma = np.array(_ma).astype(self.dtype_f)
    self.ms = np.array(_ms).astype(self.dtype_f)
    self.g = np.array(_g).astype(self.dtype_f)

def get_params(self):
    return {
            'ms':self.ms,
            'ma':self.ma,
            'n':self.n,
            'g':self.g
```

```
        }
def _read_dicom(self):
    path = self.params['dicom_path']
    files = os.listdir(path)
    files.sort()
    self.params['voxel_space'] = round(float(pydicom.dcmread(path+"/"+files[0],force=True).PixelSpacing[0]),5)

    ds = []
    for i in files:
        ds.append(pydicom.dcmread(path+"/"+i,force=True).pixel_array)
    ds = np.array(ds).astype("int8")
    return ds

def add_array(self,X,num_pix,val,dtype,y_axis = False):
    # Z 方向
    ct = np.zeros((X.shape[0],X.shape[1],num_pix),dtype = dtype)+val
    X = np.concatenate((ct,X),2)
    X = np.concatenate((X,ct),2)
    # X 方向
    ct = np.zeros((num_pix,X.shape[1],X.shape[2]),dtype = dtype)+val
    X = np.concatenate((ct,X),0)
    X = np.concatenate((X,ct),0)
    # Y 方向
    if y_axis:
        ct = np.zeros((X.shape[0],num_pix,X.shape[2]),dtype = dtype)+val
        X = np.concatenate((ct,X),1)
        X = np.concatenate((X,ct),1)

    return X

def _make_voxel_model(self):
    if self.params['dicom_path']:
        self.voxel_model = self._read_dicom()
    A = np.zeros_like(self.voxel_model).astype(bool)
    list_num = [self.ct_num,self.subc_num,self.skin_num]
    num_s = np.round(np.array(
    [self.params['th_cortical'],self.params['th_subcutaneus'],self.params['th_dermis']]
    )/self.params["voxel_space"]).astype(np.int)

    int_num = int(self.voxel_model.shape[0]/2-round(self.params["xz_size"]/(self.params["voxel_space"]*2)))+num_s[0]
    A[int_num:-int_num,:,int_num:-int_num] = 1
```

```
        x=0
        for i in A[:,int(A.shape[2]/2),int(A.shape[0]/2)]:
            if i:
                break
            x+=1
        A = A[x:-x,:,x:-x]
        self.voxel_model = self.voxel_model[x:-x,:,x:-x]

        for i in tqdm(range(3)):
            self.voxel_model = self.add_array(self.voxel_model,num_s[i],list_num[i],np.int8)

        self.voxel_model = self.add_array(self.voxel_model,1,self.end_point,np.int8,y_axis=True)
        print("Shape of voxel_model ->",self.voxel_model.shape)

class TuringModel_Cylinder(TuringModel_Rectangular):
    def __init__(self):
        self.model_name = 'TuringModel_Cylinder'
        self.dtype_f = np.float32
        self.dtype = np.int8
        self.ct_num=2
        self.subc_num=3
        self.skin_num=4
        self.air_num=5
        self.end_point = 6
        self.voxel_model = np.zeros((3,3,3),dtype = self.dtype)

        self.params = {
            'r_bone':9.14,'voxel_space':0.0245,'dicom_path':False,'bv_tv':0.138,
            'th_cortical':1.,'th_subcutaneus':2.6,'th_dermis':1.4,
            'n_space':1.,'n_trabecular':1.4,'n_cortical':1.4,'n_subcutaneus':1.4,'n_dermis':1.4,'n_air':1.,
            'ma_space':1e-
8,'ma_trabecular':0.02374,'ma_cortical':0.02374,'ma_subcutaneus':0.011,'ma_dermis':0.037,'ma_air':1e-5,
            'ms_space':1e-8,'ms_trabecular':20.54,'ms_cortical':17.67,'ms_subcutaneus':20,'ms_dermis':20,'ms_air':1e-5,
            'g_space':0.90,'g_trabecular':0.90,'g_cortical':0.90,'g_subcutaneus':0.90,'g_dermis':.90,'g_air':.90,
            }
        self.keys = list(self.params.keys())
        self._make_model_params()
        self.voxel_space = self.params['voxel_space']

    def _make_model_params(self):
        # パラメーターは，[骨梁間隙, 海綿骨, 緻密骨, 皮下組織, 皮膚, 外気]のように設定されています.
        name_list = ['_space','_trabecular','_cortical','_subcutaneus','_dermis','_air']
        _n = [];_ma = [];_ms = [];_g = []
```

```
    for i in name_list:
        _n.append(self.params['n'+i])
        _ma.append(self.params['ma'+i])
        _ms.append(self.params['ms'+i])
        _g.append(self.params['g'+i])
    _n.append(self.params['n_air'])
    self.n = np.array(_n).astype(self.dtype_f)
    self.ma = np.array(_ma).astype(self.dtype_f)
    self.ms = np.array(_ms).astype(self.dtype_f)
    self.g = np.array(_g).astype(self.dtype_f)

def round_index(self,X,num_r):
    size_x = int(X.shape[0]/2)
    size_y = int(X.shape[1])
    size_z = int(X.shape[2]/2)
    x_lab = np.tile(np.arange(X.shape[0]),(X.shape[2], 1)).T-size_x
    y_lab = np.tile(np.arange(X.shape[2]),(X.shape[0], 1))-size_z
    r_ = np.sqrt(x_lab**2+y_lab**2)
    return np.where(r_<num_r)

def _make_voxel_model(self):
    #骨梁間隙を 0,海綿骨を 1，緻密骨を 2， 皮下組織を 3，皮膚を 4，大気を 5，領域外を-1 に設定する
    if self.params['dicom_path']:
        self.voxel_model = self._read_dicom()
    A = np.zeros_like(self.voxel_model).astype(bool)
    list_num = [self.ct_num,self.subc_num,self.skin_num]
    num_s = np.round(np.array(
    [self.params['th_cortical'],self.params['th_subcutaneus'],self.params['th_dermis']]
    )/self.params["voxel_space"]).astype(np.int)

    num_tr = round(self.params["r_bone"]/self.params["voxel_space"])-num_s[0]
    ind = self.round_index(A,num_tr)
    for i in range(A.shape[1]):
        A[ind[0],i,ind[1]] = 1
    ind = np.where(A==0)
    self.voxel_model[ind] = self.air_num
    x=0
    for i in A[:,int(A.shape[2]/2),int(A.shape[0]/2)]:
        if i:
            break
        x+=1
    A = A[x:-x,:,x:-x]
```

```
        self.voxel_model = self.voxel_model[x:-x,:,x:-x]

        for i in tqdm(range(3)):
            B = self.add_array(A,num_s[i],False,bool)
            self.voxel_model = self.add_array(self.voxel_model,num_s[i],self.air_num,np.int8)
            num_tr+=num_s[i]
            ind = self.round_index(B,num_tr)
            A = np.zeros_like(B).astype(bool)
            for j in range(A.shape[1]):
                A[ind[0],j,ind[1]] = 1
            ind = np.where((A&~B)==1)
            self.voxel_model[ind] = list_num[i]

        self.voxel_model = self.add_array(self.voxel_model,1,self.end_point,np.int8,y_axis=True)
        print("Shape of voxel_model ->",self.voxel_model.shape)

class TuringModel_RnC(TuringModel_Cylinder):
    def __init__(self):
        self.model_name = 'TuringModel_RnC'
        self.dtype_f = np.float32
        self.dtype = np.int8
        self.ct_num=2
        self.subc_num=3
        self.skin_num=4
        self.air_num=5
        self.end_point = 6
        self.voxel_model = np.zeros((3,3,3),dtype = self.dtype)
        self.params = {
            'r_bone':9.14,'xz_size':17.15,'voxel_space':0.0245,'dicom_path':False,'bv_tv':0.138,
            'th_cortical':1.,'th_subcutaneus':2.6,'th_dermis':1.4,
            'n_space':1.,'n_trabecular':1.4,'n_cortical':1.4,'n_subcutaneus':1.4,'n_dermis':1.4,'n_air':1.,
            'ma_space':1e-
8,'ma_trabecular':0.02374,'ma_cortical':0.02374,'ma_subcutaneus':0.011,'ma_dermis':0.037,'ma_air':1e-5,
            'ms_space':1e-8,'ms_trabecular':20.54,'ms_cortical':17.67,'ms_subcutaneus':20,'ms_dermis':20,'ms_air':1e-5,
            'g_space':0.90,'g_trabecular':0.90,'g_cortical':0.90,'g_subcutaneus':0.90,'g_dermis':.90,'g_air':.90,
            }
        self.keys = list(self.params.keys())
        self._make_model_params()
        self.voxel_space = self.params['voxel_space']

    def _make_voxel_model(self):
        #骨梁間隙を 0,海綿骨を 1，緻密骨を 2，皮下組織を 3，皮膚を 4，大気を 5，領域外を-1 に設定する
        if self.params['dicom_path']:
```

```
        self.voxel_model = self._read_dicom()
    A = np.zeros_like(self.voxel_model).astype(bool)
    B = np.zeros_like(A).astype(bool)

    list_num = [self.ct_num,self.subc_num,self.skin_num]
    num_s = np.round(np.array(
    [self.params['th_cortical'],self.params['th_subcutaneus'],self.params['th_dermis']]
    )/self.params["voxel_space"]).astype(np.int)

    int_num = int(self.voxel_model.shape[0]/2-round(self.params["xz_size"]/(self.params["voxel_space"]*2)))+num_s[0]
    A[int_num:-int_num,:,int_num:-int_num] = 1

    num_tr = round(self.params["r_bone"]/self.params["voxel_space"])-num_s[0]
    ind = self.round_index(B,num_tr)
    for i in range(B.shape[1]):
        B[ind[0],i,ind[1]] = 1

    A = A&B
    ind = np.where(A==0)
    self.voxel_model[ind] = self.air_num
    x=0
    for i in A[:,int(A.shape[2]/2),int(A.shape[0]/2)]:
        if i:
            break
        x+=1
    A = A[x:-x,:,x:-x]
    self.voxel_model = self.voxel_model[x:-x,:,x:-x]

    for i in tqdm(range(3)):
        B = self.add_array(A,num_s[i],False,bool)
        self.voxel_model = self.add_array(self.voxel_model,num_s[i],self.air_num,np.int8)
        num_tr+=num_s[i]
        ind = self.round_index(B,num_tr)
        A = np.zeros_like(B).astype(bool)
        for j in range(A.shape[1]):
            A[ind[0],j,ind[1]] = 1
        ind = np.where((A&~B)==1)
        self.voxel_model[ind] = list_num[i]

    self.voxel_model = self.add_array(self.voxel_model,1,self.end_point,np.int8,y_axis=True)
    print("Shape of voxel_model ->",self.voxel_model.shape)

# ================================================================================
```

```
# Public montecalro model
# ========================================================================

class VoxelPlateModel(BaseVoxelMonteCarlo):
    def __init__(
        self,*,nPh=1000,dtype_f=np.float32,dtype=np.int32,
        beam_type = 'TEM00',w_beam = 0,
        beam_angle = 0,initial_refrect_by_angle = False,
        first_layer_clear = False,
    ):
        super().__init__(
            nPh = nPh, model = PlateModel(),dtype_f=dtype_f,dtype=dtype,
            w_beam=w_beam,beam_angle = beam_angle,beam_type = beam_type,
            initial_refrect_by_angle = initial_refrect_by_angle,
            first_layer_clear=first_layer_clear,
        )

class VoxelPlateExModel(BaseVoxelMonteCarlo):
    #ガラスとイントラリピッドの２層構造のみを対象とする
    #ガラスはイントラリピッドを取り囲んでいるものとする
    def __init__(
        self,*,nPh=1000,dtype_f=np.float32,dtype=np.int32,
        beam_type = 'TEM00',w_beam = 0,
        beam_angle = 0,initial_refrect_by_angle = False,
        first_layer_clear = True,
        ):
        super().__init__(
            nPh = nPh, model = PlateExModel(),dtype_f=dtype_f,dtype=dtype,
            w_beam=w_beam,beam_angle = beam_angle,beam_type = beam_type,
            initial_refrect_by_angle = initial_refrect_by_angle,
            first_layer_clear=first_layer_clear,
        )



    def build(self,*initial_data, **kwargs):
        if initial_data == () and kwargs == {}:
            pass
        else:
            self.model.set_params(*initial_data, **kwargs)
        self.model.build()

    def set_params(self,*initial_data, **kwargs):
```

```
        self.model.set_params(*initial_data, **kwargs)

    def _calc_info(self,coment=''):
        calc_info = {
            'Date':datetime.datetime.now().isoformat(),
            'coment':coment,
            'number_of_photons':self.nPh,
            'calc_dtype':self.dtype,
            'model':{
                'model_name':self.model.model_name,
                'model_params':self.model.params,
            },
            'w_beam':self.w_beam,
            'beam_angle':self.beam_angle,
            'initial_refrect_mode':self.initial_refrect_by_angle,
            'beam_mode':'TEM00',
            'fluence_mode':self.fluence_mode,
        }
        if self.beam_angle_mode:
            calc_info['wavelength'] = self.wavelength
            calc_info['beam_posision'] = self.beam_posision
            calc_info['lens_curvature_radius'] = self.lens_curvature_radius
            calc_info['grass_type'] = self.grass_type
        return calc_info

class VoxelTuringModel(BaseVoxelMonteCarlo):
    def __init__(
        self,*,nPh=1000,dtype_f=np.float32,dtype=np.int32,
        beam_type = 'TEM00',w_beam = 0,
        beam_angle = 0,initial_refrect_by_angle = False,
        first_layer_clear = False,
        model_name = 'TuringModel'
        ):
        self.namelist = ['TuringModel_Rectangular','TuringModel_Cylinder','TuringModel_RnC']
        if model_name==self.namelist[0]:
            model = TuringModel_Rectangular()
        elif model_name == self.namelist[1]:
            model = TuringModel_Cylinder()
        elif model_name == self.namelist[2]:
            model = TuringModel_RnC()
        else:
            print('Invalid name: ',model_name)
```

```
        super().__init__(
            nPh = nPh, model = model,dtype_f=dtype_f,dtype=dtype,
            w_beam=w_beam,beam_angle = beam_angle,beam_type = beam_type,
            initial_refrect_by_angle = initial_refrect_by_angle,
            first_layer_clear=first_layer_clear,
        )
        self.bone_model = False

    def _set_inital_add(self):

        if self.beam_type == 'TEM00':
            self.add =  np.zeros((3, self.nPh),dtype = self.dtype)
        self.add[0] = self._get_center_add(self.model.voxel_model.shape[0])
        self.add[1] = self._get_center_add(self.model.voxel_model.shape[1])
        if self.first_layer_clear:
            self.add[2] = self.model.get_second_layer_addz()
        else:
            self.add[2] = 1

        if self.model.model_name==self.namelist[1]:
            def _get_first_num_z(a,x):
                if a[x]==(self.model.end_point-2):
                    return x
                return _get_first_num_z(a,x+1)
            aa = self.add[:,0]
            a = self.model.voxel_model[aa[0],aa[1]]
            x=0
            zz = _get_first_num_z(a,x)
            print("Inital add for z-axis is ",zz)
            self.add[2] = zz

    def build(self,*initial_data, **kwargs):
        if initial_data == () and kwargs == {}:
            pass
        else:
            self.model.set_params(*initial_data, **kwargs)
        self.model.build(self.bone_model)
        del self.bone_model
        gc.collect()

    def set_model(self,u):
        self.bone_model = u
```

```
def set_params(self,*initial_data, **kwargs):
    self.model.set_params(*initial_data, **kwargs)

def get_model_fig(self,*,dpi=300,save_path = [False,False],):
    image = self.model.voxel_model
    resol0 = (image.shape[0]+1)*self.model.params['voxel_space']/2-\
    np.array([self.model.params['voxel_space']*i for i in range(image.shape[0]+1)])
    resol1 = (image.shape[1]+1)*self.model.params['voxel_space']/2-\
    np.array([self.model.params['voxel_space']*i for i in range(image.shape[1]+1)])
    resol2 = np.array([self.model.params['voxel_space']*i for i in range(image.shape[2]+1)])

    plt.figure(figsize=(5,5),dpi=100)
    plt.set_cmap(plt.get_cmap('gray'))
    plt.pcolormesh(resol0,resol2,image[:,int(image.shape[1]/2),:].T)
    plt.xlabel('X [mm]')
    plt.ylabel('Z [mm]')
    plt.ylim(resol2[-1],resol2[0])
    if save_path[0]:
        plt.savefig(
            save_path[0],
            dpi=dpi,
            orientation='portrait',
            transparent=False,
            pad_inches=0.0)
    plt.show()

    plt.figure(figsize=(6,5),dpi=100)
    plt.set_cmap(plt.get_cmap('gray'))
    plt.pcolormesh(resol1,resol2,image[int(image.shape[0]/2),:,:].T)
    plt.xlabel('Y [mm]')
    plt.ylabel('Z [mm]')
    plt.ylim(resol2[-1],resol2[0])
    if save_path[1]:
        plt.savefig(
            save_path[1],
            dpi=dpi,
            orientation='portrait',
            transparent=False,
            pad_inches=0.0)
    plt.show()

def _calc_info(self,coment=''):
    calc_info = {
```

```
            'Date':datetime.datetime.now().isoformat(),
            'coment':coment,
            'number_of_photons':self.nPh,
            'calc_dtype':"32 bit",
            'model':{
                'model_name':self.model.model_name,
                'model_params':self.model.params,
            },
            'w_beam':self.w_beam,
            'beam_angle':self.beam_angle,
            'initial_refrect_mode':self.initial_refrect_by_angle,
            'beam_mode':'TEM00',
        }
        return calc_info
```

pyMonteOpt/ Pymopt/ voxel/_kernel.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 22 18:14:23 2022
@author: Kaname Miura
"""
import numpy as np
from tqdm import tqdm
from numba import njit


@njit(fastmath=True)
def theta(g,rand):
    th = 0
    if g != 0.:
        th = (1 + g**2 - ((1 - g**2) / (1 - g + 2 * g * rand))** 2) / (2 * g)
        if th < -1:
            th = -1
    else:
        th = 2 * rand - 1
    return th

@njit(fastmath=True)
def vmc_kernel(
```

```
add, p,v, w, ma, ms, n, g,
voxel_model, l,
nPh, end_point
):
ma = ma.astype(np.float32)
ms = ms.astype(np.float32)
n = n.astype(np.float32)
g = g.astype(np.float32)
nPh = np.int32(nPh)
count = 0
counter = np.int32(nPh/10)
for idx in range(nPh):
    add_ = np.array([0,0,0]).astype(np.int32)
    v_ = np.array([0,0,0]).astype(np.float32)
    zero_vec = np.array([0,0,0]).astype(np.int32)
    one_vec = np.array([1,1,1]).astype(np.int32)

    wth = np.float32(0.0001)
    roulette_m = np.int32(10)
    index_ = voxel_model[add[0,idx], add[1,idx], add[2,idx]]
    index_next = np.int8(0)
    index_end = np.int8(end_point)

    fi = np.float32(0);
    cos_fi = np.float32(0); cos_th = np.float32(0)
    sin_fi = np.float32(0); sin_th = np.float32(0);

    valf = np.float32(0.); dbnum = np.int32(0); db = np.float32(1000.)

    ni = np.float32(n[index_])
    nt = np.float32(0)
    mt = np.float32(ma[index_] + ms[index_])
    st = np.float32(0)
    flag_tr = True
    flag_end = False
    while True:
        st = np.float32(-np.log(np.random.rand()))
        while True:
            valf = 0.; dbnum = 0; db = 1000.
            for i in range(3):
                if (np.abs(v[i,idx]) > 0):
                    valf = (l / 2 - np.sign(v[i,idx]) * p[i,idx])/ np.abs(v[i,idx])
                    if (valf < db):
```

```
            dbnum = i; db = valf
    if st>= db*mt:
        for i in range(3):
            p[i,idx]+=v[i,idx]*db
        p[dbnum,idx] = l/2*np.sign(v[dbnum,idx])
        st-=db*mt
        for i in range(3):
            add_[i] = add[i,idx]
        add_[dbnum] += np.sign(v[dbnum,idx])
        index_next = voxel_model[add_[0], add_[1], add_[2]]

        nt = n[index_next]
        flag_tr = True

        if (ni != nt):
            ra = 0;at = 0
            ai = np.arccos(abs(v[dbnum,idx]))
            if (ni > nt)&(ai >= np.arcsin(nt/ni)):
                ra = 1
            else:
                at = np.arcsin((ni/nt)*np.sin(ai))
                if ai!=0:
                    ra = ((np.sin(ai - at) / np.sin(ai + at))**2 \
                    +(np.tan(ai - at) / np.tan(ai + at))**2)/2
                else:
                    ra = 0
            rand_ = np.random.rand()

            if ra < rand_:
                flag_tr = True
                zero_vec[dbnum] = 1; one_vec[dbnum] = 0
                for i in range(3):
                    v[i,idx] = one_vec[i]*v[i,idx]*ni/nt\
                    +zero_vec[i] * np.sign(v[i,idx]) * np.cos(at)
                    zero_vec[i] = 0; one_vec[i] = 1

                valf = np.sqrt(v[0,idx]**2+v[1,idx]**2+v[2,idx]**2)
                for i in range(3):
                    v[i,idx] /= valf
            else:
                flag_tr = False

        if flag_tr:
```

```
            add[dbnum,idx] += np.sign(v[dbnum,idx])
            p[dbnum,idx] *= -1
            index_ = index_next
            if index_ == index_end:
                flag_end = True
                break
            mt = ma[index_] + ms[index_]
            ni = nt
        else:
            v[dbnum,idx] *= -1
    else:
        for i in range(3):
            p[i,idx]+=v[i,idx]*st/mt
        st = 0
        break

if flag_end:
    break

w[idx] -= np.float32(w[idx]*ma[index_]/mt)

if(w[idx]<=wth):
    if (1/roulette_m) < np.float32(np.random.rand()):
        for i in range(3):
            p[i,idx] = 0
            v[i,idx] = 0
            add[i,idx] = 0
        w[idx] = 0
        break
    else:
        w[idx]*=roulette_m

cos_th = theta(g[index_], np.float32(np.random.rand()))
sin_th = np.sqrt(1-cos_th**2)

fi = 2 * 3.1415927*np.float32(np.random.rand())
cos_fi = np.cos(fi)
sin_fi = np.sin(fi)

if 0.99999 < abs(v[2,idx]):
    v[0,idx] = sin_th * cos_fi
    v[1,idx] = sin_th * sin_fi
    v[2,idx] = np.sign(v[2,idx]) * cos_th
```

```
            else:
                valf = np.sqrt(1 - v[2,idx]**2)
                for i in range(3):
                    v_[i] = v[i,idx]
                v[0,idx] = sin_th * (v_[0] * v_[2] * cos_fi - v_[1] * sin_fi) / valf + v_[0] * cos_th
                v[1,idx] = sin_th * (v_[1] * v_[2] * cos_fi + v_[0] * sin_fi) / valf + v_[1] * cos_th
                v[2,idx] = -sin_th * cos_fi * valf + v_[2] * cos_th

            valf = np.sqrt(v[0,idx]**2+v[1,idx]**2+v[2,idx]**2)
            for i in range(3):
                v[i,idx] /= valf

        flag_end = False
        if idx%counter == 0:
            count+=10
            print(count, " %")

    return add, p,v, w
```

A1.2.1 voxel_gpu

pyMonteOpt/ Pymopt/ voxel_gpu/__init__.py

```
from ._cukernel import vmc_kernel
from ._classes import VoxelPlateModel
from ._classes import VoxelTuringModel

__all__ = [

'vmc_kernel',
'VoxelPlateModel',
'VoxelTuringModel',
]
```

pyMonteOpt/ Pymopt/ voxel_gpu /_classes.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 24 16:56:05 2021
@author: Kaname Miura
"""
import cupy as cp
from ._cukernel import vmc_kernel
from tqdm import tqdm
import numpy as np
import os
import pydicom
from scipy import stats
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import colors
from abc import ABCMeta, abstractmethod

import datetime,time
import json,pickle,bz2

from ..utils import readDicom,reConstArray_8,reConstArray
from ..utils.validation import _deprecate_positional_args
from ..fluence import Fluence2D,Fluence3D
from ..utils.utilities import calTime,set_params,ToJsonEncoder
from ..optics._classes import Grass
import gc
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
#os.system("taskset -p 0xff %d" % os.getpid())

func = vmc_kernel()
__all__ = [
'VoxelPlateModel','VoxelTuringModel'
]

# ======================================================================
# Base solid model
# ======================================================================
```

```
class BaseVoxelMonteCarlo(metaclass = ABCMeta):
    #@_deprecate_positional_args
    @abstractmethod
    def __init__(self,*,nPh,model,dtype_f=np.float32,dtype=np.int32,
                 beam_type = 'TEM00',w_beam = 0,
                 beam_angle = 0,initial_refrect_by_angle = False,
                 first_layer_clear = False,
                 threadnum = 128,
                 ):
        super().__init__()

        def __check_list_name(name,name_list):
            if not(name in name_list):
                raise ValueError('%s is not a permitted for factor. Please choose from %s.'%(name,name_list))

        self.beam_type_list=['TEM00',False]
        __check_list_name(beam_type,self.beam_type_list)
        self.beam_type = beam_type

        self.dtype = dtype
        self.dtype_f = dtype_f
        self.nPh = nPh
        self.w_beam = w_beam

        self.initial_refrect_by_angle = initial_refrect_by_angle
        self.beam_angle = beam_angle

        self.model = model
        self.first_layer_clear=first_layer_clear
        if threadnum > 1024:
            raise Exception('There are too many threads. threadnum < 1024.')
        else:
            self.threadnum = int(threadnum)

    def start(self,rand_seed=None):
        if rand_seed is None:
            rand_seed = np.random.randint(1e5)
        self.nPh = int(self.nPh)
        self._reset_results()
        self._generate_initial_coodinate(self.nPh)

        M = np.int32(self.model.voxel_model.shape[1])
        L = np.int32(self.model.voxel_model.shape[2])
```

```
print("")
print("###### Start (Random seed: %s) ######" %rand_seed)
print("")
start_ = time.time()
cp.get_default_memory_pool().free_all_blocks()
cp.get_default_pinned_memory_pool().free_all_blocks()

add_ = cp.asarray(self.add.astype(np.int32),dtype = np.int32)
p_ = cp.asarray(self.p.astype(np.float32),dtype = np.float32)
v_ = cp.asarray(self.v.astype(np.float32),dtype = np.float32)
w_ = cp.asarray(self.w.astype(np.float32),dtype = np.float32)
ma_ = cp.asarray(self.model.ma.astype(np.float32))
ms_ = cp.asarray(self.model.ms.astype(np.float32))
n_ = cp.asarray(self.model.n.astype(np.float32))
g_ = cp.asarray(self.model.g.astype(np.float32))
v_model = cp.asarray(self.model.voxel_model.astype(np.int8),dtype = np.int8)
l_ = cp.float32(self.model.voxel_space)
nph = cp.int32(self.nPh)
end_p = cp.int8(self.model.end_point)

func((int((self.nPh+self.threadnum-1)/self.threadnum),1), (self.threadnum,1),
(
   add_,p_,v_,w_,
   ma_,ms_,n_,g_,
   v_model,l_,
   M,L,nph,end_p,np.int32(rand_seed)
))

self.add = cp.asnumpy(add_)
self.p = cp.asnumpy(p_)
self.v = cp.asnumpy(v_)
self.w = cp.asnumpy(w_)

del add_,p_,v_,w_,ma_,ms_,n_,g_,
del v_model,l_,M,L,nph,end_p,rand_seed,
cp.get_default_memory_pool().free_all_blocks()
cp.get_default_pinned_memory_pool().free_all_blocks()
gc.collect()

self._end_process()
print("###### End ######")
self.getRdTtRate()
```

```
    calTime(time.time(), start_)

    return self

def _end_process(self):#書き換え
    #index = np.where(~np.isnan(self.w))[0]
    self.v_result = self.v#[:,index]
    self.p_result = self.p#[:,index]
    self.add_result = self.add#[:,index]
    self.w_result = self.w#[index]

def _reset_results(self):
    self.v_result = np.empty((3,1)).astype(self.dtype_f)
    self.p_result = np.empty((3,1)).astype(self.dtype_f)
    self.add_result = np.empty((3,1)).astype(self.dtype)
    self.w_result = np.empty(1).astype(self.dtype_f)
    return self

def get_voxel_model(self):
    return self.model.voxel_model

def _generate_initial_coodinate(self,nPh):
    self._set_inital_add()
    self._set_beam_distribution()
    self._set_inital_vector()
    self._set_inital_w()


def _set_inital_add(self):
    if self.beam_type == 'TEM00':
        self.add =  np.zeros((3, self.nPh),dtype = self.dtype)
    self.add[0] = self._get_center_add(self.model.voxel_model.shape[0])
    self.add[1] = self._get_center_add(self.model.voxel_model.shape[1])
    if self.first_layer_clear:
        self.add[2] = self.model.get_second_layer_addz()
    else:
        self.add[2] = 1

def _get_center_add(self,length):
    #add の中心がローカル座標（ボックス内）の中心となるため,
    #ボクセル数が偶数の時は，1/2 小さい位置を中心とし光を照射し,
    #逆変換時（_encooder）も同様に 1/2 小さい位置を中心として元のマクロな座標に戻す.
```

```
    return int((length-1)/2)

def _set_inital_vector(self):
    if self.beam_type == 'TEM00':
        self.v = np.zeros((3,self.nPh)).astype(self.dtype_f)
        self.v[2] = 1
        if self.beam_angle!=0 and self.w_beam==0:
            #ビーム径がある場合はとりあえず無視
            #角度は rad 表記
            ni = self.model.n[-1]
            nt = self.model.n[0]
            ai = self.beam_angle
            at = np.arcsin(np.sin(ai)*ni/nt)
            self.v[0] = np.sin(at)
            self.v[2] = np.cos(at)
            if self.initial_refrect_by_angle:
                Ra = ((np.sin(ai-at)/np.sin(ai+at))**2\
                +(np.tan(ai-at)/np.tan(ai+at))**2)/2

                self.inital_del_num = np.count_nonzero(Ra>=np.random.rand(self.nPh))
                self.v = np.delete(self.v, np.arange(self.inital_del_num), 1)
                self.p = np.delete(self.p, np.arange(self.inital_del_num), 1)
                self.add = np.delete(self.add, np.arange(self.inital_del_num), 1)
                sub_v = np.zeros((3,self.inital_del_num)).astype(self.dtype_f)
                sub_v[0] = np.sin(ai)
                sub_v[2] = -np.cos(ai)
                self.v_result = np.concatenate([self.v_result,
                sub_v],axis = 1)
                self.p_result = np.concatenate([self.p_result,
                self.p[:,:self.inital_del_num]],axis = 1)
                self.add_result = np.concatenate([self.add_result,
                self.add[:,:self.inital_del_num]],axis = 1)
    else:
        print("ビームタイプが設定されていません")

def _set_inital_w(self):
    if self.beam_type == 'TEM00':
        self.w = np.ones(self.nPh).astype(self.dtype_f)
        Rsp = 0
        n1 = self.model.n[-1]
        n2 = self.model.n[0]
        if n1 != n2:
```

```
            Rsp = ((n1-n2)/(n1+n2))**2
            if self.beam_angle!=0 and self.w_beam==0:
                ai = self.beam_angle
                at = np.arcsin(np.sin(ai)*n1/n2)
                Rsp = ((np.sin(ai-at)/np.sin(ai+at))**2\
                +(np.tan(ai-at)/np.tan(ai+at))**2)/2
            elif self.first_layer_clear:
                n3=self.model.n[1]
                r2 = ((n3-n2)/(n3+n2))**2
                Rsp = Rsp+r2*(1-Rsp)**2/(1-Rsp*r2)
            self.w -= Rsp

        if self.beam_angle!=0 and self.w_beam==0:
            if self.initial_refrect_by_angle:
                self.w[:] = 1
                self.w = np.delete(self.w, np.arange(self.inital_del_num), 0)
                self.w_result = np.concatenate([self.w_result,
                self.w[:self.inital_del_num]],axis = 0)
    else:
        print("ビームタイプが設定されていません")

def _set_beam_distribution(self):
    if self.beam_type == 'TEM00':
        self.p = np.zeros((3,self.nPh)).astype(self.dtype_f)
        self.p[2] = -self.model.voxel_space/2
        if self.w_beam!= 0:
            print("%s を入力"%self.beam_type)
            #ガウシアン分布を生成
            gb = np.array(self.gaussianBeam(self.w_beam)).astype(self.dtype_f)
            #ガウシアン分布を各アドレスに振り分ける

            l = self.model.voxel_space
            pp = (gb/l).astype("int16")
            ind = np.where(gb<0)
            pp[ind[0].tolist(),ind[1].tolist()] -= 1
            pa = gb - (pp+1/2)*l
            ind = np.where((np.abs(pa)>=l/2))
            pa[ind[0].tolist(),ind[1].tolist()] = \
                np.sign(pa[ind[0].tolist(),ind[1].tolist()])*(l/2)
            pa += l/2
            self.add[:2] = self.add[:2] + pp
            self.p[:2] = pa.astype(self.dtype_f)
```

```
    else:
        print("ビームタイプが設定されていません")

def _get_beam_dist(self,x,y):
    fig = plt.figure(figsize=(10,6),dpi=70)
    ax = fig.add_subplot(111)
    ax.set_aspect('equal')
    H = ax.hist2d(x,y, bins=100,cmap="plasma")
    ax.set_title('Histogram for laser light intensity')
    ax.set_xlabel('X [mm]')
    ax.set_ylabel('Y [mm]')
    fig.colorbar(H[3],ax=ax)
    plt.show()

def gaussianBeam(self,w=0.54):
    #TEM00 のビームを生成します
    r = np.linspace(-w*2,w*2,100)
    #Ir = 2*np.exp(-2*r**2/(w**2))/(np.pi*(w**2))
    Ir = np.exp(-2*r**2/(w**2))
    normd = stats.norm(0, w/2)
    x = normd.rvs(self.nPh)
    y = normd.rvs(self.nPh)
    #z = np.zeros(self.nPh)

    fig, ax1 = plt.subplots()
    ax1.set_title('Input laser light distribution')
    ax1.hist(x, bins=100, color="C0")
    ax1.set_ylabel('Number of photon')
    ax2 = ax1.twinx()
    ax2.plot(r, Ir, color="k")
    ax2.set_xlabel('X [mm]')
    ax2.set_ylabel('Probability density')
    plt.show()
    self._get_beam_dist(x,y)
    return x,y

def get_result(self):
    encoded_position = self._encooder(self.p_result,self.add_result)
    df_result = {
        'p':encoded_position,
        'v':self.v_result,
        'w':self.w_result,
```

```
        'nPh':self.nPh
    }
    return df_result

def get_model_params(self):
    return self.model.get_params()

def _encooder(self,p,add):
    space = self.model.voxel_space
    center_add_x = self._get_center_add(self.model.voxel_model.shape[0])
    center_add_y = self._get_center_add(self.model.voxel_model.shape[1])
    encoded_position = p.copy()
    encoded_position[0] = space*(add[0]-center_add_x)+p[0]
    encoded_position[1] = space*(add[1]-center_add_y)+p[1]
    encoded_position[2] = np.round(space*(add[2]-1)+p[2]+space/2,6)
    return encoded_position

def set_monte_params(self,*,nPh,model, dtype_f=np.float32,dtype=np.int32,w_beam = 0):
    self.dtype = dtype
    self.dtype_f = dtype_f
    self.nPh = nPh
    self.w_beam = w_beam
    self.model = model

def build(self,*initial_data, **kwargs):
    if initial_data == () and kwargs == {}:
        pass
    else:
        self.model.set_params(*initial_data, **kwargs)
    self.model.build()

def getRdTtRate(self):
    self.Tt_index = np.where(self.v_result[2]>0)[0]
    self.Rd_index = np.where(self.v_result[2]<0)[0]
    self.Rdw = self.w_result[self.Rd_index].sum()/self.nPh
    self.Ttw = self.w_result[self.Tt_index].sum()/self.nPh
    print('######')
    print('Mean Rd %0.6f'% self.Rdw)
    print('Mean Td %0.6f'% self.Ttw)
    print()

def save_result(self,fname,coment=''):
    start_ = time.time()
```

```
        res = self.get_result()
        save_name = fname+"_LID.pkl.bz2"
        with bz2.open(save_name, 'wb') as fp:
            fp.write(pickle.dumps(res))
        print("Monte Carlo results saved in ")
        print("-> %s" %(save_name))
        print('')
        info = self._calc_info(coment)
        save_name = fname+"_info.json"
        with open(save_name, 'w') as fp:
            json.dump(info,fp,indent=4,cls= ToJsonEncoder)
        print("Calculation conditions are saved in")
        print("-> %s" %(save_name))
        print('')

        calTime(time.time(), start_)

    def _calc_info(self,coment=''):
        _params = self.model.get_params()
        calc_info = {
            'Date':datetime.datetime.now().isoformat(),
            'coment':coment,
            'number_of_photons':self.nPh,
            'calc_dtype':"32 bit",
            'model':{
                'model_name':self.model.model_name,
                'model_params':_params,
                'model_voxel_space':self.model.voxel_space,
                'model_xy_size':self.model.xy_size,
            },
            'w_beam':self.w_beam,
            'beam_angle':self.beam_angle,
            'initial_refrect_mode':self.initial_refrect_by_angle,
            'beam_mode':'TEM00',
            'fluence_mode':self.fluence_mode,
        }
        return calc_info
# ======================================================================
# Modeling class
# ======================================================================

class VoxelModel:
```

```
    def build(self):
        pass
    def set_params(self):
        pass

    def getModelSize(self):
        print("Memory area size for voxel storage: %0.3f Mbyte" % (self.voxel_model.nbytes*1e-6))

class PlateModel(VoxelModel):
    @_deprecate_positional_args
    def __init__(self):
        self.model_name = 'PlateModel'
        self.dtype_f = np.float32
        self.dtype = np.int8
        self.params = {
            'x_size':40,'y_size':40,#X-Y は intralipid の領域を示す
            'voxel_space':0.1,
            'thickness':[1.3,40],
            'n':[1.5,1.4],
            'n_air':1.,
            'ma':[1e-5,0.02374],
            'ms':[1e-5,0.02374],
            'g':[0.9,0.9],
            }
        self.keys = list(self.params.keys())
        self._param_instantiating()
        self.voxel_model = np.zeros((3,3,3),dtype = self.dtype)

    def _param_instantiating(self):
        f = self.dtype_f
        self.thickness = self.params['thickness']
        self.n =np.array(self.params['n']+[self.params['n_air']]).astype(f)
        self.ms = np.array(self.params['ms']).astype(f)
        self.ma = np.array(self.params['ma']).astype(f)
        self.g = np.array(self.params['g']).astype(f)
        self.voxel_space = self.params['voxel_space']
        self.x_size = np.int32(round(self.params['x_size']/self.params['voxel_space']))
        self.y_size = np.int32(round(self.params['y_size']/self.params['voxel_space']))
        self.z_size = np.int32(round(np.array(self.thickness).sum()/self.params['voxel_space']))

    def build(self):
        #thickness,xy_size,voxel_space,ma,ms,g,n,n_air
```

```
        del self.voxel_model
        gc.collect()
        self._make_voxel_model()
        self.getModelSize()

    def set_params(self,*initial_data, **kwargs):
        set_params(self.params,self.keys,*initial_data, **kwargs)
        self._param_instantiating()

    def _make_voxel_model(self):

        self.voxel_model = np.empty((
            self.x_size+2,
            self.y_size+2,
            self.z_size+2
        )).astype(self.dtype)

        val = 1
        for n_,i in enumerate(self.thickness):
            val_ = round(i/self.voxel_space)
            self.voxel_model[:,:,val:val_+val] = np.int8(n_)
            val+=val_

        self.end_point = np.int8(np.array(self.thickness).size)
        self.voxel_model[0,:,:] = self.end_point
        self.voxel_model[-1,:,:] = self.end_point
        self.voxel_model[:,0,:] = self.end_point
        self.voxel_model[:,-1,:] = self.end_point
        self.voxel_model[:,:,0] = self.end_point
        self.voxel_model[:,:,-1] = self.end_point

    def get_params(self):
        return {
                'th':self.thickness,
                'ms':self.ms,
                'ma':self.ma,
                'n':self.n,
                'g':self.g
                }

class PlateExModel(VoxelModel):
    #ガラスとイントラリピッドの２層構造のみを対象とする
```

```
#ガラスはイントラリピッドを取り囲んでいるものとする
def __init__(self):
    self.model_name = 'PlateExModel'
    self.dtype = 'int8'
    self.dtype_f = 'float32'
    self.grass_num=0
    self.intra_num=1
    self.end_point = 2
    self.params = {
        'x_size':40,'y_size':40,#X-Y は intralipid の領域を示す
        'voxel_space':0.1,
        'thickness':[1.3,40],
        'n':[1.5,1.4],
        'n_air':1.,
        'ma':[1e-5,0.02374],
        'ms':[1e-5,0.02374],
        'g':[0.9,0.9],
        }
    self.keys = list(self.params.keys())
    self._param_instantiating()
    self.voxel_model = np.zeros((3,3,3),dtype = self.dtype)
    self.model_shape = (3,3,3)

def _param_instantiating(self):
    f = self.dtype_f
    self.thickness = self.params['thickness']
    self.n =np.array(self.params['n']+[self.params['n_air']]).astype(f)
    self.ms = np.array(self.params['ms']).astype(f)
    self.ma = np.array(self.params['ma']).astype(f)
    self.g = np.array(self.params['g']).astype(f)
    self.voxel_space = self.params['voxel_space']
    self.x_size = int(self.params['x_size']/self.params['voxel_space'])
    self.y_size = int(self.params['y_size']/self.params['voxel_space'])
    self.z_size = int(self.params['thickness'][1]/self.params['voxel_space'])

def build(self):
    #thickness,xy_size,voxel_space,ma,ms,g,n,n_air
    del self.voxel_model
    gc.collect()
    self._make_voxel_model()
    self.getModelSize()
```

```
    def set_params(self,*initial_data, **kwargs):
        set_params(self.params,self.keys,*initial_data, **kwargs)
        self._param_instantiating()

    def _make_voxel_model(self):
        #まずイントラリピッドを定義し，その後，ガラス面を定義する
        self.voxel_model = np.ones((self.x_size,self.y_size,self.z_size),dtype = self.dtype)
        self.num_pix = int(self.params['thickness'][0]/self.params['voxel_space'])

        #z 方向の追加
        ct = np.ones((
        self.voxel_model.shape[0],self.voxel_model.shape[1],self.num_pix+1),
        dtype = self.dtype)*self.grass_num
        self.voxel_model = np.concatenate((ct,self.voxel_model),2)
        self.voxel_model = np.concatenate((self.voxel_model,ct),2)
        #y 方向の追加
        ct = np.ones((
        self.voxel_model.shape[0],self.num_pix+1,self.voxel_model.shape[2]
        ),dtype = self.dtype)*self.grass_num
        self.voxel_model = np.concatenate((self.voxel_model,ct),1)
        self.voxel_model = np.concatenate((ct,self.voxel_model),1)
        #x 方向の追加
        ct = np.ones((
        self.num_pix+1,self.voxel_model.shape[1],self.voxel_model.shape[2]
        ),dtype = self.dtype)*self.grass_num
        self.voxel_model = np.concatenate((self.voxel_model,ct),0)
        self.voxel_model = np.concatenate((ct,self.voxel_model),0)

        # end cooding
        self.voxel_model[0,:,:] = self.end_point
        self.voxel_model[-1,:,:] = self.end_point
        self.voxel_model[:,0,:] = self.end_point
        self.voxel_model[:,-1,:] = self.end_point
        self.voxel_model[:,:,0] = self.end_point
        self.voxel_model[:,:,-1] = self.end_point

    def get_second_layer_addz(self):
        return self.num_pix+1


class TuringModel_Rectangular(VoxelModel):
    def __init__(self):
```

```
    self.model_name = 'TuringModel_Rectangular'
    self.dtype_f = np.float32
    self.dtype = np.int8
    self.ct_num=2
    self.subc_num=3
    self.skin_num=4
    self.end_point = 5
    self.voxel_model = np.zeros((3,3,3),dtype = self.dtype)

    self.params = {
        'xz_size':17.15,'voxel_space':0.0245,'dicom_path':False,'bv_tv':0.138,
        'th_cortical':1.,'th_subcutaneus':2.6,'th_dermis':1.4,
        'n_space':1.,'n_trabecular':1.4,'n_cortical':1.4,'n_subcutaneus':1.4,'n_dermis':1.4,'n_air':1.,
        'ma_space':1e-8,'ma_trabecular':0.02374,'ma_cortical':0.02374,'ma_subcutaneus':0.011,'ma_dermis':0.037,
        'ms_space':1e-8,'ms_trabecular':20.54,'ms_cortical':17.67,'ms_subcutaneus':20,'ms_dermis':20,
        'g_space':0.90,'g_trabecular':0.90,'g_cortical':0.90,'g_subcutaneus':0.90,'g_dermis':.90,
        }
    self.keys = list(self.params.keys())
    self._make_model_params()
    self.voxel_space = self.params['voxel_space']

def build(self,bone_model):
    #thickness,xy_size,voxel_space,ma,ms,g,n,n_air
    del self.voxel_model
    gc.collect()
    self.voxel_model = bone_model

    self.voxel_space = self.params['voxel_space']
    self._make_voxel_model()
    self.getModelSize()

def set_params(self,*initial_data, **kwargs):
    set_params(self.params,self.keys,*initial_data, **kwargs)
    self._make_model_params()

def _make_model_params(self):
    # パラメーターは，[骨梁間隙, 海綿骨, 緻密骨, 皮下組織, 皮膚, 外気]のように設定されています.
    name_list = ['_space','_trabecular','_cortical','_subcutaneus','_dermis']
    _n = [];_ma = [];_ms = [];_g = []
    for i in name_list:
        _n.append(self.params['n'+i])
        _ma.append(self.params['ma'+i])
```

```
        _ms.append(self.params['ms'+i])
        _g.append(self.params['g'+i])
    _n.append(self.params['n_air'])
    self.n = np.array(_n).astype(self.dtype_f)
    self.ma = np.array(_ma).astype(self.dtype_f)
    self.ms = np.array(_ms).astype(self.dtype_f)
    self.g = np.array(_g).astype(self.dtype_f)

def get_params(self):
    return {
            'ms':self.ms,
            'ma':self.ma,
            'n':self.n,
            'g':self.g
            }
def _read_dicom(self):
    path = self.params['dicom_path']
    files = os.listdir(path)
    files.sort()
    self.params['voxel_space'] = round(float(pydicom.dcmread(path+"/"+files[0],force=True).PixelSpacing[0]),5)

    ds = []
    for i in files:
        ds.append(pydicom.dcmread(path+"/"+i,force=True).pixel_array)
    ds = np.array(ds).astype("int8")
    return ds

def add_array(self,X,num_pix,val,dtype,y_axis = False):
    # Z 方向
    ct = np.zeros((X.shape[0],X.shape[1],num_pix),dtype = dtype)+val
    X = np.concatenate((ct,X),2)
    X = np.concatenate((X,ct),2)
    # X 方向
    ct = np.zeros((num_pix,X.shape[1],X.shape[2]),dtype = dtype)+val
    X = np.concatenate((ct,X),0)
    X = np.concatenate((X,ct),0)
    # Y 方向
    if y_axis:
        ct = np.zeros((X.shape[0],num_pix,X.shape[2]),dtype = dtype)+val
        X = np.concatenate((ct,X),1)
        X = np.concatenate((X,ct),1)
```

```
        return X

    def _make_voxel_model(self):
        if self.params['dicom_path']:
            self.voxel_model = self._read_dicom()
        A = np.zeros_like(self.voxel_model).astype(bool)
        list_num = [self.ct_num,self.subc_num,self.skin_num]
        num_s = np.round(np.array(
        [self.params['th_cortical'],self.params['th_subcutaneus'],self.params['th_dermis']]
        )/self.params["voxel_space"]).astype(np.int)

        int_num = int(self.voxel_model.shape[0]/2-round(self.params["xz_size"]/(self.params["voxel_space"]*2)))+num_s[0]
        A[int_num:-int_num,:,int_num:-int_num] = 1

        x=0
        for i in A[:,int(A.shape[2]/2),int(A.shape[0]/2)]:
            if i:
                break
            x+=1
        A = A[x:-x,:,x:-x]
        self.voxel_model = self.voxel_model[x:-x,:,x:-x]

        for i in tqdm(range(3)):
            self.voxel_model = self.add_array(self.voxel_model,num_s[i],list_num[i],np.int8)

        self.voxel_model = self.add_array(self.voxel_model,1,self.end_point,np.int8,y_axis=True)
        print("Shape of voxel_model ->",self.voxel_model.shape)


class TuringModel_Cylinder(TuringModel_Rectangular):
    def __init__(self):
        self.model_name = 'TuringModel_Cylinder'
        self.dtype_f = np.float32
        self.dtype = np.int8
        self.ct_num=2
        self.subc_num=3
        self.skin_num=4
        self.air_num=5
        self.end_point = 6
        self.voxel_model = np.zeros((3,3,3),dtype = self.dtype)

        self.params = {
            'r_bone':9.14,'voxel_space':0.0245,'dicom_path':False,'bv_tv':0.138,
```

```
        'th_cortical':1.,'th_subcutaneus':2.6,'th_dermis':1.4,
        'n_space':1.,'n_trabecular':1.4,'n_cortical':1.4,'n_subcutaneus':1.4,'n_dermis':1.4,'n_air':1.,
        'ma_space':1e-
8,'ma_trabecular':0.02374,'ma_cortical':0.02374,'ma_subcutaneus':0.011,'ma_dermis':0.037,'ma_air':1e-5,
        'ms_space':1e-8,'ms_trabecular':20.54,'ms_cortical':17.67,'ms_subcutaneus':20,'ms_dermis':20,'ms_air':1e-5,
        'g_space':0.90,'g_trabecular':0.90,'g_cortical':0.90,'g_subcutaneus':0.90,'g_dermis':.90,'g_air':.90,
        }
    self.keys = list(self.params.keys())
    self._make_model_params()
    self.voxel_space = self.params['voxel_space']

  def _make_model_params(self):
    # パラメーターは，[骨梁間隙, 海綿骨, 緻密骨, 皮下組織, 皮膚, 外気]のように設定されています.
    name_list = ['_space','_trabecular','_cortical','_subcutaneus','_dermis','_air']
    _n = [];_ma = [];_ms = [];_g = []
    for i in name_list:
      _n.append(self.params['n'+i])
      _ma.append(self.params['ma'+i])
      _ms.append(self.params['ms'+i])
      _g.append(self.params['g'+i])
    _n.append(self.params['n_air'])
    self.n = np.array(_n).astype(self.dtype_f)
    self.ma = np.array(_ma).astype(self.dtype_f)
    self.ms = np.array(_ms).astype(self.dtype_f)
    self.g = np.array(_g).astype(self.dtype_f)

  def round_index(self,X,num_r):
    size_x = int(X.shape[0]/2)
    size_y = int(X.shape[1])
    size_z = int(X.shape[2]/2)
    x_lab = np.tile(np.arange(X.shape[0]),(X.shape[2], 1)).T-size_x
    y_lab = np.tile(np.arange(X.shape[2]),(X.shape[0], 1))-size_z
    r_ = np.sqrt(x_lab**2+y_lab**2)
    return np.where(r_<num_r)

  def _make_voxel_model(self):
    #骨梁間隙を 0,海綿骨を 1，緻密骨を 2， 皮下組織を 3，皮膚を 4，大気を 5，領域外を-1 に設定する
    if self.params['dicom_path']:
      self.voxel_model = self._read_dicom()
    A = np.zeros_like(self.voxel_model).astype(bool)
    list_num = [self.ct_num,self.subc_num,self.skin_num]
    num_s = np.round(np.array(
    [self.params['th_cortical'],self.params['th_subcutaneus'],self.params['th_dermis']]
```

```
        )/self.params["voxel_space"]).astype(np.int)

        num_tr = round(self.params["r_bone"]/self.params["voxel_space"])-num_s[0]
        ind = self.round_index(A,num_tr)
        for i in range(A.shape[1]):
            A[ind[0],i,ind[1]] = 1
        ind = np.where(A==0)
        self.voxel_model[ind] = self.air_num
        x=0
        for i in A[:,int(A.shape[2]/2),int(A.shape[0]/2)]:
            if i:
                break
            x+=1
        A = A[x:-x,:,x:-x]
        self.voxel_model = self.voxel_model[x:-x,:,x:-x]

        for i in tqdm(range(3)):
            B = self.add_array(A,num_s[i],False,bool)
            self.voxel_model = self.add_array(self.voxel_model,num_s[i],self.air_num,np.int8)
            num_tr+=num_s[i]
            ind = self.round_index(B,num_tr)
            A = np.zeros_like(B).astype(bool)
            for j in range(A.shape[1]):
                A[ind[0],j,ind[1]] = 1
            ind = np.where((A&~B)==1)
            self.voxel_model[ind] = list_num[i]

        self.voxel_model = self.add_array(self.voxel_model,1,self.end_point,np.int8,y_axis=True)
        print("Shape of voxel_model ->",self.voxel_model.shape)

class TuringModel_RnC(TuringModel_Cylinder):
    def __init__(self):
        self.model_name = 'TuringModel_RnC'
        self.dtype_f = np.float32
        self.dtype = np.int8
        self.ct_num=2
        self.subc_num=3
        self.skin_num=4
        self.air_num=5
        self.end_point = 6
        self.voxel_model = np.zeros((3,3,3),dtype = self.dtype)
        self.params = {
            'r_bone':9.14,'xz_size':17.15,'voxel_space':0.0245,'dicom_path':False,'bv_tv':0.138,
```

```
        'th_cortical':1.,'th_subcutaneus':2.6,'th_dermis':1.4,
        'n_space':1.,'n_trabecular':1.4,'n_cortical':1.4,'n_subcutaneus':1.4,'n_dermis':1.4,'n_air':1.,
        'ma_space':1e-
8,'ma_trabecular':0.02374,'ma_cortical':0.02374,'ma_subcutaneus':0.011,'ma_dermis':0.037,'ma_air':1e-5,
        'ms_space':1e-8,'ms_trabecular':20.54,'ms_cortical':17.67,'ms_subcutaneus':20,'ms_dermis':20,'ms_air':1e-5,
        'g_space':0.90,'g_trabecular':0.90,'g_cortical':0.90,'g_subcutaneus':0.90,'g_dermis':.90,'g_air':.90,
        }
    self.keys = list(self.params.keys())
    self._make_model_params()
    self.voxel_space = self.params['voxel_space']


  def _make_voxel_model(self):
    #骨梁間隙を 0,海綿骨を 1，緻密骨を 2， 皮下組織を 3，皮膚を 4，大気を 5，領域外を-1 に設定する
    if self.params['dicom_path']:
      self.voxel_model = self._read_dicom()
    A = np.zeros_like(self.voxel_model).astype(bool)
    B = np.zeros_like(A).astype(bool)

    list_num = [self.ct_num,self.subc_num,self.skin_num]
    num_s = np.round(np.array(
    [self.params['th_cortical'],self.params['th_subcutaneus'],self.params['th_dermis']]
    )/self.params["voxel_space"]).astype(np.int)

    int_num = int(self.voxel_model.shape[0]/2-round(self.params["xz_size"]/(self.params["voxel_space"]*2)))+num_s[0]
    A[int_num:-int_num,:,int_num:-int_num] = 1

    num_tr = round(self.params["r_bone"]/self.params["voxel_space"])-num_s[0]
    ind = self.round_index(B,num_tr)
    for i in range(B.shape[1]):
      B[ind[0],i,ind[1]] = 1

    A = A&B
    ind = np.where(A==0)
    self.voxel_model[ind] = self.air_num
    x=0
    for i in A[:,int(A.shape[2]/2),int(A.shape[0]/2)]:
      if i:
        break
      x+=1
    A = A[x:-x,:,x:-x]
    self.voxel_model = self.voxel_model[x:-x,:,x:-x]

    for i in tqdm(range(3)):
```

```
            B = self.add_array(A,num_s[i],False,bool)
            self.voxel_model = self.add_array(self.voxel_model,num_s[i],self.air_num,np.int8)
            num_tr+=num_s[i]
            ind = self.round_index(B,num_tr)
            A = np.zeros_like(B).astype(bool)
            for j in range(A.shape[1]):
                A[ind[0],j,ind[1]] = 1
            ind = np.where((A&~B)==1)
            self.voxel_model[ind] = list_num[i]

        self.voxel_model = self.add_array(self.voxel_model,1,self.end_point,np.int8,y_axis=True)
        print("Shape of voxel_model ->",self.voxel_model.shape)

# ================================================================================
# Public montecalro model
# ================================================================================

class VoxelPlateModel(BaseVoxelMonteCarlo):
    def __init__(
        self,*,nPh=1000,dtype_f=np.float32,dtype=np.int32,
        beam_type = 'TEM00',w_beam = 0,
        beam_angle = 0,initial_refrect_by_angle = False,
        first_layer_clear = False,
        threadnum = 128,
    ):
        super().__init__(
            nPh = nPh, model = PlateModel(),dtype_f=dtype_f,dtype=dtype,
            w_beam=w_beam,beam_angle = beam_angle,beam_type = beam_type,
            initial_refrect_by_angle = initial_refrect_by_angle,
            first_layer_clear=first_layer_clear,
            threadnum = threadnum
        )

class VoxelPlateExModel(BaseVoxelMonteCarlo):
    #ガラスとイントラリピッドの２層構造のみを対象とする
    #ガラスはイントラリピッドを取り囲んでいるものとする
    def __init__(
        self,*,nPh=1000,dtype_f=np.float32,dtype=np.int32,
        beam_type = 'TEM00',w_beam = 0,
        beam_angle = 0,initial_refrect_by_angle = False,
        first_layer_clear = True,
        threadnum = 128,
```

```
        ):
        super().__init__(
            nPh = nPh, model = PlateModel(),dtype_f=dtype_f,dtype=dtype,
            w_beam=w_beam,beam_angle = beam_angle,beam_type = beam_type,
            initial_refrect_by_angle = initial_refrect_by_angle,
            first_layer_clear=first_layer_clear,
            threadnum = threadnum
        )

    def build(self,*initial_data, **kwargs):
        if initial_data == () and kwargs == {}:
            pass
        else:
            self.model.set_params(*initial_data, **kwargs)
        self.model.build()

    def set_params(self,*initial_data, **kwargs):
        self.model.set_params(*initial_data, **kwargs)

    def _calc_info(self,coment=''):
        calc_info = {
            'Date':datetime.datetime.now().isoformat(),
            'coment':coment,
            'number_of_photons':self.nPh,
            'calc_dtype':self.dtype,
            'model':{
                'model_name':self.model.model_name,
                'model_params':self.model.params,
            },
            'w_beam':self.w_beam,
            'beam_angle':self.beam_angle,
            'initial_refrect_mode':self.initial_refrect_by_angle,
            'beam_mode':'TEM00',
            'fluence_mode':self.fluence_mode,
        }
        if self.beam_angle_mode:
            calc_info['wavelength'] = self.wavelength
            calc_info['beam_posision'] = self.beam_posision
            calc_info['lens_curvature_radius'] = self.lens_curvature_radius
            calc_info['grass_type'] = self.grass_type
        return calc_info

class VoxelTuringModel(BaseVoxelMonteCarlo):
```

```
def __init__(
    self,*,nPh=1000,dtype_f=np.float32,dtype=np.int32,
    beam_type = 'TEM00',w_beam = 0,
    beam_angle = 0,initial_refrect_by_angle = False,
    first_layer_clear = False,
    threadnum = 128,
    model_name = 'TuringModel'
    ):
    self.namelist = ['TuringModel_Rectangular','TuringModel_Cylinder','TuringModel_RnC']
    if model_name==self.namelist[0]:
        model = TuringModel_Rectangular()
    elif model_name == self.namelist[1]:
        model = TuringModel_Cylinder()
    elif model_name == self.namelist[2]:
        model = TuringModel_RnC()
    else:
        print('Invalid name: ',model_name)

    super().__init__(
        nPh = nPh, model = model,dtype_f=dtype_f,dtype=dtype,
        w_beam=w_beam,beam_angle = beam_angle,beam_type = beam_type,
        initial_refrect_by_angle = initial_refrect_by_angle,
        first_layer_clear=first_layer_clear,
        threadnum = threadnum
    )
    self.bone_model = False

def _set_inital_add(self):
    if self.beam_type == 'TEM00':
        self.add =  np.zeros((3, self.nPh),dtype = self.dtype)
    self.add[0] = self._get_center_add(self.model.voxel_model.shape[0])
    self.add[1] = self._get_center_add(self.model.voxel_model.shape[1])
    if self.first_layer_clear:
        self.add[2] = self.model.get_second_layer_addz()
    else:
        self.add[2] = 1

    if self.model.model_name==self.namelist[1]:
        def _get_first_num_z(a,x):
            if a[x]==(self.model.end_point-2):
                return x
            return _get_first_num_z(a,x+1)
        aa = self.add[:,0]
```

```
        a = self.model.voxel_model[aa[0],aa[1]]
        x=0
        zz = _get_first_num_z(a,x)
        print("Inital add for z-axis is ",zz)
        self.add[2] = zz

def build(self,*initial_data, **kwargs):
    if initial_data == () and kwargs == {}:
        pass
    else:
        self.model.set_params(*initial_data, **kwargs)
    self.model.build(self.bone_model)
    del self.bone_model
    gc.collect()

def set_model(self,u):
    self.bone_model = u

def set_params(self,*initial_data, **kwargs):
    self.model.set_params(*initial_data, **kwargs)

def get_model_fig(self,*,dpi=300,save_path = [False,False],):
    image = self.model.voxel_model
    resol0 = (image.shape[0]+1)*self.model.params['voxel_space']/2-\
    np.array([self.model.params['voxel_space']*i for i in range(image.shape[0]+1)])
    resol1 = (image.shape[1]+1)*self.model.params['voxel_space']/2-\
    np.array([self.model.params['voxel_space']*i for i in range(image.shape[1]+1)])
    resol2 = np.array([self.model.params['voxel_space']*i for i in range(image.shape[2]+1)])

    plt.figure(figsize=(5,5),dpi=100)
    plt.set_cmap(plt.get_cmap('gray'))
    plt.pcolormesh(resol0,resol2,image[:,int(image.shape[1]/2),:].T)
    plt.xlabel('X [mm]')
    plt.ylabel('Z [mm]')
    plt.ylim(resol2[-1],resol2[0])
    if save_path[0]:
        plt.savefig(
            save_path[0],
            dpi=dpi,
            orientation='portrait',
            transparent=False,
            pad_inches=0.0)
    plt.show()
```

```
        plt.figure(figsize=(6,5),dpi=100)
        plt.set_cmap(plt.get_cmap('gray'))
        plt.pcolormesh(resol1,resol2,image[int(image.shape[0]/2),:,:].T)
        plt.xlabel('Y [mm]')
        plt.ylabel('Z [mm]')
        plt.ylim(resol2[-1],resol2[0])
        if save_path[1]:
            plt.savefig(
                save_path[1],
                dpi=dpi,
                orientation='portrait',
                transparent=False,
                pad_inches=0.0)
        plt.show()

    def _calc_info(self,coment=''):
        calc_info = {
            'Date':datetime.datetime.now().isoformat(),
            'coment':coment,
            'number_of_photons':self.nPh,
            'calc_dtype':"32 bit",
            'model':{
                'model_name':self.model.model_name,
                'model_params':self.model.params,
            },
            'w_beam':self.w_beam,
            'beam_angle':self.beam_angle,
            'initial_refrect_mode':self.initial_refrect_by_angle,
            'beam_mode':'TEM00',
        }
        return calc_info
```

pyMonteOpt/ Pymopt/ voxel_gpu /_cukernel.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 24 17:15:45 2021
@author: Kaname Miura
```

```
"""
import cupy as cp

def vmc_kernel():
    func = cp.RawKernel(r"""
    #include <curand_kernel.h>

    extern "C"{
        __device__ float theta(float g, float rand) {
            float th = 0;
            if (g != 0.) {
                float g2 = powf(g, 2);
                th = (1 + g2 - powf(((1 - g2) / (1 - g + 2 * g * rand)), 2)) / (2 * g);
                if (th < -1){th = -1;}
            }
            else {
                th = 2 * rand - 1;
            }
            return th;
        }

        __global__ void cuVMC(
            volatile int* add,
            volatile float* p,
            volatile float* v,
            volatile float* w,
            float* ma, float* ms, float* n, float* g,
            char* voxel_model, float l,
            int M, int L, int nPh, char end_point, int rand_seed
        ) {
            const int idx = threadIdx.x + blockDim.x * blockIdx.x;
            if (idx < nPh) {
                const int ix = idx;
                const int iy = idx + nPh;
                const int iz = idx + 2 * nPh;

                curandState s;
                curand_init(rand_seed, idx, 0, &s);

                // 変数系　計算用メモリ
                int add_[3] = {};
                float v_[3] = {};
```

```
float zero_vec[3] = {0,0,0}, one_vec[3] = { 1,1,1 };
float fi = 0;
float cos_fi, cos_th;
float sin_fi, sin_th;

const float wth = 0.0001;
const float roulette_m = 10;

char index = voxel_model[add[ix] * M * L + add[iy] * L + add[iz]];
char index_next = 0;
const char index_end = end_point;

float ni = n[index];
float nt = 0;
float mt = ma[index] + ms[index];

float st = 0;

float valf, db;
int dbnum, dbid;
float ai = 0, at = 0, ra = 0;

bool flag = 1;
//int step = 0;

while (flag) {
   // get photon's step size
   st = -logf(curand_uniform(&s));

   while (1) {
      // The distance between the current photon location
      // and the boundary of the current voxel
      valf = 0, dbnum = 0, dbid = 0, db = 1000;
      for (int i = 0; i < 3; i++) {
         if (fabsf(v[idx + nPh * i]) > 0) {
            valf = (1 / 2 - copysignf(1, v[idx + nPh * i]) * p[idx + nPh * i])
            / fabsf(v[idx + nPh * i]);
            if (valf < db) {
               dbnum = i, db = valf;
            }
         }
      }
      dbid = idx + nPh * dbnum;
```

```
if (st >= db * mt) {// 1 ステップが voxel 境界を越える場合
  // 光子を境界まで移動
  for (int i = 0; i < 3; i++) { p[idx + nPh * i] += v[idx + nPh * i] * db; }
  p[dbid] = copysignf(1 / 2, v[dbid]);// 計算誤差を補正
  st -= db * mt;

  // 透過先の光学特性 index を入手
  for (int i = 0; i < 3; i++) { add_[i] = add[idx + nPh * i]; }//add_の初期化
  add_[dbnum] += (int)copysignf(1, v[dbid]);
  index_next = voxel_model[add_[0] * M * L + add_[1] * L + add_[2]];

  // 透過先の屈折率を入手
  nt = n[index_next];
  if (ni != nt) {// 屈折率が変化した場合
    // 透過判別
    ai = acosf(fabsf(v[dbid])); // 入射角
    if ((ni > nt) && (ai >= asinf(nt / ni))) {// 全反射する場合
      ra = 1.;
    }
    else {// 全反射しない場合
      at = asinf((ni/nt) * sinf(ai));
      if(ai!=0){// 非常に重要
      // ai = 0 の場合このままだと ra は nan になる
      ra = (powf((sinf(ai - at) / sinf(ai + at)), 2)
      + powf((tanf(ai - at) / tanf(ai + at)), 2))/2;
      }
      else{
      ra = 0.;
      }
    }
    valf = curand_uniform(&s);

    if (ra < valf) { //透過
      // ベクトルを更新
      zero_vec[dbnum] = 1, one_vec[dbnum] = 0;
      for (int i = 0; i < 3; i++) {
        v[idx + nPh * i] = one_vec[i] * v[idx + nPh * i] * ni / nt
          + zero_vec[i] * copysignf(1, v[idx + nPh * i]) * cosf(at);
        zero_vec[i] = 0, one_vec[i] = 1;
```

```
        }
        valf = 0;
        for (int i = 0; i < 3; i++) { valf += powf(v[idx + nPh * i],2); }
        for (int i = 0; i < 3; i++) { v[idx + nPh * i] /= sqrtf(valf); }

        // Address を更新
        add[dbid] += (int)copysignf(1, v[dbid]);

        // Voxel 内の位置を更新
        p[dbid] *= -1;
        // 光学特性 index を更新
        index = index_next;
        if (index == index_end) {// 更新先が終端 voxel だった場合
          goto End;
        }
        // 光学特性を更新
        mt = ma[index] + ms[index];
        ni = nt;
      }
      else { // Fresnel 反射
        v[dbid] *= -1; //ベクトルの更新
      }
    }
    else {//透過するときの処理

      // Address を更新
      add[dbid] += (int)copysignf(1, v[dbid]);

      // Voxel 内の位置を更新
      p[dbid] *= -1;
      // 光学特性 index を更新
      index = index_next;
      if (index == index_end) {// 更新先が終端 voxel だった場合
        goto End;
      }
      // 光学特性を更新
      mt = ma[index] + ms[index];
    }
  }
  else {
    // Pthoton moving
```

```
        for (int i = 0; i < 3; i++) { p[idx + nPh * i] += (v[idx + nPh * i] * st / mt); }
        st = 0;
        break;
    }
}

// Photon absorption
w[idx] -= w[idx] * ma[index] / mt;

// serviving
if (w[idx] <= wth) {
    if ((1 / roulette_m) < curand_uniform(&s)) {
        for (int i = 0; i < 3; i++) {
            p[idx + nPh * i] = 0;
            v[idx + nPh * i] = 0;
            add[idx + nPh * i] = 0;
        }
        w[idx] = 0;
        goto End;
    }
    else {
        w[idx] *= roulette_m;
    }
}
// Photon scattering
cos_th = theta(g[index], curand_uniform(&s));
sin_th = sqrtf(1-powf(cos_th,2));

fi = 2 * 3.1415927*curand_uniform(&s);
cos_fi = cosf(fi);
sin_fi = sinf(fi);

if (0.99999 < fabsf(v[iz])) {
    v[ix] = sin_th * cos_fi;
    v[iy] = sin_th * sin_fi;
    v[iz] = copysignf(1, v[iz]) * cos_th;
}
else {
    valf = sqrtf(1 - v[iz]*v[iz]);
    v_[0] = sin_th * (v[ix] * v[iz] * cos_fi - v[iy] * sin_fi) / valf + v[ix] * cos_th;
    v_[1] = sin_th * (v[iy] * v[iz] * cos_fi + v[ix] * sin_fi) / valf + v[iy] * cos_th;
    v_[2] = -sin_th * cos_fi * valf + v[iz] * cos_th;
    for (int i = 0; i < 3; i++){ v[idx + nPh * i] = v_[i]; }
```

```
            }
            // 計算誤差の補正（単位ベクトルに変換）
            valf = 0;
            for (int i = 0; i < 3; i++) { valf += powf(v[idx + nPh * i],2); }
            for (int i = 0; i < 3; i++) { v[idx + nPh * i] /= sqrtf(valf); }
            /*
            if (step == 0){flag = 0;}
            step += 1;
            */
          }
        End:
        }
    }
}
  """, "cuVMC")
  return func
```

A1.3　ユーティリティモジュール

このモジュールには，次のファイルが含まれる．

param_generator.py: 合成生体組織のパラメータをランダムに決定するためのファイル．

readDICOM.py: 必要であれば DICOM ファイルを読み込むためのファイル．

utilities.py: 計算時間のロギング，各種パラメータセッティングの雛形とうが含まれる．

Validation.py: 検証用コードのファイル．

pyMonteOpt/ Pymopt/ util/__init__.py

```
# -*- coding: utf-8 -*-
from .validation import _deprecate_positional_args
from .readDICOM import readDicom,reConstArray_8,reConstArray
from .utilities import calTime,set_params,ToJsonEncoder,correlationLine
```

```
from .param_generator import generate_variable_params
__all__ = [
   '_deprecate_positional_args',
   'readDicom',
   'readDicom',
   'reConstArray_8',
   'reConstArray',
   'calTime','set_params',
   'ToJsonEncoder',
   'correlationLine',
   'generate_variable_params',
   ]
```

pyMonteOpt/ Pymopt/ util/param_generater.py

```
import numpy as np
from .utilities import set_params

__all__ = [
'generate_variable_params']

class generate_variable_params:
   def __init__(self):
      self.params = {
         'th_dermis':[1,2],            # 皮膚厚さの範囲
         'ma_dermis':[0.00633,0.08560], # 皮膚吸収係数の範囲
         'msd_dermis':[1.420,2.506],    # 皮膚減衰散乱係数の範囲
         'th_subcutaneus':[1,6],       # 皮下組織厚さの範囲
         'ma_subcutaneus':[0.005,0.012],# 皮下組織吸収係数の範囲
         'msd_subcutaneus':[0.83,1.396],# 皮下組織減衰散乱係数の範囲
         #'ma_marrow':[0.005,0.012],    # 骨髄吸収係数の範囲
         #'msd_marrow':[0.83,1.396],    # 骨髄減衰散乱係数の範囲
         'bv_tv':[0.115,0.02],         # 海綿骨 BV/TV の平均と分散
         'th_cortical':[0.669, 0.133], # 皮質骨厚さの平均と分散
         'corr':0.54,                  # 皮質骨と海綿骨の相関係数 Boutry2005
      }
      self.keys = list(self.params.keys())

   def set_params(self,*initial_data, **kwargs):
```

```
    set_params(self.params,self.keys,*initial_data, **kwargs)

def generate(self,n):
    drmis = self._get_dermis_params(n)
    subcut = self._get_subcut_params(n)
    bone = self._get_bone_params(n)
    #marrow = self._get_marrow_params(n)
    self.int_params ={
        'th_dermis':drmis[0],
        'ma_dermis':drmis[1],
        'msd_dermis':drmis[2],
        'th_subcutaneus':subcut[0],
        'ma_subcutaneus':subcut[1],
        'msd_subcutaneus':subcut[2],
        #'ma_marrow':marrow[0],
        #'msd_marrow':marrow[1],
        'bv_tv':bone[0],
        'th_cortical':bone[1],
    }
    return self.int_params

def get_variable_params(self):
    return self.int_params


def _uniform_dist(self,range_,n):
    return np.random.rand(n)*(range_[1]-range_[0])+range_[0]

def _get_dermis_params(self,n):
    th_ = self._uniform_dist(self.params['th_dermis'],n)
    ma_ = self._uniform_dist(self.params['ma_dermis'],n)
    msd_ = self._uniform_dist(self.params['msd_dermis'],n)
    return th_,ma_,msd_

def _get_subcut_params(self,n):
    th_ = self._uniform_dist(self.params['th_subcutaneus'],n)
    ma_ = self._uniform_dist(self.params['ma_subcutaneus'],n)
    msd_ = self._uniform_dist(self.params['msd_subcutaneus'],n)
    return th_,ma_,msd_

def _get_bone_params(self,n):
    mean = [self.params['bv_tv'][0],self.params['th_cortical'][0]] # [BV/TV, CTh]
    std = [self.params['bv_tv'][1], self.params['th_cortical'][1]] # [BV/TV, CTh]
```

```
        corr = self.params['corr']
        cov = [[std[0]**2, std[0]*std[1]*corr],
               [std[0]*std[1]*corr,std[1]**2]]

        bvtv, cth = np.random.multivariate_normal(mean, cov, n).T
        return bvtv, cth

    def _get_marrow_params(self,n):
        ma_ = self._uniform_dist(self.params['ma_marrow'],n)
        msd_ = self._uniform_dist(self.params['msd_marrow'],n)
        return ma_, msd_
```

pyMonteOpt/ Pymopt/ util/readDICOM.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jul 30 14:08:54 2019

@author: Kaname Miura
"""
#

import numpy as np
import pydicom as dicom
import os
import matplotlib.pyplot as plt

#workspace = os.getcwd()
#os.chdir(workspace)

def readDicom(path,*,size_down=True,ext_name = '.dcm'):
    #os.chdir(workspace)
    lstFilesDCM = []  # create an empty list
    for dirName, subdirList, fileList in os.walk(path):
        for filename in fileList:
            if ext_name in filename.lower(): # 拡張子が.dcm か.mag かで書き換える
                lstFilesDCM.append(os.path.join(dirName,filename))
    lstFilesDCM.sort()
    RefDs = dicom.read_file(lstFilesDCM[0],force=True) # DICOM の先頭ファイルはヘッダとなる
```

```
    RefDs.file_meta.TransferSyntaxUID = dicom.uid.ImplicitVRLittleEndian

    ConstPixelDims = (int(RefDs.Rows), int(RefDs.Columns), len(lstFilesDCM))

    ConstPixelSpacing = (float(RefDs.PixelSpacing[0]),
                         float(RefDs.PixelSpacing[1]),
                         float(RefDs.PixelSpacing[1]))

    ArrayDicom = np.zeros(ConstPixelDims, dtype=RefDs.pixel_array.dtype)

    # すべての DICOM ファイルに対して読み込む
    for filenameDCM in lstFilesDCM:
        ds = dicom.read_file(filenameDCM,force=True)
        ds.file_meta.TransferSyntaxUID = dicom.uid.ImplicitVRLittleEndian
        ArrayDicom[:, :, lstFilesDCM.index(filenameDCM)] = ds.pixel_array
    if size_down:
        ArrayDicom = _changeResolution(ArrayDicom,ConstPixelDims)

    print("ConstPixelDims: %s"%str(ConstPixelDims))
    print("ConstPixelSpacing: %s"%str(ConstPixelSpacing))
    print("Data infomation")
    print(RefDs)
    #os.chdir(workspace)
    return ArrayDicom,ConstPixelDims,ConstPixelSpacing

def _changeResolution(x,ConstPixelDims):
    #解像度を 16bit から 8bit に変更します.
    a = x.shape
    x = x.reshape(-1,ConstPixelDims[2])
    x = np.array([np.round(i/(2**8)).astype('int8') for i in x])
    return np.array(x).reshape(a[0],a[1],a[2])

def reConstArray_8(ArrayDicom, threshold=9500):
    threshold = round(threshold/(1+2**8))
    return np.where(ArrayDicom < threshold, 0, ArrayDicom)

def reConstArray(ArrayDicom, threshold=37):
    threshold = round(threshold)
    return np.where(ArrayDicom < threshold, 0, ArrayDicom)

def displayGraph(ArrayDicom,resolution):
    plt.figure(figsize=(6,6),dpi=100)
```

```
    plt.axes().set_aspect('equal', 'datalim')
    plt.set_cmap(plt.gray())
    plt.pcolormesh(resolution[1], resolution[0], ArrayDicom[:, :,0])
    plt.xlabel("mm")
    plt.ylabel("mm")
    plt.xlim(0,resolution[1].max())
    plt.show()

# 表示

#pathdcom = "DICOMfile9"
#ArrayDicom,resolution,ConstPixelDims = readDicom(pathdcom)
#ArrayDicom = reConstArray(ArrayDicom)
```

pyMonteOpt/ Pymopt/ util/utilities.py

```
import json
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import os


def calTime(end, start):
    elapsed_time = end - start
    q, mod = divmod(elapsed_time, 60)
    if q < 60:
        print('Calculation time: %d minutes %0.3f seconds.' % (q, mod))
    else:
        q2, mod2 = divmod(q, 60)
        print('Calculation time: %d h %0.3f minutes.' % (q2, mod2))

def check_folder(folder_dir):
    if not os.path.exists(folder_dir):
        os.makedirs(folder_dir)

def set_params(data,keys,*initial_data, **kwargs):
    for dictionary in initial_data:
        for key in dictionary:
            if not key in keys:
                raise KeyError(key)
```

```
            data[key] = dictionary[key]

    for key in kwargs:
        if not key in keys:
            raise KeyError(key)
        data[key] = kwargs[key]

class ToJsonEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, np.integer):
            return int(obj)
        elif isinstance(obj, np.floating):
            return float(obj)
        elif isinstance(obj, np.ndarray):
            return obj.tolist()
        else:
            return super(MyEncoder, self).default(obj)

def correlationLine(x,y):
    x = np.array(x).flatten()
    y = np.array(y).flatten()
    #X = np.linspace(x.min(), x.max(),int((x.max()-x.min())/10))

    #相関
    slope, intercept, r_value, _, _  = stats.linregress(x,y)
    r, p = stats.pearsonr(x,y)
    print(stats.spearmanr(x,y))
    p_str = ""
    if p >= 0.05:
        p_str = "p = " + str(round(p,3))
    elif p < 0.05 and p >= 0.01:
        p_str = "p < 5%"
    elif p < 0.01 and p >=0.005:
        p_str = "p < 1%"
    elif p < 0.005 and p >= 0.001:
        p_str = "p < 0.5%"
    elif p < 0.001 and 0.0001:
        p_str = "p < 0.1%"
    else:
        p_str = "p < 0.01%"

    label_ = "r = "+str(round(r_value,3)) + ", " + p_str
```

```
    print(label_)
    #print("p = %s"%p)
    ysub = np.poly1d(np.polyfit(x,y,1))(x)
    xx = [x.min(),x.max()]
    yy = [ysub.min(),ysub.max()]
    if r < 0:
        yy = [ysub.max(),ysub.min()]
    plt.plot(xx,yy,"--",color="0.2",label = label_)
```

pyMonteOpt/ Pymopt/ util/validation.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 10 15:45:55 2020

@author: kaname
"""

from functools import wraps
from inspect import signature, Parameter #,isclass
import warnings

def _deprecate_positional_args(f):
    """Decorator for methods that issues warnings for positional arguments.
    Using the keyword-only argument syntax in pep 3102, arguments after the
    * will issue a warning when passed as a positional argument.
    Parameters
    ----------
    f : callable
        Function to check arguments on.
    """
    sig = signature(f)
    kwonly_args = []
    all_args = []

    for name, param in sig.parameters.items():
        if param.kind == Parameter.POSITIONAL_OR_KEYWORD:
            all_args.append(name)
        elif param.kind == Parameter.KEYWORD_ONLY:
            kwonly_args.append(name)
```

```
    @wraps(f)
    def inner_f(*args, **kwargs):
        extra_args = len(args) - len(all_args)
        if extra_args <= 0:
            return f(*args, **kwargs)

        # extra_args > 0
        args_msg = ['{}={}'.format(name, arg)
                    for name, arg in zip(kwonly_args[:extra_args],
                                         args[-extra_args:])]
        warnings.warn("Pass {} as keyword args. From version 0.25 "
                      "passing these as positional arguments will "
                      "result in an error".format(", ".join(args_msg)),
                      FutureWarning)
        kwargs.update(zip(sig.parameters, args))
        return f(**kwargs)
    return inner_f
```

A1.4　シミュレーションモジュールの制御用ファイル

本論のデータは，virtualOBD.py ファイルを動かすことで得られた．また，virtualOBD.py は，vOBD.ipnb で制御した．

pyMonteOpt/virtualOBD.py

```
from pymopt.modeling_gpu import TuringPattern
from pymopt.voxel_gpu import VoxelTuringModel
from pymopt.utils import generate_variable_params
from pymopt import metrics as met

import datetime,time
import os,gc
import numpy as np
import pandas as pa
```

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("ticks", {'grid.linestyle': '--'})
import pandas as pd


import datetime,time
import os,gc
import numpy as np
import pandas as pa
from multiprocessing import Pool


repetitions = 1000
nPh = 1e7
iteral_num=np.arange(repetitions)

range_params_norm = {
   'th_dermis':[1,2],            # 皮膚厚さの範囲
   'ma_dermis':[0.00633,0.08560], # 皮膚吸収係数の範囲
   'msd_dermis':[1.420,2.506],   # 皮膚減衰散乱係数の範囲
   'th_subcutaneus':[1,6],       # 皮下組織厚さの範囲
   'ma_subcutaneus':[0.00485,0.01239],# 皮下組織吸収係数の範囲
   'msd_subcutaneus':[0.83,1.396],# 皮下組織減衰散乱係数の範囲
   'bv_tv':[0.134,0.028],        # 海綿骨 BV/TV の平均と分散
   'th_cortical':[0.804, 0.149], # 皮質骨厚さの平均と分散
   'corr':0.54,                  # 皮質骨と海綿骨の相関係数 Boutry2005
}
range_params_osteoporosis = range_params_norm.copy()
range_params_osteoporosis['bv_tv'] = [0.085,0.022]
range_params_osteoporosis['th_cortical'] = [0.487,0.138]
range_params_osteopenia = range_params_norm.copy()
range_params_osteopenia['bv_tv'] = [0.103,0.030]
range_params_osteopenia['th_cortical'] = [0.571,0.173]

def determining_params_range():
   a = np.random.rand()
   range_params = 0
   if a <= 0.4:
      print('Osteoporosis')
```

```
        range_params = range_params_osteoporosis
    elif a >= 0.6:
        print('Normal')
        range_params = range_params_norm
    else:
        print('Osteoprnia')
        range_params = range_params_osteopenia
    return range_params

model_params ={
    'grid':30,
    'dx':1/30,
    'dt':1,
    'du':0.0002,
    'dv':0.01,
    'length':10,
    'repetition':100,
    'voxelsize':0.0306,
    'seed':False,
    'ct_coef':4.5e4,
    'tile_num_xz':2,
    'tile_num_y':4,
}


#th_coef = np.array([-7.6618293,1.64450117,-0.45237661,0.60426539])
th_coef = np.array([-10.93021385,   2.62630274,  -0.50913966,   0.60371039])

monte_params = {
    'voxel_space':model_params['voxelsize'],
    'xz_size':17.15,
    'symmetrization':True,
    'enclosure':True,

    'n_space':1.4,
    'n_trabecular':1.55,
    'n_cortical':1.55,
    'n_subcutaneus':1.4,
    'n_dermis':1.4,
    'n_air':1.,

    'ma_trabecular':0.02374,
    'ma_cortical':0.02374,
```

```
    'ms_trabecular':20.588,
    'ms_cortical':20.588,

    'g_space':0.90,
    'g_trabecular':0.90,
    'g_cortical':0.90,
    'g_subcutaneus':0.90,
    'g_dermis':0.90,
}


def generate_bone_model(bv_tv,path,model_params):
    model_params['bv_tv']=bv_tv
    tp = TuringPattern()
    tp.set_params(model_params)
    tp.set_threshold_func_coef(th_coef)
    if not os.path.exists(path):
        os.makedirs(path)
    u = tp.modeling(path,save_dicom=False)
    bvtv_ = tp.bv_tv_real
    del tp
    gc.collect()
    return u,bvtv_

def calc_montecalro(vp,iteral,params,path,u):
    print()
    print('####################################')
    print('# %s'%iteral)

    nn = 0
    params['th_dermis'] = vp['th_dermis'][nn]
    params['ma_dermis'] = vp['ma_dermis'][nn]
    params['ms_dermis'] = vp['msd_dermis'][nn]/(1-params['g_dermis'])

    params['th_subcutaneus'] = vp['th_subcutaneus'][nn]
    params['ma_subcutaneus'] = vp['ma_subcutaneus'][nn]
    params['ms_subcutaneus'] = vp['msd_subcutaneus'][nn]/(1-params['g_subcutaneus'])

    params['ma_space'] = vp['ma_subcutaneus'][nn]
    params['ms_space'] = vp['msd_subcutaneus'][nn]/(1-params['g_space'])

    params['bv_tv'] = vp['bv_tv'][nn]
```

```
    params['th_cortical'] = vp['th_cortical'][nn]
    #print(params)
    model = VoxelTuringModel(
        nPh = nPh,
        model_name = 'TuringModel'
    )
    model.set_model(u)

    model.build(**params)
    start = time.time()
    model = model.start(iteral)
    print('%s sec'%(time.time()-start))
    print('# %s'%iteral)
    print("Save -> %s"%path)
    model.save_result(path,coment='for machine learning')
    res = model.get_result()

    del model
    gc.collect()
    return res,params

def calc_ray_tracing(res,monte_params,path,alias_name):
    l = monte_params["xz_size"]+monte_params["th_subcutaneus"]*2+monte_params["th_dermis"]*2
    nn = 300
    dr = 30/nn
    nn_ = 400
    dr_ = 40/nn_
    margin = 1e-8
    ind = np.where((res["v"][2]<0)&(res["p"][2]<margin))[0]
    alphaRd,Rd = met.radialDistance(res["p"][:,ind],res["w"][ind],nn,dr,res["nPh"])

    ind = np.where(res["v"][2]>0&(res["p"][2]>l-margin))[0]
    alphaTt,Tt = met.radialDistance(res["p"][:,ind],res["w"][ind],nn,dr,res["nPh"])

    ind = np.where((res["v"][0]<0))[0]
    alpha_ssyz,Ssyz = met.lineDistance(res["p"][:,ind],res["w"][ind],nn_,dr_,res["nPh"],y_range=5)

    print('# Ray Tracing save -> %s'%path)

    path_ = path+"_B"
    aa = alias_name+"_B"
    df = pd.DataFrame()
    df[aa] = Rd
```

```
    df.index = alphaRd
    df.to_csv(path_+".csv")

    path_ = path+"_F"
    aa = alias_name+"_F"
    df = pd.DataFrame()
    df[aa] = Tt
    df.index = alphaTt
    df.to_csv(path_+".csv")

    path_ = path+"_L"
    aa = alias_name+"_L"
    df = pd.DataFrame()
    df[aa] = Ssyz
    df.index = alpha_ssyz
    df.to_csv(path_+".csv")



def calc(iteral):
    gvp = generate_variable_params()
    range_params = determining_params_range()

    gvp.set_params(range_params)
    vp = gvp.generate(1)

    if vp['bv_tv'][0] > 0 and vp['th_cortical'][0] > 0:
        alias_name = "-".join((str(datetime.datetime.now().isoformat()).split('.')[0]).split(':'))+'_it'+f'{iteral:04}'

        print('### iteral number ',iteral)
        print('Alias name: ',alias_name)
        model_path = './model_result/'
        monte_path = './monte_result/'
        opt_path = './opt_result/'

        path_ = model_path+alias_name+'_dicom'
        u,bv_tv = generate_bone_model(vp['bv_tv'][0],path_,model_params)
        print('it: ',iteral,', change bvtv: ',vp['bv_tv'][0],'-->',bv_tv)
        vp['bv_tv'][0] = bv_tv
        path_ = monte_path+alias_name
        res,params_ = calc_montecalro(vp,iteral,monte_params,path_,u)
        print('###### end monte calro in it: ',iteral)
```

```
        path_ = opt_path+alias_name
        calc_ray_tracing(res,params_,path_,alias_name)
        print('')
        print('################ End %s it ###################'%iteral)
        print('')
    else:
        print("Invalid parameter was generated")
        print("BV/TV : ",vp['bv_tv'][0])
        print("th_cortical : ",vp['th_cortical'][0])



if __name__ == "__main__":

    for iteral in range(repetitions):
        calc(iteral)
    print()
    print('######################')
    print(datetime.datetime.now())
```

pyMonteOpt/vOBD.ipynb

```
import datetime
dt_now = datetime.datetime.now()
print(dt_now)

%run virtualOBD.py
```

A2　機械学習用コード

ここでは，本論で用いた SVM のモデリング例を示す.

vOBDM5_SVM. ipynb

```
import pandas as pa
import numpy as np
from scipy import stats

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("ticks", {'grid.linestyle': '--'})
sns.set_context("paper")#paper notebook talk poster
cp = sns.color_palette(n_colors=24)
sns.palplot(cp)

from sklearn import metrics
from sklearn.model_selection import KFold
from sklearn.svm import SVR

def pooling(df,spl = 10):
    ind = np.arange(int(df.shape[0]/spl))*spl
    pool_val = [];ind_ = []
    for i in range(len(ind[:-1])):
        pool_val.append(df.values[ind[i]:ind[i+1]].mean(0))
        ind_.append(np.mean(df.index[ind[i]:ind[i+1]]))
    pool_val.append(df.values[ind[-1]:].mean(0))
    ind_.append(np.mean(df.index[ind[-1]:]))
    return pa.DataFrame(np.array(pool_val),columns = df.columns,index = ind_)

###### データの準備 ######
## データの読み込み
B = pa.read_csv('B.csv',index_col = 0) # 後方
F = pa.read_csv('F.csv',index_col = 0) # 前方
L = pa.read_csv('L.csv',index_col = 0) # 側方
sim_params = pa.read_csv('params.csv',index_col = 0)

B = B.sort_index(axis='columns')
```

```
F = F.sort_index(axis='columns')
L = L.sort_index(axis='columns')
sim_params = sim_params.sort_index(axis='index')

## データの加工
# 使用領域の決定
B = B.iloc[5:100]
F = F.iloc[5:100]
L = L.iloc[:200]

# 側方の平滑化
nroll = 10
L_roll = L.rolling(nroll, center=True).mean()
L_roll = L_roll.dropna(axis=0)

# データ削減
L_roll_pool = pooling(L_roll,spl=20)
B_pool = pooling(B,spl=4)
F_pool = pooling(F,spl=4)

## 学習用データセットの準備
# 強度分布データ
dataset = pa.DataFrame(np.concatenate([(B_pool.T.values),np.log(L_roll_pool.T.values)],axis=1))

# 後方
dataset['B_mean'] = np.log(B_pool.values).mean(0)
dataset['B_var'] = np.log(B_pool.values).var(0)

# 前方
dataset['F_mean'] = np.log(F_pool.values).mean(0)
dataset['F_var'] = np.log(F_pool.values).var(0)

# 側方
dataset['L_mean'] = np.log(L_roll_pool.values[:]).mean(0)
dataset['L_var'] = np.log(L_roll_pool.values).var(0)

# ターゲット
dataset['target'] = sim_params.aBMD_square.values
features = dataset.drop('target',axis=1).columns

# Database normalization
```

```
X = dataset[features].values
X =(X-X.mean(0))/X.std(0)
y = dataset.target.values

##### モデリング #####
model = SVR(kernel='rbf', gamma=0.002,C = 5000,epsilon = 0.03)
model = model.fit(X,y)

##### 検証 #####
def train_(X,y,model,cv = 10,label = 'aBMD $[g/cm^2]$'):
    res_pred = np.zeros_like(y);r2_add=[]
    cv = KFold(n_splits=cv, shuffle=True, random_state=1234)
    for train_index, test_index in cv.split(X, y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        model = model.fit(X_train,y_train)
        pred = model.predict(X_test)
        res_pred[test_index] = pred
        print('r2 = ', metrics.r2_score(y_test,pred))
        r2_add.append(metrics.r2_score(y_test,pred))

    print('All r2 = ', metrics.r2_score(y,res_pred))
    print('Mean r2 = ', round(np.mean(r2_add),5),'±', round(np.std(r2_add),5))

    showGraph(y,res_pred,label = label,dpi=100)
    return res_pred

def correlationLine(x,y,score = 'r'):
    x = np.array(x).flatten()
    y = np.array(y).flatten()
    #相関
    if score =='r':
        slope, intercept, r_value, _, _  = stats.linregress(x,y)
        r, p = stats.pearsonr(x,y)
        print(stats.spearmanr(x,y))
        label_ = "r = "+str(round(r_value,3))
    if score == 'r2':
        r_value = metrics.r2_score(x,y)
        label_ = "$r^2$ = "+str(round(r_value,3))
        print('pearsonr:',stats.pearsonr(x,y))
    ysub = np.poly1d(np.polyfit(x,y,1))(x)
    xx = [x.min(),x.max()]
```

```
    yy = [ysub.min(),ysub.max()]
    if r_value < 0:
        yy = [ysub.max(),ysub.min()]
    plt.plot(xx,yy,"--",color="0.2",label = label_)

def showGraph(y,res_pred,label = 'aBMD $[g/cm^2]$',dpi=100,figsize =(4,4)):
    plt.figure(figsize=figsize,dpi=dpi)
    plt.plot(y,res_pred,'.' ,c = 'k')
    correlationLine(y,res_pred,score = 'r2')
    plt.legend(edgecolor='none')
    plt.xlabel('Reference '+label)
    plt.ylabel('Predicted '+label)
    plt.show()

    plt.figure(figsize=figsize,dpi=dpi)
    plt.plot((y+res_pred)/2,y-res_pred,'.' ,c = 'k')
    max_ = max((y+res_pred)/2)
    min_ = min((y+res_pred)/2)
    mean_ = np.mean(y-res_pred)
    upp_ = mean_+1.96*np.std(y-res_pred)
    low_ = mean_-1.96*np.std(y-res_pred)
    plt.plot([min_,max_],[mean_,mean_],':',c = 'k')
    plt.plot([min_,max_],[upp_,upp_],'-.',c = 'k')
    plt.plot([min_,max_],[low_,low_],'-.',c = 'k')
    correlationLine((y+res_pred)/2,y-res_pred)
    plt.legend(edgecolor='none')
    label_ = 'Mean of predicted and reference '+label
    plt.xlabel(label_)
    label_ = 'Difference between \n predicted and reference '+label
    plt.ylabel(label_)
    plt.xlim(0,1)
    plt.show()
    print('loa_upper: ',round(upp_,5))
    print('loa_lower: ',round(low_,5))
    print("Mean: ",mean_)

model = SVR(kernel='rbf', gamma=0.002,C = 5000,epsilon = 0.03)
pred_ = train_(X,y,model,cv = 10,label = 'aBMD $[g/cm^2]$')

showGraph(y,pred_,label = 'aBMD $[g/cm^2]$',dpi=200)
```

謝　辞

本研究を進めるにあたり，学部から数えて、7 年間にわたって様々なご指導いただきました田中茂雄教授には紙面上ではありますが深く感謝の意を表します．また，ご助言とご指導をいただきました坂本二郎教授，北山哲士教授，田中志信教授，内藤尚准教授，村越道生准教授にお礼申し上げます．金沢大学病院整形外科の松原秀憲助教に関しましては骨の医学的側面に対しご助言いただきました．石川工業高等専門学校，越野亮准教授には主に機械学習の面でご助言いただきました．ベンチャービジネスラボラトリー，産学官地域アドバイザーの林伸市様ならびに栗正治様，金沢大学 TLO 様には主に本装置のビジネス方面にて様々なアドバイスまた実際にご協力をいただきました．また，金沢市ものづくり産業支援課の山田康弘様，IT ビジネスプラザ武蔵アドバイザーの宮田人司様にもビジネスサイドからの助言をいただき，金沢市役所様より研究の支援をいただきました．２重学位プログラム並びに臨床試験プロジェクト際には，King Mongkut’s University of Technology Thonburi（KMUTT）のアナック・カンタチチャワナ准教授のご協力なくしては実現できませんでした．また，Chulabhorn 病院様には，臨床試験を行うにあたり施設，技術また資金面で全面的なご協力をいただきました．3 年間 KMUTT に留学した際公私ともにお世話になりました SMART Lab の皆様，ならびに国際学会に参加する際にプレゼンテーションの指導を引き受けてくださいました Darunsikkhalai School for Innovative Lerning の英語教師であるサーヴェッシュ・ナイドゥー氏およびマジッド・パティソン氏に深く感謝の意を申し上げます．そして，公私ともにお世話になった同輩の渡辺寛之氏，泉雅樹氏に感謝し，今後のご健勝とご活躍をお祈りいたします．また，同研究室でお世話になりました，赤江景氏ならびに諸先輩がた，イサク・サルタナ・リムボン氏，モグダラ・クンティカ氏，若森毅士氏，柳瀬義寛氏，小林瑞明氏，鮫島寛幸氏，大竹氏竣介，田村陽生氏に感謝とともに今後の研究生活ならびにご健勝とご活躍をお祈りしております．この研究生活の 7 年間を通してお世話になったすべての皆様にこの場を借りて感謝し謝辞とさせていただきます．