ポートフォリオ

# 深層学習を用いた最適化手法による高次直列結合マイクロリング共振器波長フィルタの設計の高効率化

令和 6 年 2 月 7 日提出

指導教員 荒川太郎 教授

横浜国立大学大学院理工学府

数物・電子情報系理工学専攻
応用物理分野

22NC307 福與 太一

# 要約

近年，情報トラフィックの増加によって，伝送容量，消費電力，通信速度の向上が必要とされている．これを解決する手段の一つとしてフォトニックネットワークが存在し，光波長フィルタはその経路制御に用いられている重要なデバイスである．本研究では，光波長フィルタの一つであるマイクロリング共振器（Micro Ring　Resonator: MRR）光波長フィルタに注目した．本研究の対象はマイクロリングを複数並べた高次直列結合マイクロリング波長フィルタであり，これは小型で設計性に優れている．しかし，リング次数が増加するに連れて設計パラメータも同様に増加してしまうことや，設計時に要求される条件が多く，それらは互いに独立していない．そのため，所望の条件を満たす最適な設計値を得ることがヒューリスティックな方法においては困難である．これを解決する手段として深層学習を用いる．ただし，メタヒューリスティクス全般に言えることだが，解の最適性，収束性は保証されていないことに注意したい．

本研究では深層学習の代表的な手法の一つである全結合型ディープニューラルネットワーク(Deep Neural Network: DNN)を用いる．これによって視覚的かつより直感的な最適化を目指し，理想的なフィルタ特性を実現する設計値を得ることを目的とする．本研究に用いられる深層学習(ディープラーニング)とは，人間の神経細胞を模したニューラルネットワークを用いた機械学習の手法の一つである．深層学習は回帰及び分類において大きな成果が得られるため，本研究では回帰に注目した．結果として，異径 8 次までに対して所望の特性を持つグラフを与える事で，それを再現する設計値を得る事を可能とした．挿入損失を 3 dB，3 dB 波長帯域幅を 0.8 nm とするなら異径 6 次 MRR が最適であり，結果として，所望の特性を得るための最適なリング次数はその特性によって異なるという事，トップとクロストーク，3dB 波長帯域幅と形状及び挿入損失はトレードオフの関係にあるという事，両側に同径のリングを並べる従来の並び方はバランスよく設計できるが，トップの部分の形状がよりよいリングの並び方は他にあるという事の三つを結論付けた．

# 目次

# 第１章　序論

## 1.1　研究背景

近年，インターネット利用の拡大及びソーシャルメディアやオンラインサービスの普及による動画や高画質のコンテンツ増加によって，ネットワークに要求される情報量は増加し続けている．これに起因して，伝送容量の増大，消費電力の低減，通信速度の高速化の三点を実現する事が必要とされている．伝送容量の向上においては，単一の伝送経路に対して複数のデータストリームを同時に送信可能とする波長分割多重方式（Wavelength Division Multiplexing: WDM）が存在する．更に，消費電力及び通信速度の向上を図るためには光信号を変換を介さずに伝送する必要があるが，これを解決するものとしてフォトニックネットワークが存在し，高効率で低消費電力なフォトニック素子の開発が進んでいる．光波長フィルタはフォトニックネットワークの光路制御において重要なデバイスである．光波長フィルタとして代表的なものとしてアレイ導波管格子(Arrayed Waveguide Grating: AWG)[1]が存在するが，これは構造が直線的であり，より小型な光波長フィルタが望まれている．マイクロリング共振器（Micro Ring　Resonator: MRR）光波長フィルタは小型でアレイ導波管格子に対してより狭域である．また，これまでに様々なタイプの MRR 波長フィルタの研究・開発がされてきた[2-16]．加えて，高次直列結合 MRR は 3dB 波長帯域や共振波長間隔などを自由に設計できる設計性の高さから，光波長フィルタとしての性能を効果的に向上させられる．しかし、設計性の高さゆえに所望のフィルタ特性を得る事は困難である。この高次における設計の困難さを解決するために様々な設計手法が提案され[17-19]，近年では機械学習的手法の一種である差分進化法を用いた設計が提案されている[20]．また，光分野全体においても機械学習を用いた例は多数存在し，深層学習によるフォトニックナノ結晶の Q 値最適化 [21]，ニューラルネットワークによるセンシングしたデータの分類 [22], [23]，ニューラルネットワークによる多モード自己干渉 MRR のマルチパラメータセンシング [24]，深層学習による超短パルスの再構成 [25]などの，人の手では解決が困難な問題に適用されている．本研究では，高次直列結合マイクロリング MRR に対して機械学習の中でも深層学習と呼ばれるものを適用し，設計を行っていく．

## 1.2　研究目的

マイクロリング共振器波長フィルタは小型で設計性に優れているが，リング次数が増加するにあたって設計パラメータも同様に増加し，加えて設計時に要求される条件が多く，それらは互いに独立していない．要求される条件とは，共振波長間隔（Free Spectral Range: FSR），3dB 波長帯域幅，中心波長，リップル，クロストークなどがあげられる．そのため，最適な設計値を得ることがヒューリスティックな方法においては困難である．また，これまでに，伝達行列法[17]，結合モード理論[18]，デジタルフィルタ設計[19]，差分進化法[20]

などの設計手法が設計・開発されてきたが，これらの手法はリング周長や結合率の決定の際にフィルタ特性に対する複数の評価項目や評価関数があり，それらを用いてより良い結果を模索する手法である．これらに対して本研究では，機械学習の中でも深層学習と呼ばれるものを適用して高次直列結合 MRR フィルタを設計することを提案する．従来の機械学習は評価対象を指定し，どのように評価するのかを人の手で決める必要があるが，深層学習ではその必要がないため，人の目で気付けない特徴を見抜き，予測することが可能である．本研究に用いられる深層学習とは，人間の神経細胞を模したニューラルネットワークを用いた機械学習的手法の一つである．深層学習は回帰及び分類において大きな成果が得られるため，本研究では特に回帰に注目した．また，光デバイス設計において機械学習を用いた例が近年増えている[21-27]．MRR フィルタの設計に機械学習を適用した報告はこれまでに”デジタルフィルタ設計法を用いた直列結合マイクロリング共振器波長フィルタの設計”[19]及び”差分進化法を用いた高次直列結合マイクロリング波長フィルタの設計手法の検討”[20]があるが，中でも深層学習を適用し設計を行った報告はない．コンピュータ全体の性能向上により，機械学習アルゴリズムの現実的な適用可能性が高まってきており，ファジー理論はマイクロリング共振器波長フィルタの最適化に用いられ[28]，勾配降下法は 4 次リングの設計に用いられ[29]，近年では差分進化法によって最大 14 次リングの設計が行われたが、本研究では同等のリング次数の設計が可能である．

## 1.3 本論文の構成

本論文では第 2 章でマイクロリング共振器の基本原理と本研究に用いている深層学習の基本原理について述べる．第 3 章ではマイクロリング共振器に対しての深層学習の適用方法と評価方法について述べる．第 4 章では実際にプログラムを用いて最適化した結果を述べる．第 5 章では得られた結果とそれに対する考察を述べる．

# 第２章　理論

## 2.1　はじめに

ここでは 2.2 節でマイクロリング共振器の基本原理を，2.3 節及び 2.4 節で本研究に用いている深層学習の基本原理について述べる．

## 2.2　マイクロリング共振器の原理

### 2.2.1　マイクロリング共振器の基本原理

マイクロリング共振器（Micro Ring Resonator: MRR）を用いた光波長フィルタの一つとして，特定の波長に対して分離及び混合を行う光分岐挿入装置(Optical Add/Drop Multiplexer: OADM)が存在する．これは様々な波長を持つ光が入力されるのに対し，特定の波長を持つ光が出力される機能を持つ．

図 2.1 に OADM の一種であるマイクロリング共振器を示す．このマイクロリング共振器は，様々な波長を持つ光が入力された際に，リングと共振をする波長の光のみが出力される性質がある．リングと特定の光とが共振するための条件式は式(2.1)のように表せる．ここで，$\beta$は伝搬定数，$L$はリング周長，$N$は共振次数である．

$$\beta L = 2\pi N \tag{2.1}$$

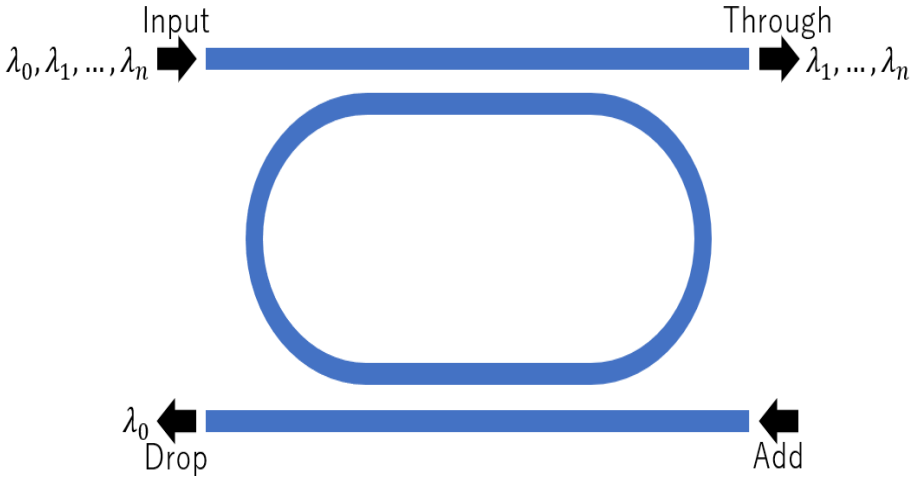


図 2.1 マイクロリング共振器

一般的に波長$\lambda$と伝搬定数$\beta$には式(2.2)のような関係がある．

$$\lambda = \frac{2\pi}{\beta} \tag{2.2}$$

ここで，伝搬経路内の実行屈折率を$n_{\mathrm{eff}}$とすると式(2.3)のように表せる．

$$\lambda = \frac{2\pi n_{\mathrm{eff}}}{\beta} \tag{2.3}$$

共振波長を$\lambda_0$とすると式(2.1)，(2.3)より式(2.4)と変形できる．

$$\lambda_0 = \frac{n_{\mathrm{eff}} L}{N} \tag{2.4}$$

式(2.4)をリング周長$L$について，及び共振次数$N$についてそれぞれ変形すると，

$$L = \frac{\lambda_0 N}{n_{\mathrm{eff}}} \tag{2.5}$$

$$N = \frac{L n_{\mathrm{eff}}}{\lambda_0} \tag{2.6}$$

式(2.6)について，$\lambda_0$で微分をすると，

$$\frac{dN}{d\lambda_0} = \frac{d}{d\lambda_0}\frac{n_{\mathrm{eff}} L}{\lambda_0}$$

$$= \frac{\lambda_0 \dfrac{d}{d\lambda_0} n_{\mathrm{eff}} L - n_{\mathrm{eff}} L \dfrac{d}{d\lambda_0}\lambda_0}{\lambda_0^2}$$

$$= \frac{\lambda_0 \dfrac{d}{d\lambda_0} n_{\mathrm{eff}} L - n_{\mathrm{eff}} L}{\lambda_0^2} \tag{2.7}$$

共振次数 N が 1 だけ異なる時の波長の間隔を$\Delta\lambda$とし，その際に$dN = 1$となることから，

$$\Delta\lambda = -\frac{\lambda_0^2}{n_{\mathrm{eff}} L - \lambda_0 \dfrac{d}{d\lambda_0} n_{\mathrm{eff}} L}$$

$$= -\frac{\lambda_0^2}{n_{\mathrm{g}} L} \tag{2.8}$$

このとき，式(2.8)の$n_{\mathrm{g}}$を群屈折率と呼び，式(2.9)のように定義した．

$$n_{\mathrm{g}} = n_{\mathrm{eff}} - \lambda_0 \frac{d}{d\lambda_0} n_{\mathrm{eff}} \tag{2.9}$$

また，$|\Delta\lambda|$を共振波長間隔（Free Spectral Range: FSR）と呼び，式(2.10)のようにかける．

$$\mathrm{FSR} = |\Delta\lambda| = \frac{\lambda_0^2}{n_{\mathrm{g}} L} \tag{2.10}$$

### 2.2.2　マイクロリング共振器の伝達関数

マイクロリング共振器内を光が伝搬するモデルを図 2.2 に示す．但し，マイクロリング共振器における光の伝搬には導波路及びリング間での光の結合と，リング内を周回する光の

伝搬とで分けて考えていることに注意したい．同一導波路を伝搬する場合には$\eta\sqrt{t}$を掛け合わせ，隣接する導波路に結合する場合には$-j\eta\sqrt{K}$を掛け合わせる．導波路間で光が結合する際には位相が$\frac{\pi}{2}$遅れることから$-j$をかけている．ここで，$K$は結合率，$t$は透過率，$\eta$は電界透過率であり，式(2.11)が成立する．また，式(2.12)の連立方程式が成立する．

$$K + t = \eta \tag{2.11}$$

$$\begin{cases} R_n = -j\eta\sqrt{K}R_{n-1} + \eta\sqrt{t}T_n \\ T_{n-1} = -j\eta\sqrt{K}T_n + \eta\sqrt{t}R_n \end{cases} \tag{2.12}$$

この式(2.12)を変形し，行列表示にすると式(2.13)のようになる．

$$\begin{bmatrix} T_n \\ R_n \end{bmatrix} = -\frac{1}{j\eta\sqrt{K}} \begin{bmatrix} 1 & -\eta\sqrt{t} \\ \eta\sqrt{t} & -\eta^2 \end{bmatrix} \begin{bmatrix} T_{n-1} \\ R_{n-1} \end{bmatrix} \tag{2.13}$$

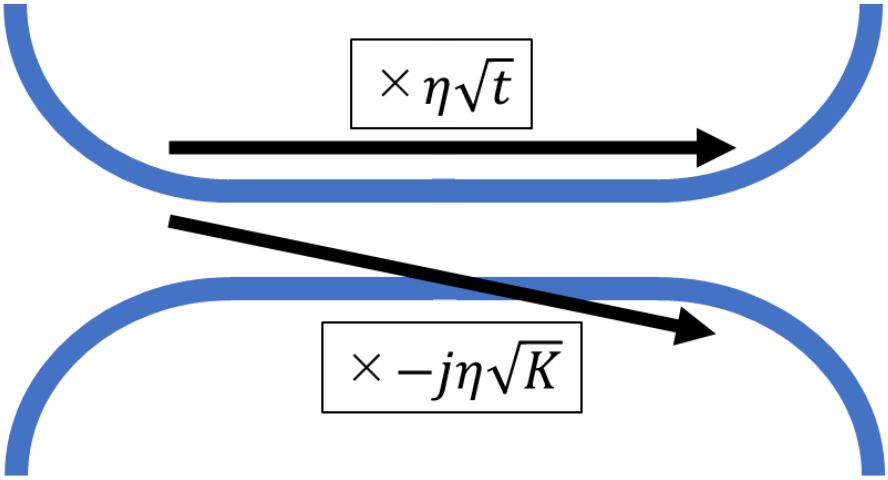


図 2.2 マイクロリング共振器の光の伝搬

ここで，$T_n, T_{n-1}, R_n, R_{n-1}$の関係式の簡単化のため，式(2.13)を用いて式(2.14)のように$C(K)$と置くと式(2.13)を式(2.15)の様に書き換えられる．

$$C(K) = -\frac{1}{j\eta\sqrt{K}} \begin{bmatrix} 1 & -\eta\sqrt{\eta - K} \\ \eta\sqrt{\eta - K} & -\eta^2 \end{bmatrix} \tag{2.14}$$

$$\begin{bmatrix} T_n \\ R_n \end{bmatrix} = C(K) \begin{bmatrix} T_{n-1} \\ R_{n-1} \end{bmatrix} \tag{2.15}$$

この式(2.15)から結合部における光の伝搬は結合率$K$によって決定されることがわかる．次に，リング内を周回する光の伝搬について考える．リング内を周回する光の伝搬モデルを図 2.3 に示す．光がリング内を半周周回する場合に$\sqrt{a}e^{-\frac{j\beta L}{2}}$を掛け合わせることで，リング内の伝搬を式(2.17)の連立方程式で表せる．但し，伝搬損失係数を$\alpha$としたとき，電力の透過率$a$については式(2.16)として表される．

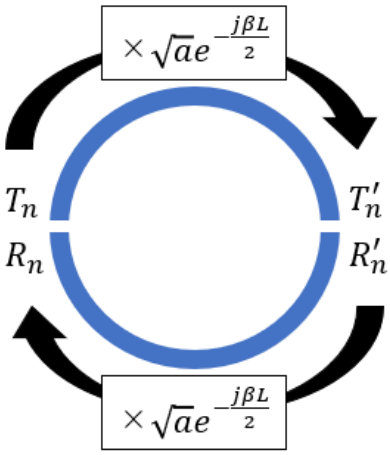


図 2.3 単一マイクロリング内の光の伝搬

$$a = e^{-\alpha L} \tag{2.16}$$

$$\begin{cases} T_n' = \sqrt{a}e^{-\frac{j\beta L}{2}} T_n \\ R_n = \sqrt{a}e^{-\frac{j\beta L}{2}} R_n' \end{cases} \tag{2.17}$$

この式(2.17)を変形し，行列表示すると式(2.18)と表される．

$$\begin{bmatrix} T_n \\ R_n \end{bmatrix} = \begin{bmatrix} \frac{e^{\frac{j\beta L}{2}}}{\sqrt{a}} & 0 \\ 0 & \sqrt{a}e^{-\frac{j\beta L}{2}} \end{bmatrix} \begin{bmatrix} T_n' \\ R_n' \end{bmatrix} \tag{2.18}$$

ここで，$T_n, T_n', R_n, R_n'$の関係式の簡単化のため，式(2.19)のように$R(a, L, \lambda)$と置くと，式(2.20)のように表される．

$$\begin{bmatrix} T_n \\ R_n \end{bmatrix} = R(a, L, \lambda) \begin{bmatrix} T_n' \\ R_n' \end{bmatrix} \tag{2.19}$$

続いて，M 次直列結合マイクロリング共振器を考える．図 2.4 に M 次直列結合マイクロリングのモデルを示す．式(2.15)と式(2.19)を用いると M 番目と M-1 番目のリングにおける関係式は式(2.20)の様に表せる．

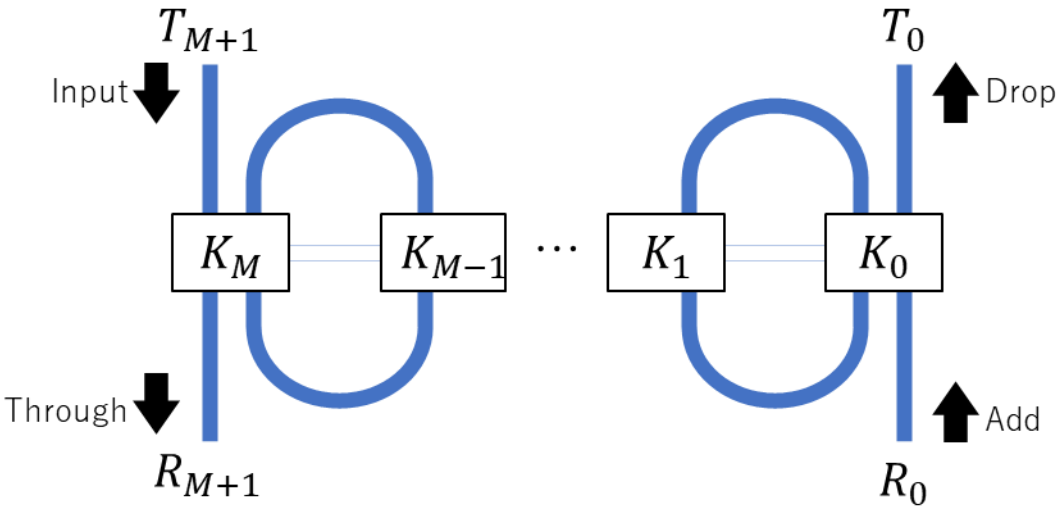


図 2.4 M 次直列結合マイクロリング

$$\begin{bmatrix} T_M \\ R_M \end{bmatrix} = R(a_{M-1}, L_{M-1}, \lambda)C(K_{M-1}) \begin{bmatrix} T_{M-1} \\ R_{M-1} \end{bmatrix}$$

$$= \Phi_{M-1} \begin{bmatrix} T_{M-1} \\ R_{M-1} \end{bmatrix} \tag{2.20}$$

式(2.20)で用いた$\Phi_{N-1}$について展開すると，式(2.21)の様に表せる．

$$\Phi_{M-1} = R(a_{M-1}, L_{M-1}, \lambda)C(K_{M-1})$$

$$= -\frac{1}{j\eta\sqrt{a_{M-1}K_{M-1}z^{-1}}} \begin{bmatrix} 1 & -\eta\sqrt{\eta - K_{M-1}} \\ \eta a_{M-1}\sqrt{\eta - K_{M-1}}z^{-1} & -a_{M-1}\eta^2 z^{-1} \end{bmatrix}$$

$$= -\frac{1}{j\gamma_{M-1}\sqrt{K_{M-1}z^{-1}}} \begin{bmatrix} 1 & -\eta\sqrt{\eta - K_{M-1}} \\ \frac{\gamma_{M-1}^2}{\eta}\sqrt{\eta - K_{M-1}}z^{-1} & -\gamma_{M-1}^2 z^{-1} \end{bmatrix} \tag{2.21}$$

ただし，

$$\gamma_n^2 = \eta^2 a_n \tag{2.22}$$

と置いている．ここで，M+1 次直列結合マイクロリング共振器のモデルを考える．これを図 2.5 に示す．式(2.20)を用いることで M 次における伝達関数を求めることが出来る．

$$\begin{bmatrix} T_{M+1} \\ R_{M+1} \end{bmatrix} = \prod_{n=0}^{M} \Phi_n = \frac{1}{(-j)^{M+1} z^{-\frac{M+1}{2}} \prod_{n=0}^{M} \gamma_n \sqrt{K_n}} \begin{bmatrix} A_M(z) & C_M(z) \\ B_M(z) & D_M(z) \end{bmatrix} \begin{bmatrix} T_0 \\ R_0 \end{bmatrix} \tag{2.23}$$

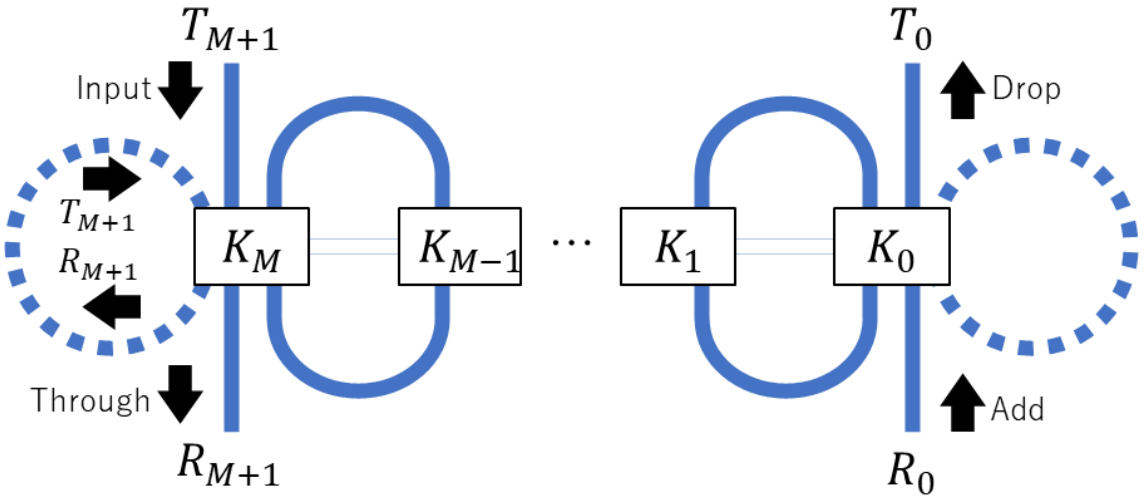


図 2.5 M+1 次直列結合マイクロリング共振器

但し，$A_M(z), B_M(z), C_M(z), D_M(z)$は$z$の多項式である．ここで Add ポートからの光の入力は無いものとすると，$R_0 = 0$とすれば

$$H_M = \left|\frac{T_0}{T_{M+1}}\right| = \frac{\prod_{n=0}^{M} \gamma_n \sqrt{K_n}}{|A_M(z)|} \tag{2.24}$$

式(2.24)を用いてシミュレーションを行うと図 2,6 や図 2.7 のようなグラフを得られる．これらに用いている設計値は後述する表 2.1 と以下の表 2.2 を用いている．

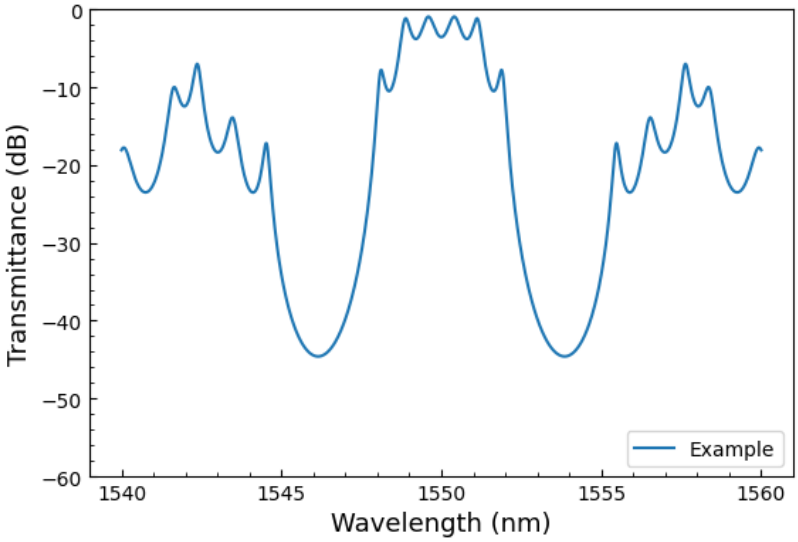


図 2.6 結合率$K_M = 0.5$とした 6 次リングのシミュレーション結果

表 2.1 図 2.6 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.5 | |
| 1 | 0.5 | 82.4 µm |
| 2 | 0.5 | 82.4 µm |
| 3 | 0.5 | 82.4 µm |
| 4 | 0.5 | 55.0 µm |
| 5 | 0.5 | 55.0 µm |
| 6 | 0.5 | 55.0 µm |

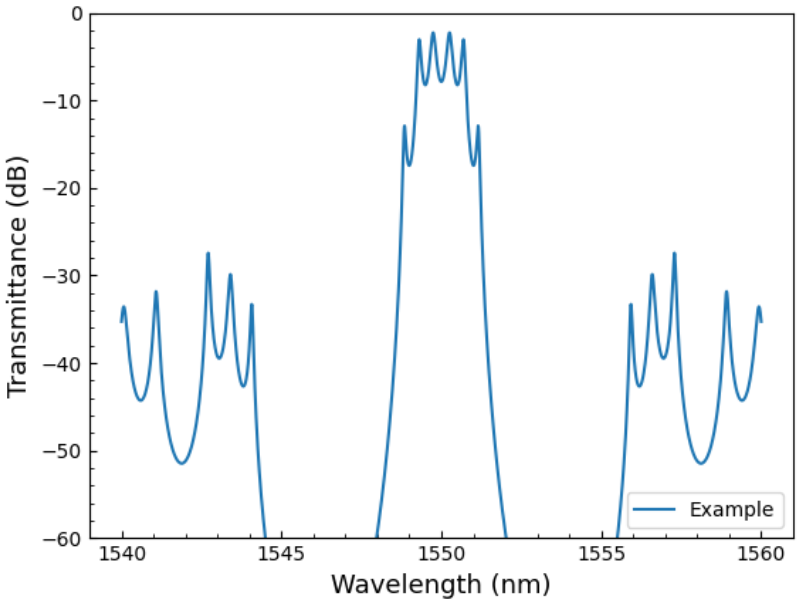


図 2.7 結合率$K_M = 0.2$とした 6 次リングのシミュレーション結果

表 2.2 図 2.7 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.2 | |
| 1 | 0.2 | 82.4 µm |
| 2 | 0.2 | 82.4 µm |
| 3 | 0.2 | 82.4 µm |
| 4 | 0.2 | 55.0 µm |
| 5 | 0.2 | 55.0 µm |
| 6 | 0.2 | 55.0 µm |

### 2.2.3　バーニャ効果

2 種類以上の異なるリング周長のマイクロリングを直列で組み合わせた時，伝達関数の共振波長間隔FSRが増大する．これをバーニャ効果と呼ぶ．図 2.8 に 2 次異径マイクロリング共振器における伝達関数とそれぞれのみを用いた単一マイクロリング共振器の伝達関数を示す．ここでのリング周長はそれぞれ$L = 100, 150\ (\mu m)$となっている．

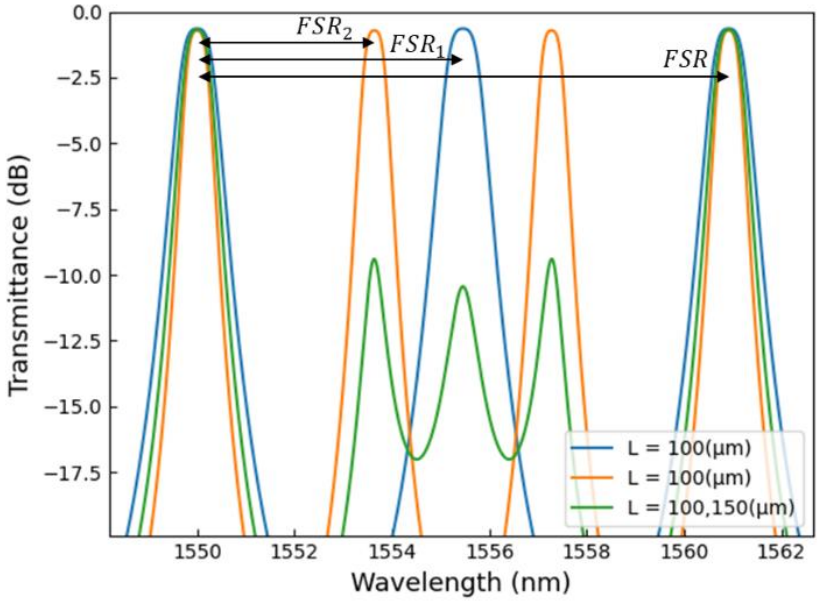


図 2.8 異径 2 次マイクロリング共振器の伝達関数

また，図 2.8 より式(2.25)の関係が成り立っていることが分かる．

$$\mathrm{FSR} = 2\mathrm{FSR}_1 = 3\mathrm{FSR}_2 \tag{2.25}$$

つまり，異径リングを組み合わせた際の実質的な共振波長間隔は各リングの共振波長間隔の最小公倍数であるといえる．

## 2.3　深層学習の原理 [30]

### 2.3.1　深層学習の基本原理

深層学習（ディープラーニング）とは機械学習の手法の一つであり，人間の神経細胞の仕組みを模した多層構造のニューラルネットワークのことを呼ぶ．特に，深層学習は音声や画像の認識，特定，予測や自然言語処理などにおいて高い成果を出している．深層学習によって画像や音声の認識を行うためには，膨大な量の教師データを用いて学習させる必要があ

るが，従来の機械学習とは違い人が特徴部分を定義する必要がない．まずはニューラルネットワークについての簡単な概要図を図 2.9 に示す．

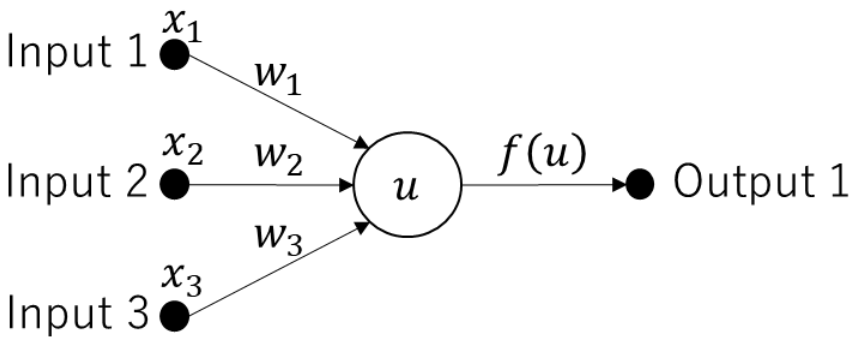


図 2.9 ノード間の入出力に関する概要図（入力$x_i$，重み$w_i$，内部データ$u$，出力$f(u)$）

これはニューラルネットワークにおける神経細胞と同等の役割を果たすものであり，パーセプトロンと呼ばれる．また，矢印の部分をエッジ，丸の部分をノードと呼ぶ．これを複数用いたものがニューラルネットワークであり，さらに多層化したものが深層学習である．

個々のノード間では，入力$x_i$，入力数 n，重み$w_i$，内部データ$u$，バイアス$b$，出力$f(u)$で構成される．これは式(2.26)で表されたものに活性化関数というものを通じたものが出力データとなる．

$$u = \sum_{i=1}^{n} w_i x_i + b \tag{2.26}$$

ここで，$n$は各層におけるノード数を示す．また，活性化関数は複数存在するが，本研究で用いているものは’ Identity function’と呼ばれるものであり，出力時に関数を通さずにそのまま出力するためのものである．

また，深層学習の学習時における評価方法としては損失関数$Loss$（従来では$L$と表記するが，リング周長$L$との差別化を図るために$Loss$と表記する）というものが存在する．これはニューラルネットワークによる予測データとそれに対応する教師データとの離れ度合いを評価するものである．本研究では平均二乗誤差(Mean Square Error: MSE)を用いている．つまり，値が低いほど評価が良いという事である．これを式(2.27)に示す．

$$Loss = \frac{1}{n}\sum_{i=1}^{n}(d_i - y_i)^2 \tag{2.27}$$

ここで, $d_n$は n 番目の実際の値, $y_n$は n 番目の予測した値である. これらに対して Dropout という過程が加わる. Dropout とは過学習を防ぐために用いられているものであり, 一定の割合で学習時に出力を 0 とすることで一部のデータが欠損していても正しく認識ができるようにするという狙いがある．以上をまとめたものを図 2.10 に示す．$y_i$とは，最終的な出力のことを示す．

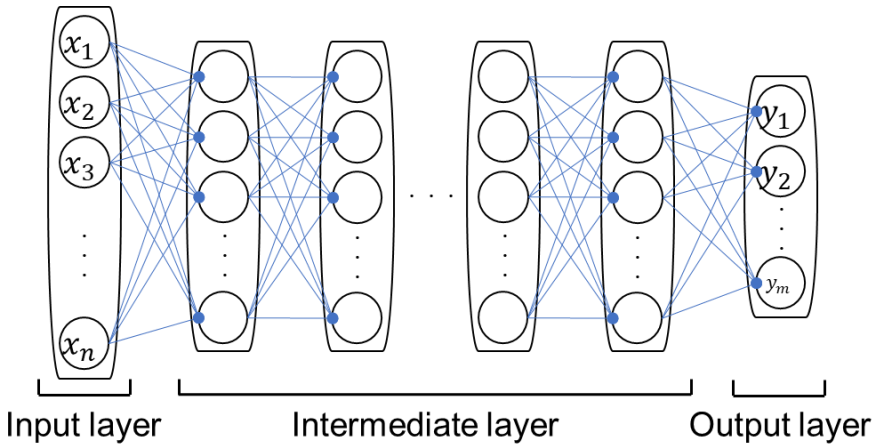


図 2.10 深層学習のモデル概要図

図 2.9 に示したような同じ役割のノードを複数まとめたものを層と呼ぶ．入力層（Input layer）とは入力データの格納先であり，本研究では伝達関数行列を入力データとして対応付けている．次に，中間層（Intermediate layer）について，一般的には入力層と後述する出力層の間に存在する層であり，活性化関数を通じて計算され，より複雑な特徴やパターンを学習するためのものである．最後に，出力層（Output layer）とは深層学習が与えられた入力に基づいて行った予測が生成されるための場所である．

学習時の最適化アルゴリズムであるオプティマイザーは Adam を用いており，学習率は 0.001 である．Adam とは再急降下法の派生形であり，損失関数の値を低くすること目的としたときに，現在では最も使われている[31]．学習率とは一回の学習でのパラメータ更新の度合いを示す値である．

# 第 3 章　設計手法

## 3.1　はじめに

ここではマイクロリング共振器波長フィルタの設計手法及びその過程について述べる．

## 3.2　深層学習の適用方法

この節では深層学習の適用方法について説明する．初期条件を設定し，結合率の決定方法及びハイパーパラメータの決定方法について述べる．

本研究では深層学習を用いる手法として，結合率$K$を無作為に生成し，設計値と設計値により算出される伝達関数行列を全結合ニューラルネットワーク上で学習させる．つまり，設計値と伝達関数との対応関係を学習させる．所望の特性を持つフィルタ特性のグラフを学習済みのモデルに入力し，入力に近い設計値を出力として得ることを最終的な目標とする．入力グラフは，図 3.1 に示すように，シミュレーションを行わずに目標設定を行うための模式的なグラフである．このグラフに対して後述する補正方法を適用する．これは，ディープラーニングの特性上，予測値を用いた伝達関数は目標に対して必ず誤差を持つため，目標に近づけるために近傍探索を行う必要があるためである．

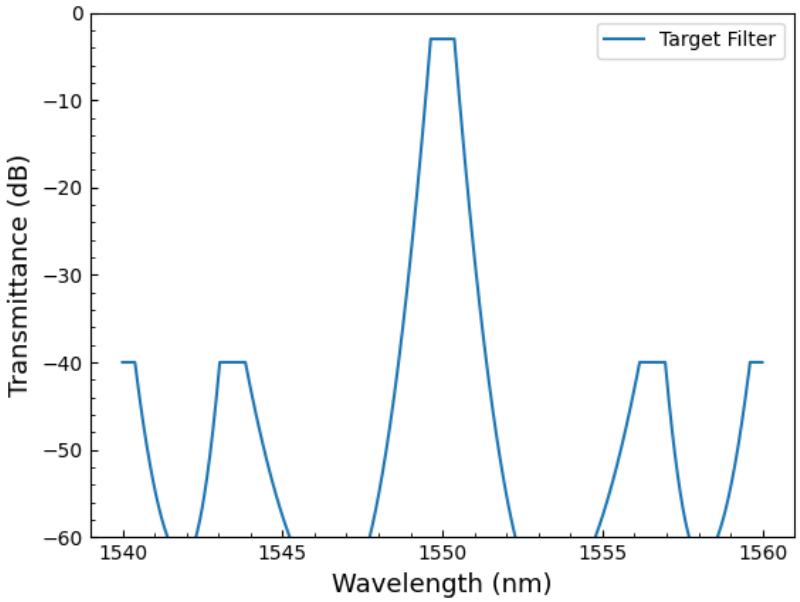


図 3.1 目標設定のための模式的なグラフ

### 3.2.1　初期条件

初期条件として，デバイス依存の定数である結合損$\eta$，実効屈折率$n_{\mathrm{eff}}$，群屈折率$n_{\mathrm{g}}$，伝搬

損失係数$\alpha$を設定し，設計時の条件として共振波長間隔FSR，中心波長$\lambda_0$，及び設計するリング次数$n$を与える．また，深層学習のモデルを作成するにあたって入力データ数，中間層の層数，中間層のノード数，出力データ数を与える．共振波長間隔が与えられるとリング周長$L$の組み合わせの候補が決定されるため，同一のものを用いている．これらをまとめたものと表 1 に示す．

表 3.1 初期条件

| | | |
|---|---|---|
| 実効屈折率 | $n_{\mathrm{eff}}$ | 2.2 |
| 群屈折率 | $n_{\mathrm{g}}$ | 4.4 |
| 結合損 | $\eta$ | 0.996 |
| 伝搬損失係数 | $\alpha$ | 52.96 Np/m |
| 中心波長 | $\lambda_0$ | 1550 nm |
| 共振波長間隔 | FSR | 20 nm |
| 入力データ数 | | 2000 |
| 中間層数 | | 2 |
| 中間層ノード数 | | 1250 |

### 3.2.2 ハイパーパラメータの決定

深層学習のモデルを作成するうえで重要となるのがハイパーパラメータである．これは中間層の数，中間層のノード数，損失関数，活性化関数などが挙げられる．最適なハイパーパラメータを探す設計手法は確立されていないため，本研究において最適なものを模索するため，検証を行い決定した．

まず，中間層の数を決定するためにノード数を全て 1000 個に固定したうえで中間層を増やし，$Loss$及び学習時間を測定した．ノード数を 1000 個に固定したのは入力数 2000 に対してその半分であるためであり，入力に対して少ないノード数であれば結果は変わらないと予想されるためである．検証結果を図 3.2 に示す．Training time per epoch は訓練データを一巡するのにかかる平均時間を示している．前提として，結合率を 0 から 1 まで一様に分布させた学習データセット数を 10 万，学習回数は学習が収束したと確認できるまでとした．

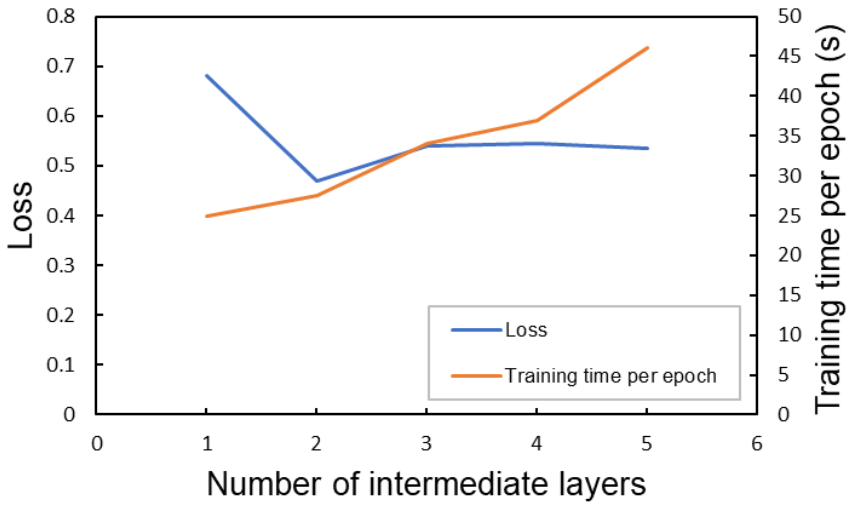


図 3.2 中間層の層数の検証結果

この結果より，精度が最も高いと考えられるのは中間層の数が 2 層の時であることが分かる．3 層以上の場合においても殆ど同様の*Loss*を得られていることや，更に層数を増やすことでより精度を高められる可能性があるが，学習時間の指数的増加が予想されるため，中間層の数は 2 層で決定した．

次に中間層のノード数を決定するため，中間層の層数を 2 層としたうえでノード数を増やし，*Loss*及び学習時間を測定した．その結果を図 3.3 に示す．

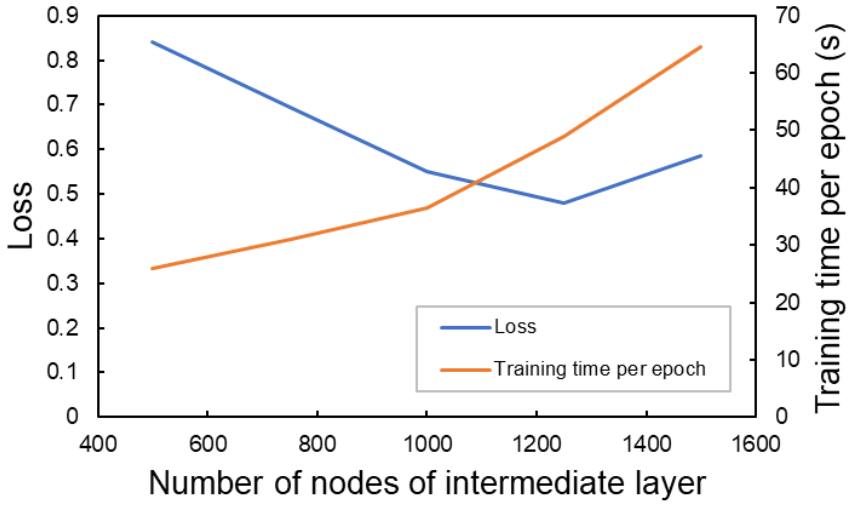


図 3.3 中間層のノード数の検証結果

この結果より，精度が最も高いと考えられるのはノード数が 1250 の時であることが分かる．一般的に深層学習で回帰を用いる際には入力層のノード数より少ない方がいいとさ

れているため，ノード数を 1250 で決定した．

### 3.2.3　設計値の学習

3 章のモデルを用いる前提での全体の最適化フローチャートを図 3.4 に示す．

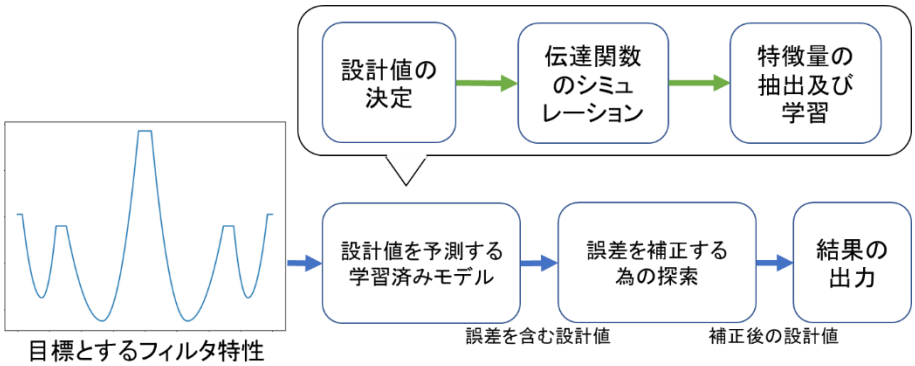


図 3.4 最適化のフローチャート

まず，深層学習において学習に必要なデータを用意する必要がある．本研究では設計値である結合率を無作為に決定する方法として，0.05 から 0.6 の間で一様分布となるような乱数を 10 万セット用意した．これらの設計値のセットを用いてシミュレーションを行い，学習を行う．

式(2.24)を模したプログラムを用いてマイクロリング共振器の伝達関数を計算している．深層学習に対する入力数について，マイクロリング共振器の特性上，共振波長は一定の間隔で現れる．つまり，目標設計値である共振波長間隔FSR及び中心波長$\lambda_0$より，式(3.1)の範囲を考えればよい

$$\lambda_0 - \frac{1}{2}\mathrm{FSR} \leq \lambda \leq \lambda_0 + \frac{1}{2}\mathrm{FSR} \tag{3.1}$$

つまり，十分な計算範囲は共振波長間隔FSRだけで考えればよい．また，計算される伝達関数行列のグラフの精度低下を防ぐために計算間隔を0.01 nmとしているため，必要な計算量は式(3.2)より$FSR \times 10^{11}$回となる．本研究では，FSR = 20 (nm)としているため，計算量（伝達関数行列の長さ）は 2000 となる．また，想定している手法は伝達関数の入力から出力とする結合率$K$の対応関係を学習させ，予測することである．即ち深層学習を用いた回帰である．

### 3.2.4　補正手法

深層学習による予測は必ず誤差が生じてしまう．これを解決するために簡易的な補正を加える手法を用いた．

ここでは，フローチャートに示す通り，設計にあたって深層学習の出力に対して補正を加える仕組みについて述べる．深層学習から得られた出力は誤差逆伝搬法という原理に基づくものであるため，最適解ではなくとも求める解に近い値が得られる．予測結果例を表 3.2 に示す．

表 3.2 深層学習による予測結果例

| | $K_0$ | $K_1$ | $K_2$ | $K_3$ | $K_4$ |
|---|---|---|---|---|---|
| 目標値 | 0.16 | 0.79 | 0.72 | 0.38 | 0.49 |
| 予測値 | 0.21 | 0.75 | 0.76 | 0.41 | 0.46 |

これら予測値をより目標に近づけるために補正する手法として勾配法というものを参考にした[32]. 勾配法とは主にニューラルネットワークで最適な重みを探すための探索手法であり，尚且つ単純なアルゴリズムで構成されている．このアルゴリズムの一般的な過程は

1．深層学習から得た値を出発地$x$とする．

$$x = (x_1, \dots, x_n) \tag{3.2}$$

と定める

2．出力値付近での評価関数の変化率を求める．

3．変化率をもとに次の出発地$x$`とする．

$$x' = (x_1', \dots, x_n') = \left(x_1 - \alpha\frac{\partial f}{\partial x_1}, \dots, x_n - \alpha\frac{\partial f}{\partial x_n}\right) \tag{3.3}$$

$\alpha$は学習率とよばれる一定の係数である．本研究では一般的に用いられている値として 0.01 に設定している．

4．2~3 の手順を繰り返し変化率が 0 になる$x$を見つける．

5．勾配の要素がすべて 0 となるまで更新し続け，最終的な解を

$$x_{\text{final}} = (x_1, \dots, x_n)$$

とする．

これら 4 つの手順で構成されている勾配法は解析的に説くのが困難な時に有効なアルゴリズムで，最適化問題などに用いられており，本研究では評価値が極小となる場所を探すために用いている．理由として，後述する評価関数は値が小さいほど評価が高い関数であるためである．

このアルゴリズムの具体的な適用方法についてまとめると，深層学習によって得られる誤差を含んだ設計値に対して勾配法により一定の値ずつずらし，評価値が低い方を採用する流れになっている．

## 3.3 評価方法

本研究では入力は理想的なフィルタであることを前提としていることと，設計値から計算されない伝達関数も想定している．そのためフィルタの特性としての評価ではなく，予測

値から計算される伝達関数がどれだけ入力に近い概形をとっているかが重要となる．即ち，入力と出力によって算出される伝達関数との積分値が小さいほど高い評価を与えている．積分の計算方法を式(3.3)に示す．

最適化における評価関数について，本研究では，学習済みモデルへの入力はある程度存在し得ない理想的なフィルタであると仮定し，シミュレーションによって得られたフィルタと比較する．そのため，入力のグラフと出力の設計値により得られるグラフの二つを比較するため，フィルタの特性としての評価ではなく，出力から計算される伝達関数のグラフがどれだけ入力のグラフに近い概形をとっているかが重要となる．即ち，入力と出力によって算出される伝達関数との積分値が小さいほど高い評価を与える必要がある．しかし，これら二つのグラフを面積で比較したとき一つの問題が生じてしまう．フィルタ特性において重要な部分はトップやサイドローブの部分であり，グラフの下部になるほど重要ではなく．また，グラフの上部の部分の積分値が小さい方がフィルタの性能としてはより重要になるため望ましい．つまり，下部の面積変化量を無視するために一定の閾値を用意し，よりグラフで上になる部分に対しての積分値は小さいほど高く評価する必要がある．加えて，深層学習の起用目的でもある評価関数の複雑化を避け，評価する二つのグラフ$y_1(\lambda), y_2(\lambda)$を比較するために，次式(3.4)を用いた．閾値の設定理由として，一般的にクロストークは 30 dB より大きい事が望ましいとされるためである．

$$E = \begin{cases} \displaystyle\int_{\lambda_0-\frac{1}{2}\mathrm{FSR}}^{\lambda_0+\frac{1}{2}\mathrm{FSR}} \left|1-\frac{y_1(\lambda)}{y_2(\lambda)}\right| d\lambda \left(y_1(\lambda) < y_2(\lambda)\right) \\ \displaystyle\int_{\lambda_0-\frac{1}{2}\mathrm{FSR}}^{\lambda_0+\frac{1}{2}\mathrm{FSR}} \left|1-\frac{y_2(\lambda)}{y_1(\lambda)}\right| d\lambda \left(y_2(\lambda) < y_1(\lambda)\right) \\ 0 \qquad\qquad (y_1(\lambda) \text{ and } y_2(\lambda) \leq -30) \end{cases} \tag{3.4}$$

# 第 4 章　異径 M 次リング共振器の設計

## 4.1　はじめに

ここでは，第 3 章で説明した深層学習のモデルを用いた最適化手法による設計結果を示す．

## 4.2　設計結果

例として，対象となる高次直列結合マイクロリングの中で異径 6 次リングを図 4.1 に示す．設計対象となる異径高次 MRR フィルタの設計結果を示す．

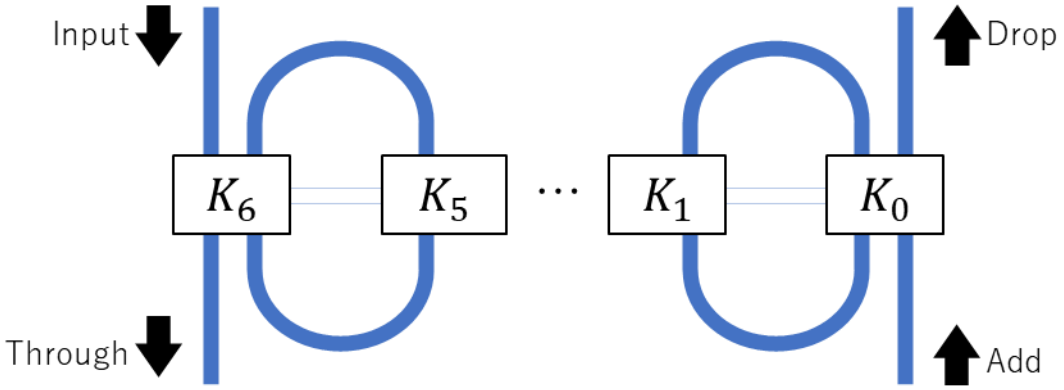


図 4.1　異径 6 次リングのモデル図

初期条件については表 3.1 を参照のもと，まずは異径 2 次リングの設計結果ついて示す．その後，4 次～8 次までの設計結果を示し，各種設計目標を変えた場合についての設計結果を示していく．最後に，本研究の成果の一つであるリングの並び方の最適化結果について，従来の並べ方との比較を行っていく．図 4.2 は表 4.2 の設計目標を用いて設計を行ったものであり，橙色の点線が設計目標のフィルタ，青色の実線が設計結果となるフィルタである．挿入損失を 3 dB，3 dB 波長帯域幅$\lambda_{3dB}$を 0.8 nm，リプルを 0 dB，クロストークを 30 dB，シェイプファクター$F$を 0.8 とした．これらの値は一般的に用いられている Mux/DeMux の性能を調査し決定したものである．但し，設計目標を与える上では一定の制約があり，ある程度の経験的なものである事には注意が必要である．例として，リング次数にもよるが挿入損失は 2dB 以上，クロストークはリング次数×15 dB 以下といった具合である．これは，物理的に不可能な目標を与えると，設定した目標に近づけようと最適化を行うアルゴリズムであるため，結果として性能の悪い設計結果が出てしまうからである．シェイプファクター$F$とは 1 dB 帯域の長さと 10 dB 帯域の長さを割ったものであり，目標値とした 0.8 とは，1dB 波長帯域幅が 0.8 nm の時に 10dB 波長帯域幅は約 0.9 nm となる値である．また，この値は 3dB 波長帯域幅にも影響があるため，3dB 波長帯域幅が長い場合は高く，短い場合

は低く設定する必要がある．よって 3dB 波長帯域幅に対し 10 dB 帯域幅が 0.1 nm 程度増加するようなシェイプファクターを設定する．定義式を式(4.1)に示す．また，挿入損失とは最低での損失を意味している．

$$F = \frac{\lambda_{1\mathrm{dB}}}{\lambda_{10\mathrm{dB}}} \tag{4.1}$$

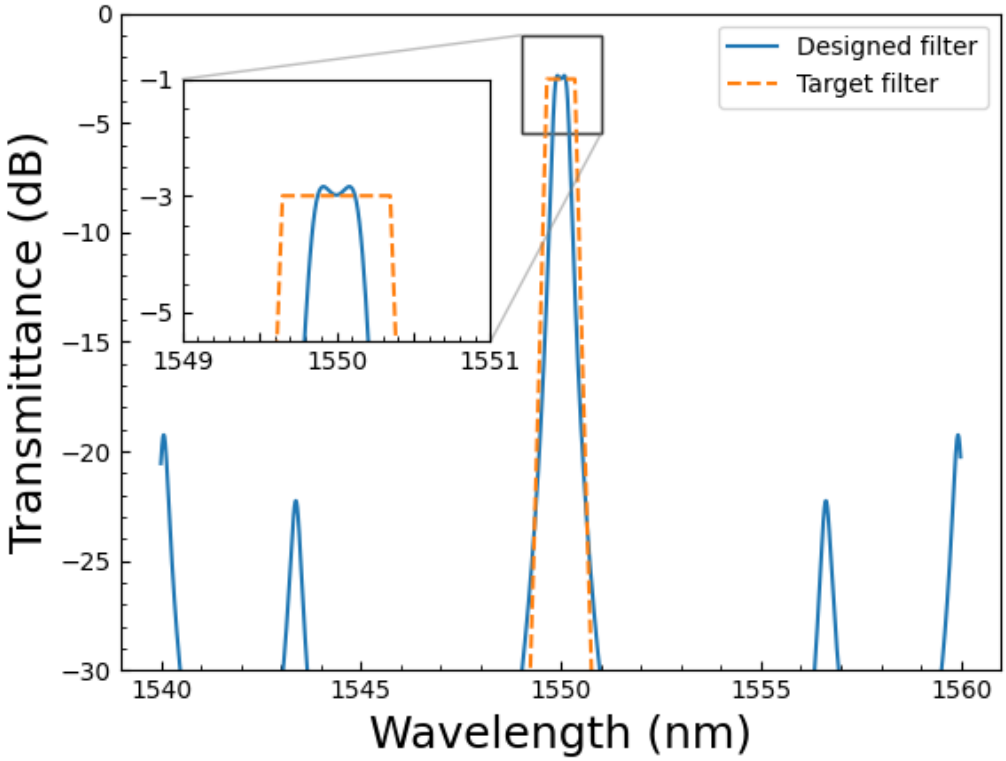


図 4.2 異径 2 次 MRR の設計結果（評価値：152.4）

表 4.1 図 4.2 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.12 | |
| 1 | 0.01 | 55.0 μm |
| 2 | 0.2 | 82.4 μm |

表 4.2 図 4.2 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 3 dB | 0.8 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | -2.84 dB | 0.44 nm | 0.14 dB | 16.4 dB | 0.61 |

異径 4 次 MRR においての設計では，設計目標に近い結果は得られなかった．挿入損失以外の部分については顕著で，3dB 波長帯域幅が約半分，結合率が全体的に低いのにも関わらずクロストークも同様の結果となってしまっている．これは，リングの個数が 2 つであるため箱型化しにくく，損失も発生しにくい事が原因として考えられる．

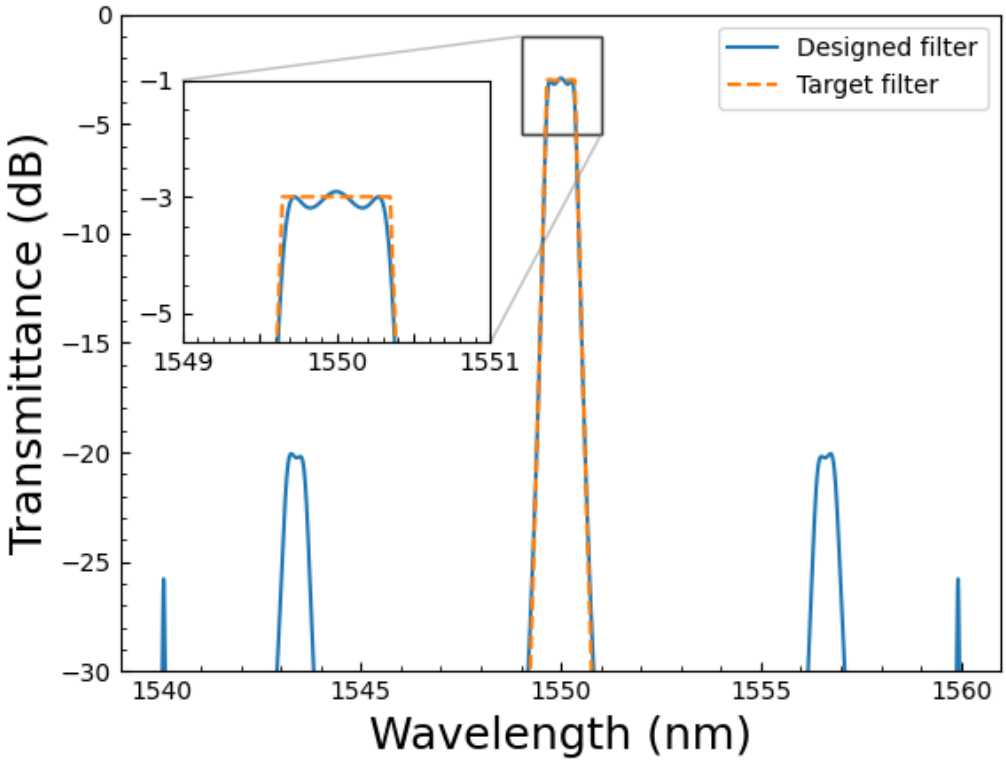


図 4.3 異径 4 次 MRR の設計結果（評価値：70.5）

表 4.3 図 4.3 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.45 | |
| 1 | 0.06 | 82.4 µm |
| 2 | 0.03 | 82.4 µm |
| 3 | 0.04 | 55.0 µm |
| 4 | 0.72 | 82.4 µm |

表 4.4 図 4.3 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 3 dB | 0.8 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | 2.91 dB | 0.78 nm | 0.27 dB | 17.1 dB | 0.78 |

異径 4 次 MRR においての設計結果はリングの個数が増えたため，異径 2 次 MRR より全体的な損失が大きくなり，より箱型化している．特性としてはクロストークが 17.1 dB と目標に対してかなり悪くなってしまっていて，フィルタとしての使用は難しい．これは，4 次リングにおける挿入損失や 3dB 波長帯域幅を重視した最適化が行われているためであると考えられる．また，昨年度はクロストークが 31 dB のものを設計したが，トップの形状や 3dB 波長帯域幅が 0.94 nm という設計結果になったことから，クロストークとトップの部分の性能はトレードオフの関係にあると考えられる．

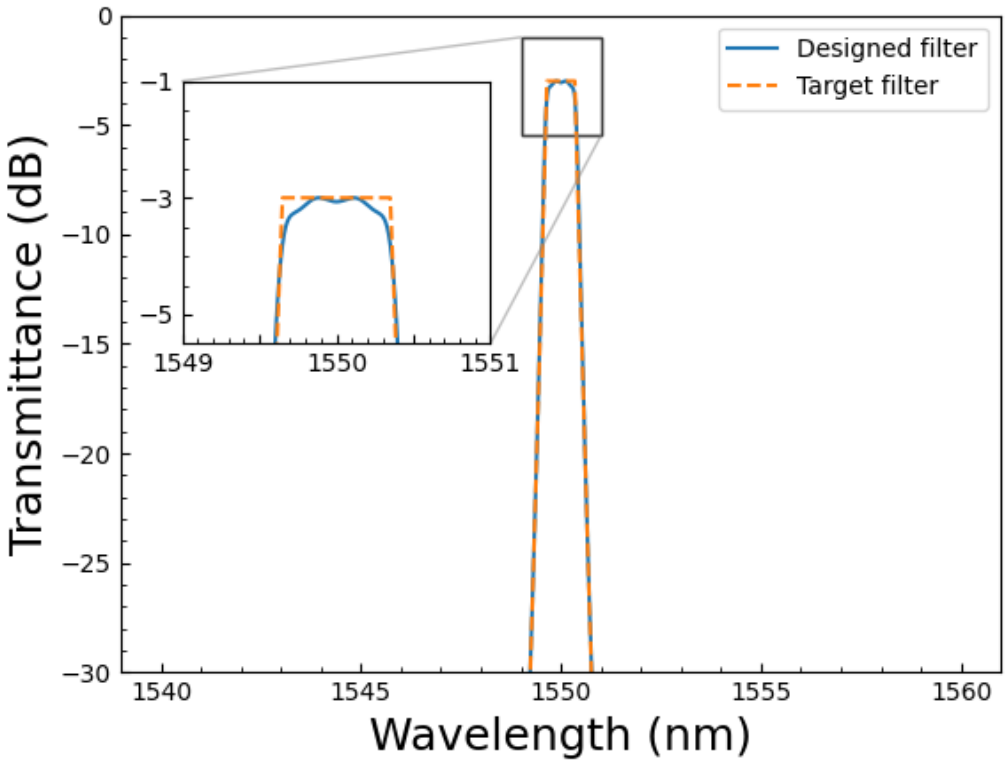


図 4.4 異径 6 次 MRR の設計結果（評価値：7.46）

表 4.5 図 4.4 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.32 | |
| 1 | 0.04 | 55.0 µm |
| 2 | 0.04 | 82.4 µm |
| 3 | 0.04 | 82.4 µm |
| 4 | 0.06 | 55.0 µm |
| 5 | 0.23 | 55.0 µm |
| 6 | 0.71 | 55.0 µm |

表 4.6 図 4.4 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 3 dB | 0.8 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | 3.00 dB | 0.82 nm | 0.06 dB | 31.0 dB | 0.80 |

異径 6 次 MRR においての設計結果は，トップの部分が若干丸みを帯びてしまっているが，設計目標に近い結果を得られた．異径 4 次 MRR に比べてクロストークも 30dB より大きいため，フィルタの性能としては十分といえる．

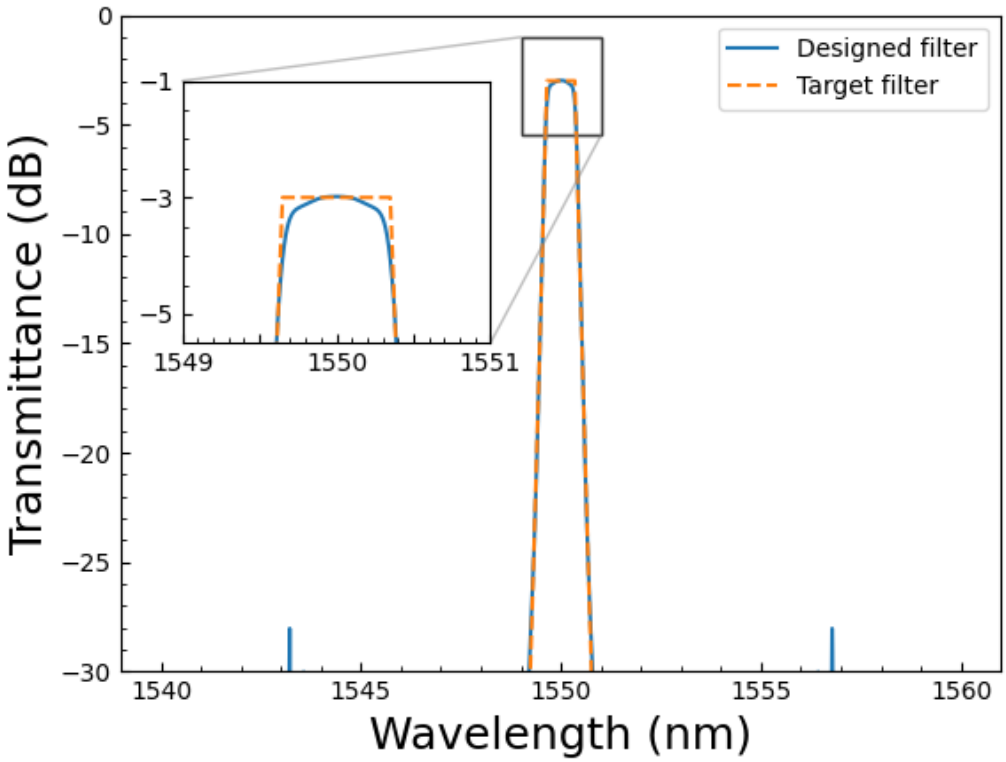


図 4.5 異径 8 次 MRR の設計結果（評価値：6.20）

表 4.7 図 4.5 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.4 | |
| 1 | 0.05 | 55.0 μm |
| 2 | 0.04 | 82.4 μm |
| 3 | 0.05 | 82.4 μm |
| 4 | 0.12 | 82.4 μm |
| 5 | 0.18 | 82.4 μm |
| 6 | 0.13 | 55.0 μm |
| 7 | 0.25 | 55.0 μm |
| 8 | 0.78 | 55.0 μm |

表 4.8 図 4.5 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 3 dB | 0.8 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | 2.98 dB | 0.80 nm | 0 dB | 25.0 dB | 0.80 |

異径 8 次 MRR においての設計結果は，設計目標に近い結果が得られた．異径 6 次 MRR 同様の設計結果といえるが，トップの形状は異径 6 次 MRR よりもフラットであるのに対し，クロストークがごく一部分だけ飛び出てきてしまっている．

以上の異径 2,4,6,8 次 MRR の設計結果について，従来の異径高次直列 MRR では，両サイドに同径のリングをまとめた並び方が良いとされてきたが，今回の設計結果はこれまでと異なるものが得られた．例として上記の図 4.5 のようなリング次数が 8 次のものならば A-A-A-A-B-B-B-B（A=82.4 μm, B=55.0 μm）というものである．しかし，本研究の設計結果としては表 4.4 では B-A-A-B-B-B（A=82.4 μm, B=55.0 μm），表 4.7 では B-A-A-A-A-B-B-B（A=82.4 μm, B=55.0 μm）のように ，片側の同径リング群に対してもう片方のリングが端に入ってくるような並び方をとっており，これが設計結果として最も良い評価値を得た．これら従来の並び方と今回の設計結果による並び方については図 4.5-8 及び表 4.7-14 で比較した．まずは上記の図 4.5 に対して従来の並び方との比較を行うため，同様の設計目標を与えた際の設計を行った．その結果を図 4.6 及び表 4.9-10 に示す．

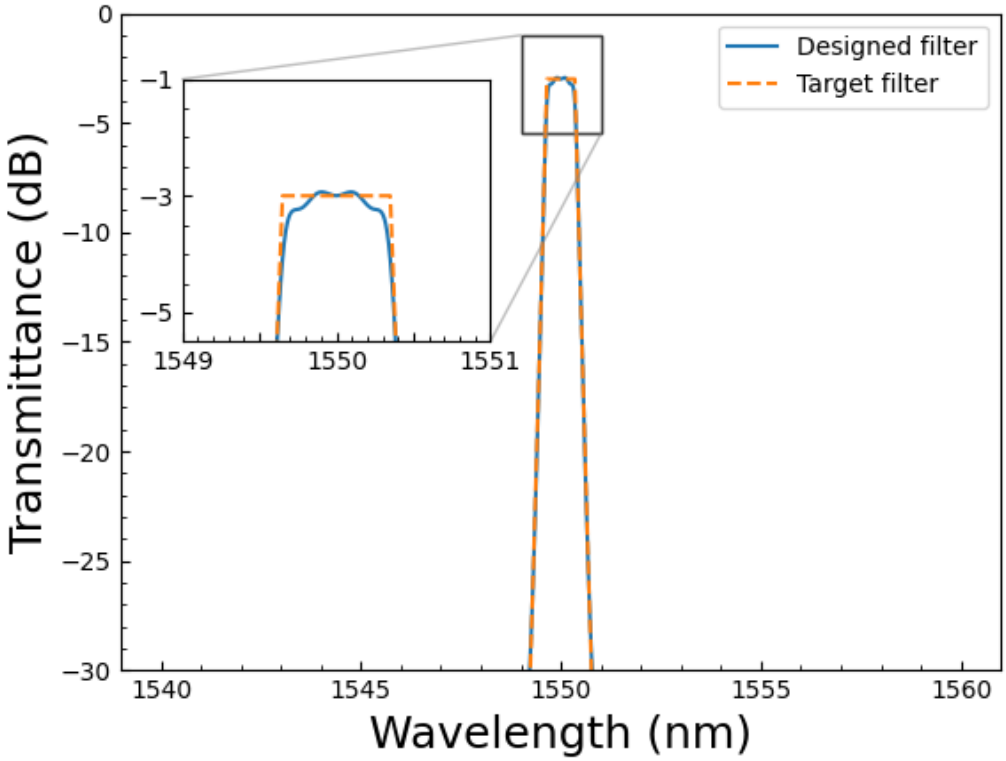


図 4.6 リングの並び方を A-A-A-A-B-B-B-B とした異径 8 次 MRR の設計結果（評価値：7.20）

表 4.9 図 4.6 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.4 | |
| 1 | 0.06 | 82.4 μm |
| 2 | 0.04 | 82.4 μm |
| 3 | 0.06 | 82.4 μm |
| 4 | 0.13 | 82.4 μm |
| 5 | 0.15 | 55.0 μm |
| 6 | 0.12 | 55.0 μm |
| 7 | 0.2 | 55.0 μm |
| 8 | 0.66 | 55.0 μm |

表 4.10 図の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 3 dB | 0.8 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | 2.93 dB | 0.8 nm | 0.06 dB | 38.0 dB | 0.8 |

従来の並び方における異径 8 次 MRR の設計結果は図 4.5 の結果とほとんど変わらず，設計目標にほとんど一致する結果となった．これら二つの違いを比較すると，従来の並び方は挿入損失が僅かだが 0.05 dB 改善し，クロストークに関しては 13 dB も向上している．しかしトップの形状はフラットではなくリプルが現れている．よって今回の設計結果における結論として，挿入損失がほぼ誤差の範囲である事とトップの形状が良いことから，トップに関わる部分は新しい並び方の方が僅かに優れており，クロストークに関わる部分は従来の並び方の方が優れているといえる．しかし，フィルタとしての重要な要素はトップの部分であり，クロストークは最低限あれば問題ないため，僅かに新しい並び方の方が上回っていると考えられる．しかし，これらの結果から大きな差が見当たらないため，リング周長を固定したうえで設計目標の挿入損失を 2 dB に変更したうえで再度設計を行った．図 4.7-8 及び表 4.11-14 にその結果を示す．

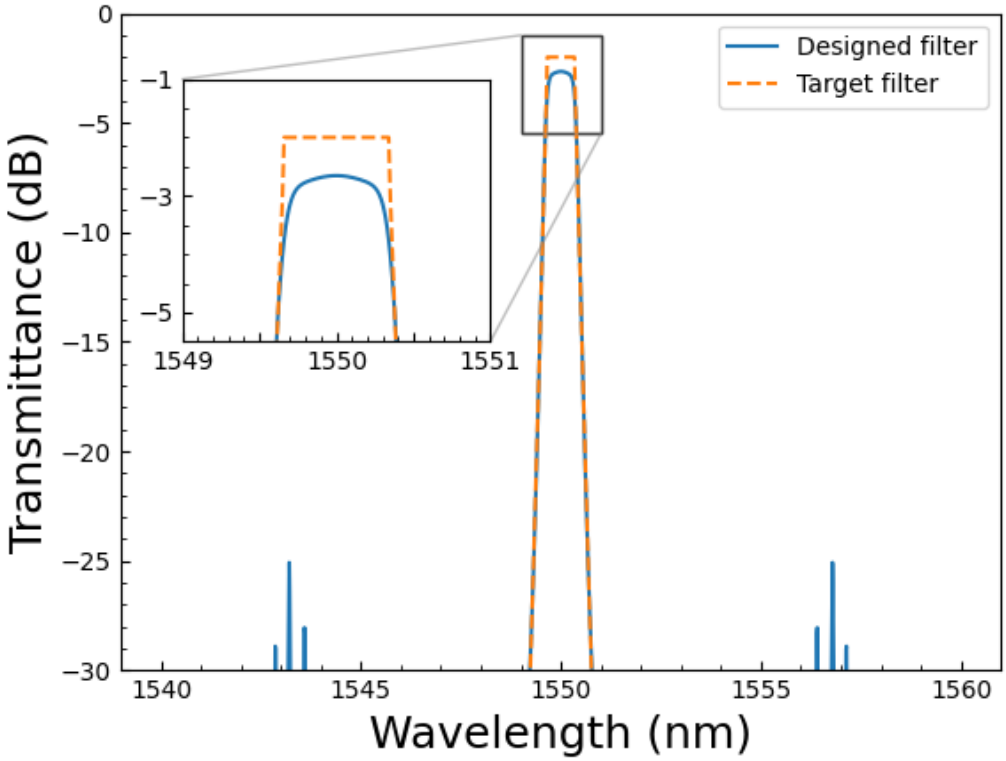


図 4.7 リングの並び方を B-A-A-A-A-B-B-B とした異径 8 次 MRR の設計結果（評価値：29.0）

表 4.11 図 4.7 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.51 | |
| 1 | 0.07 | 55.0 μm |
| 2 | 0.04 | 82.4 μm |
| 3 | 0.05 | 82.4 μm |
| 4 | 0.22 | 82.4 μm |
| 5 | 0.38 | 82.4 μm |
| 6 | 0.15 | 55.0 μm |
| 7 | 0.2 | 55.0 μm |
| 8 | 0.76 | 55.0 μm |

表 4.12 図 4.7 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 2 dB | 0.8 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | 2.65 dB | 0.8 nm | 0 dB | 22.4 dB | 0.8 |

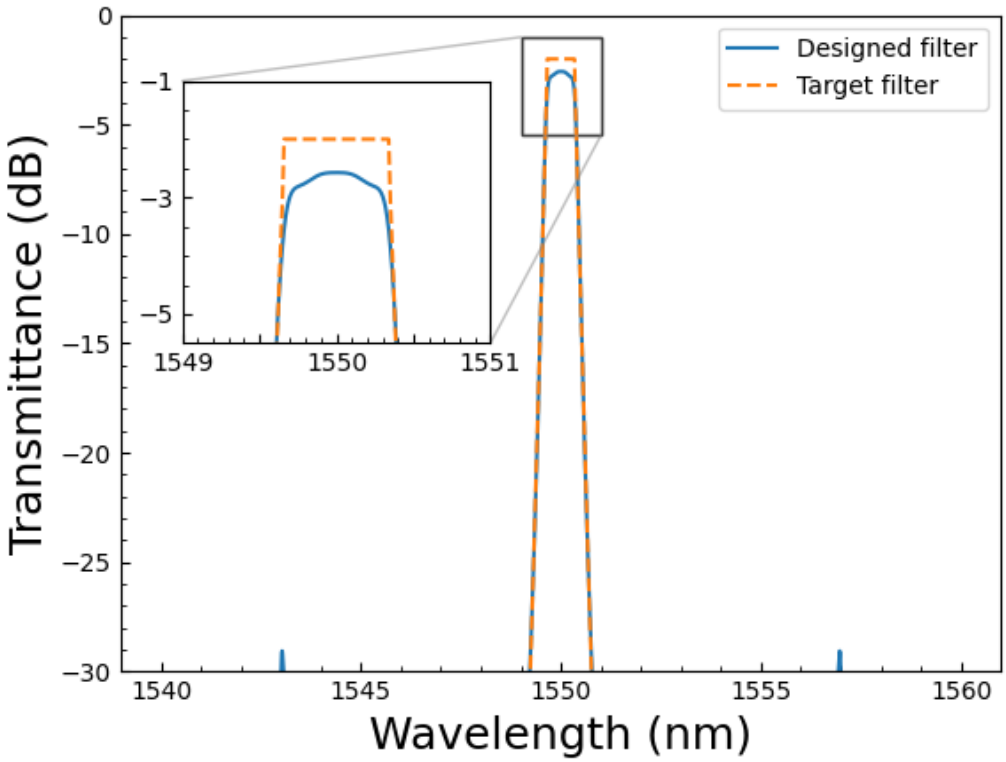


図 4.8 リングの並び方を A-A-A-A-B-B-B-B とした異径 8 次 MRR の設計結果（評価値：34.1）

表 4.13 図 4.8 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.45 | |
| 1 | 0.06 | 82.4 μm |
| 2 | 0.04 | 82.4 μm |
| 3 | 0.06 | 82.4 μm |
| 4 | 0.18 | 82.4 μm |
| 5 | 0.28 | 55.0 μm |
| 6 | 0.22 | 55.0 μm |
| 7 | 0.34 | 55.0 μm |
| 8 | 0.82 | 55.0 μm |

表 4.14 図 4.8 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 2 dB | 0.8 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | 2.57 dB | 0.8 nm | 0 dB | 26.5 dB | 0.8 |

結果として，設計目標を変えた場合においても大きな差は生じなかったが，比較を行うと先ほどと同様の差が見受けられる．これら二つの違いを比較すると，従来の並び方は挿入損失が僅かだが 0.08 dB 改善し，クロストークに関しては 4.1 dB も向上している．図 4.6-7 との違いは従来の並び方にリプルが現れていない点だが，よりフラットなのは新しい並び方である．よって，従来の並び方と今回設計した並び方では若干の差でトップを重視した設計を行うのであれば後者，クロストークも含めバランスよく設計するのであれば前者が優れていると結論付ける．

次に，設計の柔軟性を示すために異径 6 次 MRR の設計目標を変えて設計を行った．まず，挿入損失を 0 dB, 1 dB, 2 dB に変えた設計結果を図 4.9-11 及び表 4.15-20 に示す．最後に，他にも一般的に用いられる 3dB 波長帯域幅として 0.4 nm, 1.6 nm に変えた設計結果を図 4.12-13 及び表 4.21-26 に示す．

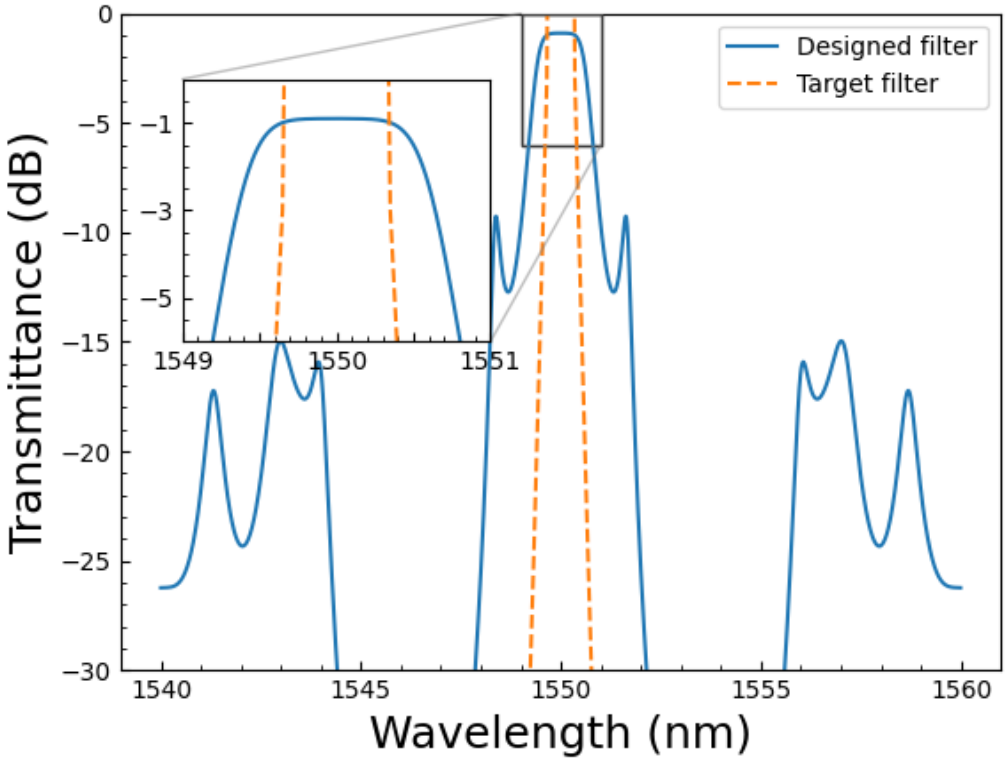


図 4.9 異径 6 次 MRR で挿入損失 0 dB を目標に設計した結果（評価値：7644）

表 4.15 図 4.9 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.75 | |
| 1 | 0.29 | 55.0 µm |
| 2 | 0.46 | 55.0 µm |
| 3 | 0.49 | 55.0 µm |
| 4 | 0.2 | 82.4 µm |
| 5 | 0.3 | 82.4 µm |
| 6 | 0.87 | 82.4 µm |

表 4.16 図 4.9 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 0 dB | 0.8 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | 0.90 dB | 1.40 nm | 0 dB | 8.4 dB | 0.65 |

挿入損失を 0 dB として設計を行った結果，3dB 波長帯域幅はほぼ 2 倍でクロストークは

8.4 dB という実用的ではない設計目標とは程遠いフィルタが得られた．原因として，挿入損失 0 dB というものが物理的に不可能なものであるのにもかかわらず，それに近づけるアルゴリズムであるため全体的な概形が崩れてしまったからだと考えられる．しかし，異径 6 次 MRR において挿入損失 0.9 dB という数値が得られていることからかなりの低損失であるといえる．

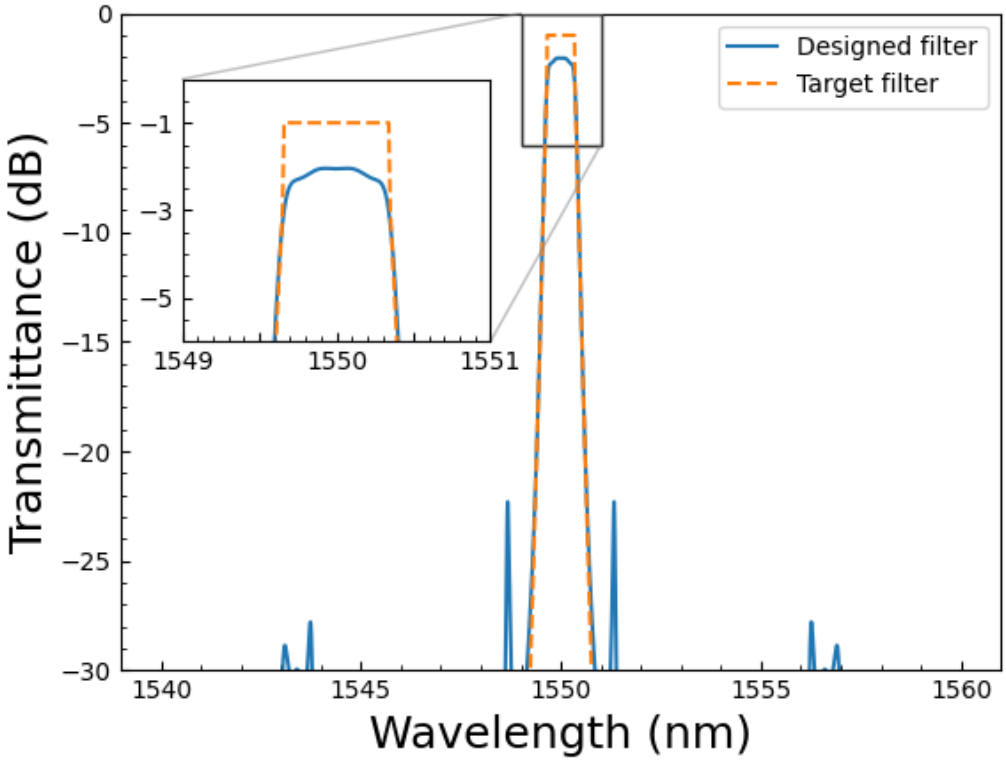


図 4.10 異径 6 次 MRR で挿入損失 1 dB を目標に設計した結果（評価値：95.6）

表 4.17 図 4.10 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.33 | |
| 1 | 0.08 | 55.0 µm |
| 2 | 0.37 | 55.0 µm |
| 3 | 0.31 | 55.0 µm |
| 4 | 0.06 | 82.4 µm |
| 5 | 0.06 | 82.4 µm |
| 6 | 0.43 | 82.4 µm |

表 4.18 図 4.10 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 1 dB | 0.8 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | 2.04 dB | 0.78 nm | 0.01 dB | 20.2 dB | 0.79 |

挿入損失を 1 dB として設計を行った結果，フィルタとしては実用的だが挿入損失は目標値とは約 1 dB の差が生じてしまった．原因としては，図 4.9 の結果と同様の理由で，他の設計目標を満たしつつ，挿入損失 1 dB を実現するようなフィルタが物理的に不可能なものであるからと考えられる．また，クロストークが 20 dB であることや 3dB 波長帯域幅が目標とする設計値より小さくなっていることから，あまりフィルタで用いるには適さない設計結果となってしまった．

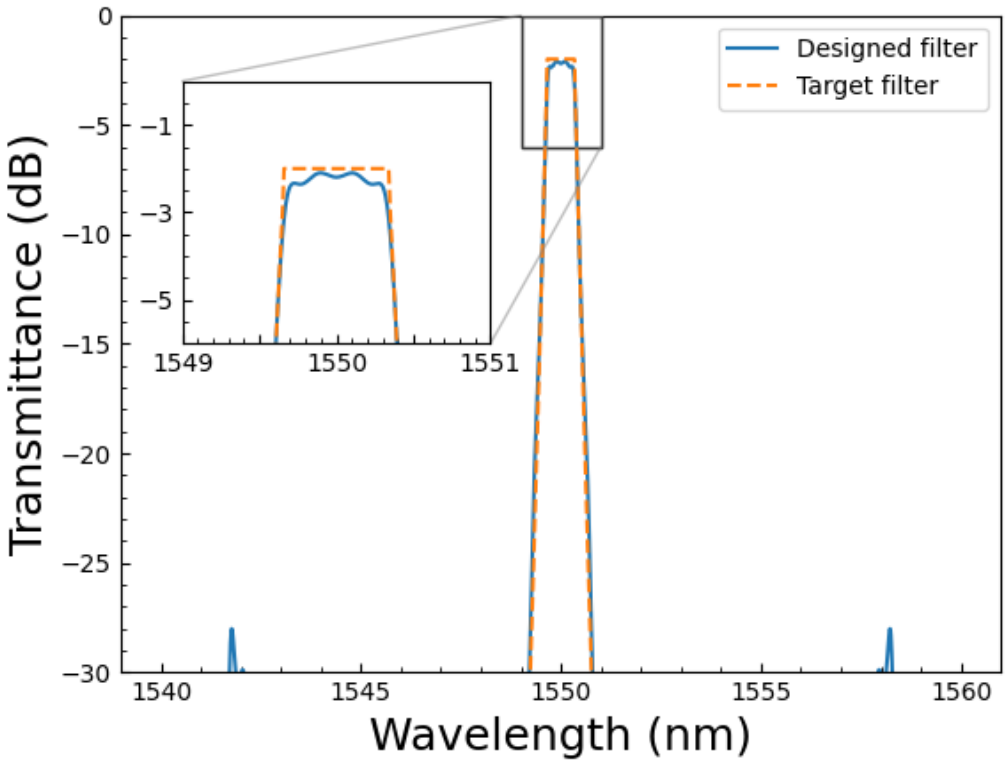


図 4.11 異径 6 次 MRR で挿入損失 2 dB を目標に設計した結果（評価値：17.6）

表 4.19 図 4.11 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.47 | |
| 1 | 0.12 | 55.0 μm |
| 2 | 0.07 | 55.0 μm |
| 3 | 0.07 | 55.0 μm |
| 4 | 0.11 | 137.4 μm |
| 5 | 0.15 | 137.4 μm |
| 6 | 0.53 | 137.4 μm |

表 4.20 図 4.11 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 2 dB | 0.8 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | 2.10 dB | 0.78 nm | 0.25 dB | 25.9 dB | 0.79 |

挿入損失を 2 dB として設計を行った結果, クロストークに 4 dB ほど差が生じているが, 全体的に設計目標を満たした設計結果が得られた. この設計結果の特性は本研究において最もバランスよく高性能に設計出来たもの一つであり, この結果から 3dB 波長帯域幅を 0.8 nm として設計するのであれば挿入損失は 2 dB が上限であると考えられる.

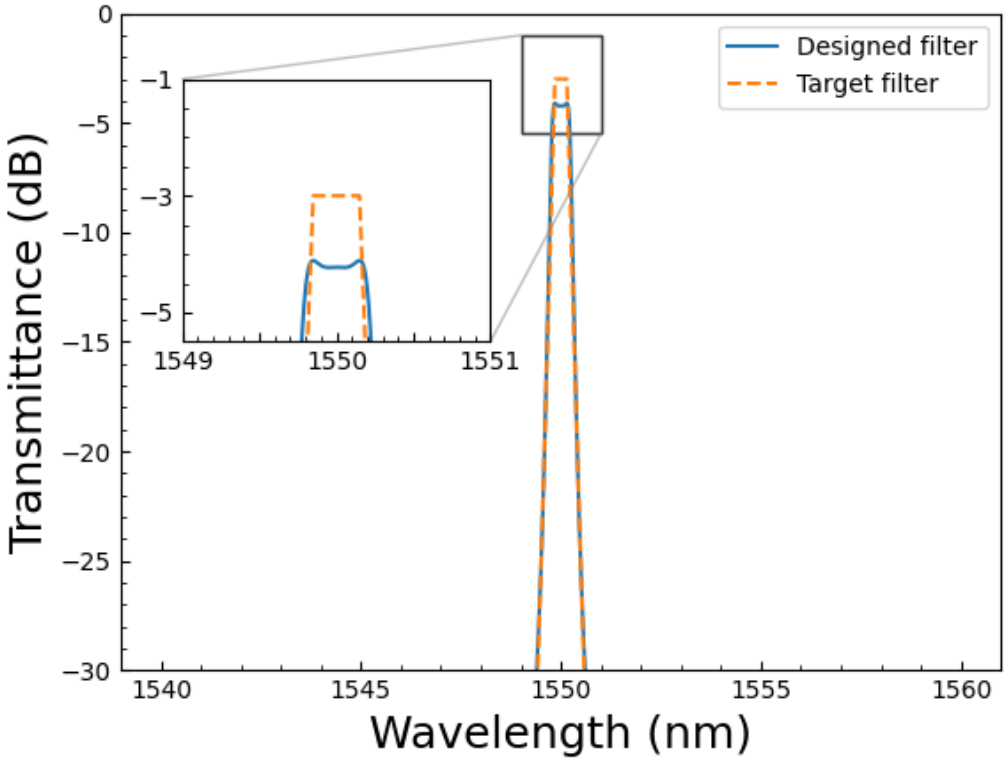


図 4.12 異径 6 次 MRR で 3 dB 波長帯域幅 0.4 nm を目標に設計した結果（評価値：22.6）

表 4.21 図 4.12 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.36 | |
| 1 | 0.02 | 82.4 μm |
| 2 | 0.02 | 82.4 μm |
| 3 | 0.04 | 82.4 μm |
| 4 | 0.15 | 55.0 μm |
| 5 | 0.16 | 55.0 μm |
| 6 | 0.58 | 55.0 μm |

表 4.22 図 4.12 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 3 dB | 0.4 nm | 0 dB | 30 dB | 0.8 |
| 設計結果 | 4.10 dB | 0.52 nm | 0.11 dB | 30.2 dB | 0.76 |

3dB 波長帯域幅を 0.4 nm として設計を行った結果，目標値に対し 3dB 波長帯域幅が 1.3 倍，挿入損失は 1 dB の差があるものが得られた．このことから，より狭域なものを設計するためにはリング次数がより高いものである必要があるということが分かる．また，図 4.3 の設計結果においてトップの部分とクロストークはトレードオフの関係にあると結論付けたが，今回の結果から挿入損失及び形状と 3dB 波長帯域幅においてもトレードオフの関係にあると考えられる．

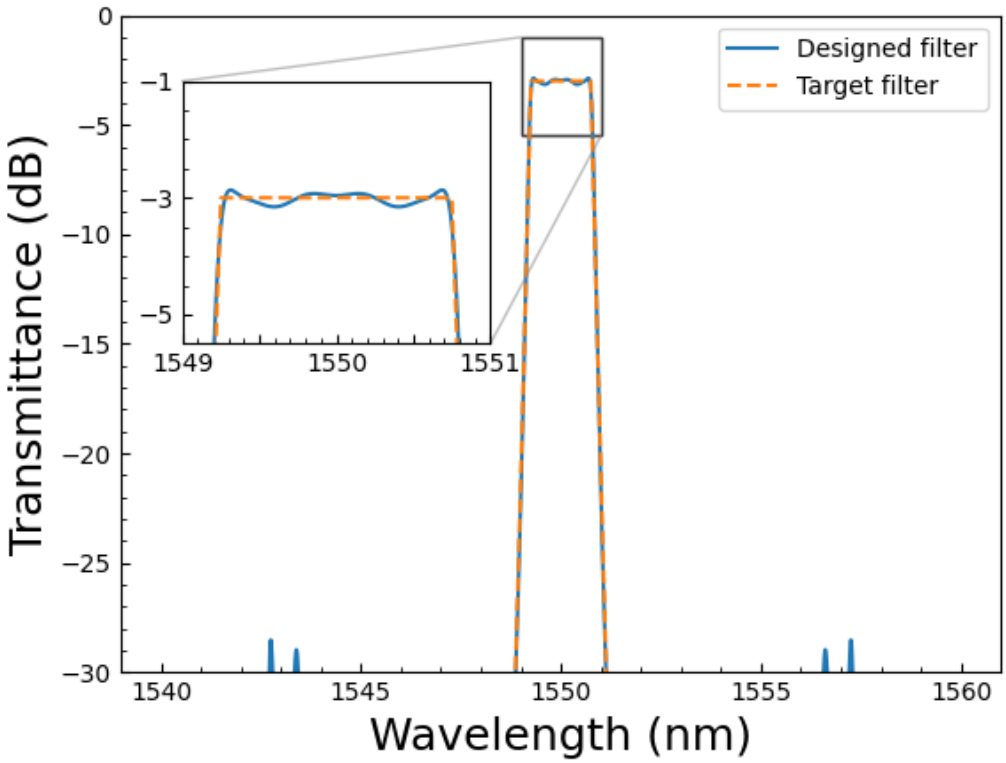


図 4.13 異径 6 次 MRR で 3 dB 波長帯域幅 1.6 nm を目標に設計した結果（評価値：8.8）

表 4.23 図 4.13 の設計値

| $M$ | $K_M$ | $L_M$ |
|---|---|---|
| 0 | 0.75 | |
| 1 | 0.21 | 55.0 µm |
| 2 | 0.09 | 55.0 µm |
| 3 | 0.1 | 55.0 µm |
| 4 | 0.15 | 82.4 µm |
| 5 | 0.2 | 82.4 µm |
| 6 | 0.28 | 82.4 µm |

表 4.24 図 4.13 の設計目標と設計結果の特性

| | 挿入損失 | 3dB 波長帯域幅 | リプル | クロストーク | シェイプファクター |
|---|---|---|---|---|---|
| 目標値 | 3 dB | 1.6 nm | 0 dB | 30 dB | 0.9 |
| 設計結果 | 2.86 dB | 1.62 nm | 0.28 dB | 25.6 dB | 0.91 |

3dB 波長帯域幅を 1.6 nm として設計を行った結果，設計目標を十分に満たし，これまでの設計結果で最も形状が箱型であるものが得られた．また設計目標であるシェイプファクターも変更されていることに関しては，本章冒頭で説明している．

# 第 5 章　結論

本研究では近年の光通信量の増大を解決するフォトニックネットワークに用いる経路制御において重要なデバイスである高次直列結合マイクロリング共振器波長フィルタの設計手法について，深層学習を用いた最適化手法を提案した．

第 1 章ではフォトニックネットワークに関わる研究背景についてまとめ，マイクロリング共振器について小型で設計性が高いという利点と，設計性の困難さについて説明し，それに対して本研究の目的である深層学習を用いた解決法を提案した．

第 2 章ではマイクロリング共振器の基本原理とその特性について説明し，異なるリング周長を用いることで共振波長間隔FSRが増大するバーニャ効果について説明した．また，本研究で用いる深層学習に用いている活性化関数及び損失関数について，及び深層学習の基本原理についてまとめた．

第 3 章では深層学習の初期条件及びデバイス依存の条件について触れ，マイクロリング共振器に対してどのように適用していくかのフローチャートを示した．また，深層学習を用いるうえでのハイパーパラメータの決定方法の検証結果を示した．深層学習のみでは正確な予測ができないため，これを候補出しとして扱うための補正手法について，原理及びそのアルゴリズムを示した．これによって設計値の高精度な予測を可能にした．そして，評価方法で積分値を用いている理由について説明し，その計算手法について示した．

第 4 章では異径 2,4,6,8 次リング MRR の設計結果を示した．各次数に対して同様の設計目標を与え，リング次数が変わることによって生じるフィルタへの影響を考察した．異径 8 次リング MRR においてリングの並び方によって生じるフィルタへの影響を考察した．この設計手法の柔軟性の確認のために異径 6 次リングに対していくつかの異なる設計目標を与え，その設計結果と設計目標の一部分を変えることによって生まれる影響についての考察を示した．結果として，所望の特性を得るための最適なリング次数はその特性によって異なるという事，トップとクロストーク，3dB 波長帯域幅と形状及び挿入損失はトレードオフの関係にあるという事，従来の並び方はバランスよく設計できるが，トップの部分の形状がよりよいリングの並び方は他にあるという事の三つを結論付けた．

# 謝辞

本研究の遂行にあたり，指導教員として多大な助言とご指導をしていただいた荒川太郎教授に深く感謝致します.

合同輪講では多くのご指導及びご指摘をしていただきました馬場俊彦教授，関口康爾教授，大矢剛嗣准教授，西島喜明准教授，水野洋輔准教授に深く感謝致します.

研究室内での同グループのミーティングでは多くの意見をくださった長屋陸人氏，長沼遼人氏に深く感謝致します.

研究室の同期として，五十嵐純平氏，大森百香氏，内田悠介氏，関口岳氏，田中優次氏に深く感謝致します.

研究室の後輩として，青見美河氏，衛藤雄大氏，中澤遼太郎氏，柳田玲氏に深く感謝致します.

# 参考文献


[1] S. Fukazawa, F. Ono, and T. Baba, "Compact arrayed-waveguide grating demultiplexer using Si photonic wire waveguide," Journal of Applied Physics, vol. 43, no. 5B, pp. SK1014 (2004).

[2] Y. Yanagase, S. Suzuki, Y. Kokubun and Sai Tak Chu, "Box-like filter response and expansion of FSR by a vertically triple coupled microring resonator filter," in Journal of Lightwave Technology, vol. 20, no. 8, pp. 1525-1529 (2002).

[3] Y. Goebuchi, T. Kato and Y. Kokubun, "Multiwavelength and Multiport Hitless Wavelength-Selective Switch Using Series-Coupled Microring Resonators," in IEEE Photonics Technology Letters, vol. 19, no. 9, pp. 671-673 (2007).

[4] S. Xiao, M. H. Khan, H. Shen, and M. Qi, "A highly compact third-order silicon microring add-drop filter with a very large free spectral range, a flat passband and a low delay dispersion." Optics express, vol. 15, no. 22, pp. 14765-14771 (2007).

[5] C. Li, C. Zhou, and A. W. Poon, "Silicon microring carrier-injection-based modulators/switches with tunable extinction ratios and OR-logic switching by using waveguide cross-coupling." Optics express, vo. 15, no. 8, pp. 5069-5076 (2007).

[6] A. M. Prabhu, H. L. Liew, and V. Van, "Generalized parallel-cascaded microring networks for spectral engineering applications." JOSA B, vol. 25, no. 9, pp. 1505-1514 (2008).

[7] H. L. Lira, S. Manipatruni, and M. Lipson, "Broadband hitless silicon electro-optic switch for on-chip optical networks." Optics Express, vol. 17, no. 25, pp. 22271-22280 (2009).

[8] C. Chaichuay, P. P. Yupapin and P. Saeung, "The serially coupled multiple ring resonator filters and Vernierеffect", Appl. Opt., vol. 39, no. 1, pp. 175-194 (2009).

[9] R. Boeck, N. A. F. Jaeger, N. Rouger, and L. Chrostowski, "Series-coupled silicon racetrack resonators and the Vernier effect: theory and measurement." Opt. Express, vol. 18, no. 24, pp. 25151-25157 (2010).

[10] T. Makino, T. Gotoh, R. Hasegawa, T. Arakawa, and Y. Kokubun, "Microring Resonator Wavelength Tunable Filter Using Five-Layer Asymmetric Coupled Quantum Well." J. Lightwave Technol, vol. 29, pp. 2387-2393 (2011).

[11] A. P. Masilamani and V. Van, "Design and realization of a two-stage microring ladder filter in silicon-on-insulator." Opt. Express, vol. 20, no. 22, pp. 24708-24713 (2012).

[12] H. Ikehara, T. Goto, H. Kamiya, T. Arakawa, and Y. Kokubun, "Hitless wavelength-selective switch based on quantum well second-order series-coupled microring resonators." Opt. Express, vol. 21, no. 5, pp. 6377-6390 (2013).

[13] H. Kamiya, T. Goto, H. Ikehara, R. Katouf, T. Arakawa, and Y. Kokubun, "Hitless wavelength-selective switch with quadruple series-coupled microring resonators using multiple-quantum-well waveguides." Opt. Express, vol. 21, no. 18, pp. 20837-20850 (2013).

[14] K. Padmaraju, L.-W. Luo, X. Zhu, M. Glick, R. Dutt, M. Lipson, and K. Bergman, "Wavelength Locking of a WDM Silicon Microring Demultiplexer using Dithering Signals." Optical Fiber Communication Conf., Tu2E.4 (2014).

[15] I. S. Amiri, S. E. Alavi, M. R. K. Soltanian, N. Fisal, A. S. M. Supa'at, and H. Ahmad, "Increment of Access Points in Integrated System of Wavelength Division Multiplexed Passive Optical Network Radio over Fiber." Scientific reports, vol. 5, pp. 11897 (2015).

[16] L.-W. Luo, N. Ophir, C. P. Chen, L. H. Gabrielli, C. B. Poitras, K. Bergmen, and M. Lipson, "WDM-compatible mode-division multiplexing on a silicon chip." Nat Commun, vol. 5, pp. 3069 (2014).

[17] G. Barbarossa, A. Matteo, and M. Armenise, "Theoretical analysis of triple-coupler ring-based optical guided-wave resonator." Journal of Lightwave Technology, vol. 13, pp. 148-157 (1995).

[18] B. E. Little, S. T. Chu, H. A. Haus, J. Foresi and J.-P. Laine, "Microring resonator channel dropping filters," Journal of Lightwave Technology, vol. 15, no. 6, pp. 998-1005 (1997).

[19] Z. Peng, T. Komatsubara, M. Yamauchi, and T. Arakawa, "Design of a series-coupled microring resonator wavelength filter using the digital filter design method." J. Opt. Soc. Am. B, vol. 38, no. 10, pp. 2837-2846 (2021).

[20] Y. Udagawa and T. Arakawa, "Design of high-order series-coupled microring resonator wavelength filter with differential evolution method," 2021 26th Microoptics Conference (MOC), Hamamatsu, Japan, pp. 1-2 (2021).
[21] T. Asano and S. Noda. "Optimization of photonic crystal nanocavities based on deep learning." Optics express, vol. 26, no. 25, pp. 32704-32717 (2018)
[22] A. V. Saetchnikov, E. A. Tcherniavskaia, V. A. Saetchnikov, and A. Ostendorf, "Intelligent Optical Microresonator Imaging Sensor for Early Stage Classification of Dynamical Variations." Advanced Photonics Research, vol. 2 (2021).
[23] B. Yang, M. Carotta, G. Faglia, M. Ferroni, V. Guidi, G. Martinelli, and G. Sberveglieri, "Quantification of H2S and NO2 using gas sensor arrays and an artificial neural network." Sens. Actuators B, vol. 43, pp. 235-238 (1997).
[24] D. Hu, C.-l. Zou, H. Ren, J. Lu, Z. Le, Y. Qin, S. Guo, C. Dong, and W. Hu, "Multi-parameter sensing in a multimode self-interference micro-ring resonator by machine learning." Sensors, vol. 20, no. 3, pp. 709 (2020).
[25] T. Zahavy, A. Dikopoltsev, D. Moss, G. I. Haham, O. Cohen, S. Mannor, and M. Segev, "Deep learning reconstruction of ultrashort pulses." Optica, vol. 5, no. 5, pp. 666-73 (2018).
[26] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A survey of optimization methods from a machine learning perspective." IEEE Trans. Cybernetics, vol. 50, no. 8, pp. 3668-81 (2020).
[27] T. Tanimura, T. Hoshida, K. Shiota, E. Katayama, T. Kato, S. Watanabe, and H. Morikawa, "Deep Learning Based Optical Monitoring toward Optical Network Automation", IEICE Trans. B, vol. J101-B, no. 12, pp. 1014-1025 (2018)
[28] M. S. Laleh, M. Razaghi, O. Jafari, and H. Bevrani, "Performance optimization of an optical filter based on serially coupled microring resonators using a fuzzy logic system." Optical Engineering, vol. 58, no. 2, pp. 026115 (2019).
[29] T. J. Zimmerling, Y. Ren, H. Ngo, and V. Van, "Fast thermo-optic optimization of high-order SOI microring optical filters by method of gradient descent." Silicon Photonics XV, SPIE, vol. 11285, pp. 234-41 (2020).
[30] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning." Nature, vol. 521, no. 7553, pp.436-44 (2015).
[31] D. P. Kingma, and Ba. Jimmy, "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
[32] Elijah Polak, ed. "Optimization: algorithms and consistent approximations." Springer Science & Business Media, vol. 124 (2012).

# 本研究に関する発表

国際会議

[1] T, Fukuyo, and T. Arakawa. “Design of high-order series-coupled microring resonator wavelength filter with deep learning,” The 11th International Symposium on Materials Science and Surface Technology (MSST), PB-11, Japan (2022).

[2] T, Fukuyo, and T. Arakawa. “Optimization method using deep learning for highly efficient design of high-order series-coupled microring resonator wavelength filters.” The 12th International Symposium on Materials Science and Surface Technology (MSST), PB-09, Japan (2023).

国内会議

[1] 福與太一，荒川太郎，”深層学習を用いた高次直列結合マイクロリング波長フィルタの設計”，電子情報通信学会ソサエティ大会，C-3/4-38,（2022）

# 付録 作成・使用したソースコード

以下に本研究で作成・使用したソースコードを載せておく.

```
import matplotlib.pyplot as plt
import numpy as np

def graph_cretate(axis,dataset,nameset): #dataset は配列  axis は x 軸
nameset は配列で凡例用の名前  L と s は実験用なので削除予定  一枚表示 or 二枚重ねる
用の関数です
    if len(dataset) == 1:
        _onegraph_create(axis,dataset[0],nameset[0])
        _onetopgraph_create(axis,dataset[0],nameset[0])
    elif len(dataset) == 2:
        _twograph_create(axis,dataset[0],dataset[1],nameset[0],nameset[1
])
        _twotopgraph_create(axis,dataset[0],dataset[1],nameset[0],namese
t[1])


def _onegraph_create(axis,data1,name1):      #一つのグラフのみ表示
    plt.rcParams["xtick.direction"]="in"     #目盛り内向き
    plt.rcParams["ytick.direction"]="in"     #目盛り内向き
    plt.plot(axis,data1,label=name1)         #プロット
    plt.xlabel("Wavelength (nm)",fontsize=13)        #x 軸ラベル
    plt.ylabel("Transmittance (dB)",fontsize = 13)   #y 軸ラベル
    plt.ylim(-60,0)                                  #y 軸表示範囲-60~0dB
    plt.xticks(np.arange(1540,1565,5))               #x 軸目盛りの表示位置を
設定
    plt.minorticks_on()                              #補助目盛オン
    plt.legend(bbox_to_anchor=(1,1),loc="upper right")  #凡例を右上に表示
loc だけでなく位置を変えたいなら anchor も変える必要ありなため，調べてください
    plt.show()                                           #グラフ表示

def _onetopgraph_create(axis,data1,name1):  #一つのグラフをトップの部分を拡大
して表示
    plt.rcParams["xtick.direction"]="in"     #目盛り内向き
```

```
    plt.rcParams["ytick.direction"]="in"    #目盛り内向き
    plt.plot(axis,data1,label=name1)        #プロット
    plt.xlabel("Wavelength (nm)",fontsize = 13)     #x 軸ラベル
    plt.ylabel("Transmittance (dB)",fontsize = 13)  #y 軸ラベル
    plt.ylim(-6,0)                                  #y 軸表示範囲-6~0dB
    plt.xlim(1549,1551)                             #x 軸表示範囲
1549nm~1551nm
    plt.xticks(np.arange(1549,1551.25,0.5))         #x 軸目盛りの表示位置を
設定
    plt.minorticks_on()                             #補助目盛オン
    plt.legend(bbox_to_anchor=(1,1),loc="upper right")  #凡例を右上に表示
loc だけでなく位置を変えたいなら anchor も変える必要ありなため，調べてください
    plt.show()                                          #グラフ表示

def _twograph_create(axis,data1,data2,name1,name2):     #二つのグラフを重
ねて表示
    plt.rcParams["xtick.direction"]="in"    #目盛り内向き
    plt.rcParams["ytick.direction"]="in"    #目盛り内向き
    plt.plot(axis,data1,label=name1)        #一つ目のグラフをプロット
    plt.plot(axis,data2,label=name2)        #二つ目のグラフをプロット
    plt.xlabel("Wavelength (nm)",fontsize = 13)     #x 軸ラベル
    plt.ylabel("Transmittance (dB)",fontsize = 13)  #y 軸ラベル
    plt.ylim(-60,0)                                 #y 軸表示範囲-60~0dB
    plt.xticks(np.arange(1540,1565,5))              #x 軸目盛りの表示位置を
設定
    plt.minorticks_on()                             #補助目盛オン
    plt.legend(bbox_to_anchor=(1,1),loc="upper right")  #凡例を右上に表示
loc だけでなく位置を変えたいなら anchor も変える必要ありなため，調べてください
    plt.show()              #グラフ表示

def _twotopgraph_create(axis,data1,data2,name1,name2):
    plt.rcParams["xtick.direction"]="in"    #目盛り内向き
    plt.rcParams["ytick.direction"]="in"    #目盛り内向き
    plt.plot(axis,data1,label=name1)        #一つ目のグラフをプロット
    plt.plot(axis,data2,label=name2)        #二つ目のグラフをプロット
    plt.xlabel("Wavelength (nm)",fontsize = 13)     #x 軸ラベル
```

```
    plt.ylabel("Transmittance (dB)",fontsize = 13)      #y 軸ラベル
    plt.ylim(-6,0)                                      #y 軸表示範囲-6~0dB
    plt.xlim(1549,1551)                                 #x 軸表示範囲
1549nm~1551nm
    plt.xticks(np.arange(1549,1551.25,0.5))          #x 軸目盛りの表示位置を
設定
    plt.minorticks_on()                                 #補助目盛オン
    plt.legend(bbox_to_anchor=(1,1),loc="upper right")  #凡例を右上に表示
loc だけでなく位置を変えたいなら anchor も変える必要ありなため，調べてください
    plt.show()                  #グラフ表示
```

ソースコード 1：create_graph.py

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow as tf
import random
import os
from mymath import minus
from transfer_function import TransferFunction


#以下研究で用いていた初期設定
# L=np.array([82.4e-6,82.4e-6,55.0e-6,55.0e-6])
# number_of_rings = 4                                   #リング数
# n = 100000                                            #データ数
# data_K = np.array(read_K(n,number_of_rings))          #K の格納先  DNN の教師データ

# xaxis = np.arange(1540e-9,1560e-9,0.01e-9)            #シミュレーション範囲 1.54μm~1.56μm
# epochs = 200                                          #訓練回数
# batch_size = 1000                                     #学習時に一度に計算するデータ数

def DNN_K(L,number_of_rings,n,data_K,xaxis,epochs,batch_size):  #K を予測する DNN を作成する関数
    input_data = []                                     #DNN に用いるデータの仮格納先
    #データセットの用意
    for i in range(n):  #伝達関数と結合率Kの対応関係を学習させるため，X と Y でそれぞれ格納
        trans_data = TransferFunction(L,data_K[i],config={'center_wavelength':1550e-9,'eta':0.996,'n_eff':2.2,'n_g':4.4,'alpha':52.96})
```

```
        input_data.append(list(map(minus,trans_data.simulate(xaxis))))
#データを学習向けに加工している．伝達関数は基本負であるが，正の値の方が学習がうまくいくため map 関数で配列全体を正にしている
        train_X =
np.reshape(input_data,(n,len(xaxis)))                    #DNN に入力できるように配列の形を変えている
        train_Y =
np.reshape(data_K,(n,number_of_rings+1))                 #DNN に入力できるように配列の形を変えている

    #シード値の固定　再現性があることを示す
    seed = 1
    tf.random.set_seed(seed)
    np.random.seed(seed)
    random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)

    #モデルの構造　入力層 1 中間層 3 出力層 1　活性化関数は回帰を期待して線形(identity とも呼ぶ)
    model_DNN =
Sequential()                                                #Sequential という鎖型のモデルを作成
    model_DNN.add(Dense(len(xaxis), input_shape = (len(xaxis),),
activation='linear'))  #層は Dense という種類で，dense(入力数，形，活性化関数) input_shape の部分は試行錯誤の末によるもの
    model_DNN.add(Dropout(0.2))
            #過学習を抑制するために学習を 20%の確率で忘れさせる
    model_DNN.add(Dense(1250,activation='linear'))
    model_DNN.add(Dropout(0.2))
    model_DNN.add(Dense(1250,activation='linear'))
    model_DNN.add(Dropout(0.1))
    model_DNN.add(Dense(1250,activation='linear'))
    model_DNN.add(Dropout(0.1))
    model_DNN.add(Dense(number_of_rings+1, activation='linear'))
```

```
    model_DNN.compile(optimizer = Adam(lr=0.001),
loss="mean_squared_error", metrics="accuracy")    #オプティマイザーは Adam
で学習率は 0.1% 損失関数は平均二乗誤差

    #モデルの構造を表示する
    print(model_DNN.summary())

    #CallBacks  最も学習がうまくいったところを呼び戻して保存する
    filename = "MRR/data/DNN2_K_" + str(number_of_rings) + "_" + str(n)
+".h5"
    checkpoint = ModelCheckpoint(filepath=filename,
monitor="loss",verbose=0,save_best_only=True,save_weights_only=False,mo
de="min",period=1)   #loss を監視し，mode  min より loss が一番低いところで保存

    #学習開始
    history = model_DNN.fit(train_X, train_Y, batch_size=batch_size,
epochs=epochs, verbose=1, validation_split=0.01,callbacks=[checkpoint])

    #学習度のグラフ表示及び保存
    plt.plot(range(epochs),history.history["loss"],label="loss")
                      #loss をプロット
    plt.plot(range(epochs),history.history["val_loss"],label="val_loss"
)                       #val_loss をプロット
    plt.xlabel("epoch")
                     #x 軸ラベル
    plt.ylabel("loss")
                    #y 軸ラベル
    plt.ylim(0,20)
                    #y 軸の表示範囲
    plt.legend(bbox_to_anchor=(1,0),loc="lower
right")                                          #右下に凡例
    plt.savefig("MRR/data/DNN_K_" + str(number_of_rings) + "_" + str(n)
+ "_" + "figure.jpg")   #グラフ保存
    plt.show()
                    #グラフ表示
```

ソースコード 2：DNN_K.py

```
import numpy as np


def evaluate_graph(graph):                                  #まとめて出力するためのもの
    evaluate_data = [evaluate_insertion_loss(graph),evaluate_3dbband(graph)*0.01,evaluate_ripple(graph),evaluate_crosstalk(graph),evaluate_shape_factor(graph)]
    print("insertion loss = {} dB".format(evaluate_data[0]))
    print("3dbband = {} nm".format(evaluate_data[1]))
    print("ripple = {} dB".format(evaluate_data[2]))
    print("crosstalk = {} dB".format(evaluate_data))
    print("shape_factor = {}".format(evaluate_data[4]))
    return

def evaluate_insertion_loss(graph): #グラフ内の最大値を返す　挿入損失計算用
    return np.amax(graph)

def evaluate_ripple(graph): #3dB 波長帯域での極大と極小の差を返す リプル計算用(　グラフの真ん中がトップである前提)
    start_number = int(len(graph)/2)                        #真ん中にトップが来ると仮定している
    length = evaluate_3dbband(graph)                        #3db 波長帯域の長さを得る
    point = _local_maximum_and_minimum(graph[start_number - int(length/2)-1:start_number + int(length/2)])  #グラフ内の極大極小をとる x 座標を得る
    if not point[0][0] or not point[0][1]:                  #もしなければリプル 0 とする
        return 0
    ripple = max(point[1][0]) - min(point[1][1])            #リプル＝最も大きい極大値-最も小さい極小値
    return ripple

def evaluate_crosstalk(graph):  #3db 波長帯域外かつ右半分での傾きが正なっているところの y 座標の最大値を返す(グラフの真ん中がトップである前提)
```

```
    start_number = int(len(graph)/2)                        #真ん中にトップが来ると仮定している
    length = evaluate_3dbband(graph)                        #3db 波長帯域の長さを得る
    temp1 = 0                                               #仮格納先
    temp2 = 0
    for i in range(len(graph)):
        temp1 = temp2
        temp2 = graph[start_number + int(length/2) + i]     #真ん中の x 座標＋3dB 波長帯域の長さからスタート
        if temp2 > temp1:                                   ##初めての極大を検出
            point = start_number + int(length/2) + i
            break
    return np.amax(graph) - max(graph[point:])              #クロストークの計算をして返す

def evaluate_shape_factor(graph):                           #国分先生の定義通りの計算
    return _onedb(graph)/_tendb(graph)

def evaluate_3dbband(graph):        #最大値から-3 したところの交点の長さを出力
    y = evaluate_insertion_loss(graph)                      #最大値を格納
    start_number = int(len(graph)/2)                        #スタート位置
    for i in range(start_number):                           #最大値から-3 となる x 座標を得る
        if graph[start_number+i] - y + 3 <0:
            break
    return i*2  #対称性から中心から交点の距離*2 としている

def _local_maximum_and_minimum(graph):   #グラフ内の極大極小をとる x 座標を得る リプル計算用
    temp1 = 0
    temp2 = -100                        #y 座標の閾値
    max_point_x = []
    min_point_x = []
```

```
    max_point_y = []
    min_point_y = []
    clock = 0
    for i in range(len(graph)):
        temp1 = temp2
        temp2 = graph[i]
        if temp2 < temp1 and clock == 0:
            max_point_x.append(i)
            max_point_y.append(graph[i])
            clock = 1
        elif temp2 > temp1 and clock == 1:
            min_point_x.append(i)
            min_point_y.append(graph[i])
            clock = 0
    point_x = []
    point_x.append(max_point_x)
    point_x.append(min_point_x)
    point_y = []
    point_y.append(max_point_y)
    point_y.append(min_point_y)
    point = []
    point.append(point_x)
    point.append(point_y)

    return point    #返り値は 2*2 行列

def _onedb(graph):  #shapefactor 用  3db 波長帯域計算と同じ原理
    y = evaluate_insertion_loss(graph)
    temp = int(evaluate_3dbband(graph)/2)
    start_number = int(len(graph)/2)
    for i in range(temp):
        if graph[start_number-temp+i] - y + 1 <0:
            break
    return (temp - i)*2

def _tendb(graph):  #shapefactor 用  3db 波長帯域計算と同じ原理
```

```
    y = evaluate_insertion_loss(graph)
    temp = int(evaluate_3dbband(graph)/2)
    start_number = int(len(graph)/2)
    for i in range(temp):
        if graph[start_number-temp-i] - y + 10 <0:
            break
    return (temp +i)*2
```

ソースコード 3：evaluate_figure.py

```
import matplotlib.pyplot as plt
import numpy as np


def base_array(n,bottom):   #仮で用意する配列
    return [bottom]*n

def set_3db(center,db_length,top,data): #トップの部分を形成
    for i in range(db_length):
        data[int(center-int(db_length/2)+i)] = top

def set_cross(center,length,cross,data):   #サイドローブのトップを形成
    if cross == 0:
        return
    for i in range(length):
        data[int(center-int(length/2)+i+656)] = cross
        data[int(center-int(length/2)+i-655)] = cross

def set_cross2(length,cross2,data):     #2 つ目のサイドローブのトップを形成
    if cross2 == 0:
        return
    for i in range(int(length/2)):
        data[i] = cross2
        data[-i] = cross2

def
set_slope_fromcrosstocenter(center,db_length,cross_length,cross,data):#
トップとサイドローブの間のスロープ形成
    bottomtop = 67.5 #100 #300 #67.5  #トップとサイドローブの間の頂点の y 座標
    half_db_length = int(db_length/2)
    half_cross_length = int(cross_length/2)
    top = int((center + half_db_length + center-half_cross_length +
656)/2)
    a1 = (-3+bottomtop)/((center + half_db_length-top)**2)
    a2 = (cross+bottomtop)/((center - half_cross_length + 656 -
top)**2)
```

```
    for i in range (top - (center + half_db_length)):
        data [center + half_db_length + i] = a1*((center +
half_db_length + i - top)**2) - bottomtop
        data [center - half_db_length - i] = data [center +
half_db_length + i]
    for j in range (center - half_cross_length + 656 - top):
        data [top + j] = a2*((j)**2) - bottomtop
        data [2*center - top - j] = data [top + j]

def set_slope_fromcrosstocross(center,cross_length,cross,cross2,data):#
サイドローブ間のスロープ部分の形成
    bottomtop = 62.5 #100 #50 #105 #55 #62.5
    half_cross_length = int(cross_length/2)
    top = int((center + half_cross_length + 656 + 2001 -
cross_length)/2)
    a1 = (cross+bottomtop)/((center + half_cross_length + 656 -
top)**2)
    a2 = (cross2+bottomtop)/((2001 - half_cross_length - top)**2)
    for i in range (top - (center + half_cross_length + 611)):
        data [center + half_cross_length + 656 + i] = a1*((center +
half_cross_length + 656 + i - top)**2) - bottomtop
        data [center - half_cross_length - 656 - i] = data [center +
half_cross_length + 656 + i]
    for j in range (2001 - half_cross_length - top + 1):
        data [top + j] = a2*((j)**2) - bottomtop
        data [2*center - top - j] = data [top + j]

def set_ripple(center,position,ripple): #未実装，理想的なフィルタにおいて必要
性は無い
    return center

def
generate_figure(center,db_length,cross_length,cross,cross2,number_of_ar
ray,bottom,top):
    data = base_array(number_of_array,bottom)
    set_3db(center,db_length,top,data)
```

```
    set_cross(center,cross_length,cross,data)
    set_cross2(cross_length,cross2,data)
    set_slope_fromcrosstocenter(center,db_length,cross_length,cross,dat
a)
    set_slope_fromcrosstocross(center,cross_length,cross,cross2,data)

    return data

# 使い方

# axis = np.arange(1540,1560.01,0.01)
# plt.plot(axis,generate_figure(center = 1000,db_length =
80,cross_length = 80,cross = -44,cross2 = -39,number_of_array =
2001,bottom = -100,top = -3))
# plt.show()
```

ソースコード 4：generate_figure.py

```
import numpy as np
import math
import csv

def _floor_number(list,n):   #丸め込み関数
    for i in range (len(list)):
        list[i] = math.floor(list[i] * 10 ** n) / (10 ** n)
    return list

def
generate_L(n,number_of_rings):
   #L を生成して CSV ファイルとして保存
    file_name = "MRR/data/L" + str(number_of_rings) + "_" + str(n)
+".csv"
    with open(file_name,"w",newline="") as file:
        writer = csv.writer(file)
        for i in range(n):
            L = _floor_number(np.random.uniform(50e-6,150e-
6,number_of_rings),7) #50μ～150μ の範囲で小数 7 位までの乱数をリングの数だけ
nparray として生成
            writer.writerow(L)

def
generate_K(n,number_of_rings):
   #K を生成して CSV ファイルとして保存
    file_name = "MRR/data/K" + str(number_of_rings) + "_" + str(n)
+".csv"
    with open(file_name,"w",newline="") as file:
        writer = csv.writer(file)
        for i in range(n):
            K =
_floor_number(np.random.uniform(0.05,0.60,number_of_rings+1),2) #0.05～
0.6 までの小数 2 位までの乱数をリングの数だけ nparray として生成
            writer.writerow(K)
```

ソースコード 5：generate_LK.py

```
from tensorflow.python.keras.models import load_model

def
load_NN(n:int,number_of_rings:int):
                #ニューラルネットワークの読み込み
    file_name = "MRR/data/DNN2_K_" + str(number_of_rings) + "_" +
str(n) +".h5"      #ファイル名
    model_DNN = load_model(file_name)
    return model_DNN
```

ソースコード 6：load_NN.py

```
import numpy as np
import math
import tensorflow as tf
import random

def minus(x):                                          #map 関数とあわせて用いる 負をかけて小数第 5 位までを切り捨て
    return math.floor(-x * 10 ** 4) / (10 ** 4)

def minus_and_round(x):                                #負をかけて，小数第 5 位までを切り捨て
    return round(math.floor(-x * 10 ** 4) / (10 ** 4),0)

def minus_and_round_and_random(x):                                #負をかけて，小数第 5 位までを切り捨て
    return math.floor(x * 10 ** 4) / (10 ** 4) + random.uniform(-0.5,0.5)

def mean_percentage_squared_error(y_true,y_pred):   #平均二乗パーセント誤差 未使用
    percentage_error = (y_true - y_pred) / y_true
    loss = tf.math.reduce_mean(percentage_error**2)
    return loss


def graph_integrate(data1:np.array,data2:np.array,boader): #積分値の計算 boader の設定値によって一定以上の部分のみ積分計算をする
    if len(data1)==len(data2):
        area = 0
        for i in range(len(data1)):
                if data1[i]>data2[i]:
                    if data2[i]>-boader:
                        area += abs((1/data1[i])*(data1[i]-data2[i]))
                    elif data1[i]>-boader:
                        area += abs((1/data1[i])*(data1[i]+boader))
                    else:
```

```
                    pass
            if data2[i]>data1[i]:
                if data1[i]>-boader:
                    area += abs((1/data2[i])*(data2[i]-data1[i]))
                elif data2[i]>-boader:
                    area += abs((1/data2[i])*(data2[i]+boader))
                else:
                    pass
        return area
    else:
        print("length of datas is not match")              #互いの配列の
長さが一致していない場合は何もせず返す
        return
```

ソースコード 7：mymath.py

```
import numpy as np
import csv
from mymath import minus_and_round,graph_integrate
from transfer_function import TransferFunction
from load_NN import load_NN
from read_data import read_K
import copy

def
pre_K_NN(n:int,number_of_rings:int,L:np.array):
        #K を予測するためのニューラルネットワーク
    pre_number =
10                                                          #予測値
を 10 通り返す
    xaxis = np.arange(1540e-9,1560e-9,0.01e-
9)                                      #シミュレーション範囲 1540nm~1560nm を
0.01nm 間隔でプロットするための x 軸用の配列
    model_DNN =
load_NN(n,number_of_rings)                                          #モデ
ルのロード
    pre_data_K =
np.array(read_K(pre_number,number_of_rings))                        #予測
に用いる入力 K
    file_name = "MRR/data/pred_K" + str(number_of_rings) + "_" + str(n)
+".csv"     #結果の保存先のファイル名
    with open(file_name,"w",newline="") as
file:                                   #ファイル展開
        writer =
csv.writer(file)                                                    #書き込
み用
        for i in range(pre_number):
            pre_data =
TransferFunction(L,pre_data_K[i],config={'center_wavelength':1550e-
9,'eta':0.996,'n_eff':2.2,'n_g':4.4,'alpha':52.96})   #シミュレーションデータ
を格納
```

```
            temp =
np.array(list(map(minus_and_round,pre_data.simulate(xaxis))))    #NN に
入力するために加工
            pred_Y =
model_DNN.predict(temp.reshape(1,len(xaxis)))                    #予測値を
格納
            print(pred_Y[0],pre_data_K[i])
           #結果を表示
            writer.writerow(np.append(pred_Y[0],pre_data_K[i]))
            #結果のデータを書き込み

def
search_K(L,K,xaxis,target_trans_data,boader):
                #target には目標とする伝達関数の配列　L と K は目標に近づけたい補
正前の配列　補正法は勾配法を用いている
    ans_K =
0
    true_K =
copy.copy(K)
    temp =
10                                                                  #
ans_K 及び true_K は仮格納先　temp は探索繰り返し回数
    data = TransferFunction(L,K,config={'center_wavelength':1550e-
9,'eta':0.996,'n_eff':2.2,'n_g':4.4,'alpha':52.96})       #シミュレーション
    trans_data =
data.simulate(xaxis)                                              #伝達関
数仮格納先
    for i in
range(len(true_K)):                                                #fo
r 内は勾配法の計算
        temp_K = copy.copy(true_K)
        for j in range(temp*2+1):
            if
0.01<true_K[i]<0.99:                                             #0.0
1<K<0.99 内だったら格納　そうでなければ計算続行
                temp_K[i] = true_K[i] + 0.01*(temp-j)
```

```
            if temp_K[i]<0.01 or  temp_K[i]>0.99:
                continue
            data =
TransferFunction(L,temp_K,config={'center_wavelength':1550e-
9,'eta':0.996,'n_eff':2.2,'n_g':4.4,'alpha':52.96})
            trans_data = data.simulate(xaxis)
            temp_S =
graph_integrate(target_trans_data,trans_data,boader)       #target と現在
の L と temp_K による伝達関数で囲われた部分の面積を出す
            if j ==
0:                                                          #K[i]につい
ての探索終了時の面積を格納
                S=temp_S
            elif S >
temp_S:                                                     #より面積が小
さくなる K[i]ならば，その K[i]及び面積を格納
                S=temp_S
                ans_K = temp_K[i]
        if ans_K !=
0:                                                          #K[i]の値を
確定，なければそのまま
            true_K[i] = ans_K
    data = TransferFunction(L,true_K,config={'center_wavelength':1550e-
9,'eta':0.996,'n_eff':2.2,'n_g':4.4,'alpha':52.96})
    trans_data = data.simulate(xaxis)
    temp_S = graph_integrate(target_trans_data,trans_data,boader)
    return
true_K                                                                  #
補正後の K を返す

def fix_K(K,number):
    for i in range(len(K)):
        if K[i]>1 or K[i]<0:
            K[i]=number
    return K
```

ソースコード 8：optimize_data.py

```
import csv

def
read_K(n:int,number_of_rings:int):
       #K の読み込み
    file_name = "MRR/data/K" + str(number_of_rings) + "_" + str(n)
+".csv"          #ファイル名の読み込み
    with open(file_name) as
file:                                                    #ファイル展開
        reader = csv.reader(file,quoting =csv.QUOTE_NONNUMERIC)
        data = [row for row in reader]
    return
data                                                                #
行ごとのデータを配列で返す

def
read_L(n:int,number_of_rings:int):
       #L の読み込み
    file_name = "MRR/data/L" + str(number_of_rings) + "_" + str(n)
+".csv"          #ファイルの読み込み
    with open(file_name) as
file:                                                    #ファイル展開
        reader = csv.reader(file,quoting =csv.QUOTE_NONNUMERIC)
        data = [row for row in reader]
    return data
```

ソースコード 9：read_data.py

```
import csv
import numpy as np
import itertools

def _read_csv_values(file_name):
    with open(file_name) as file:
        reader = csv.reader(file, quoting=csv.QUOTE_NONNUMERIC)
        data = [row for row in reader]
    return data

def _generate_fixed_length_combinations(a, b, number_of_rings):
    combinations = itertools.product([a, b], repeat=number_of_rings)
    # 全て同じ要素で構成される配列を除外
    filtered_combinations = [comb for comb in combinations if not all(x
== comb[0] for x in comb)]
    return np.array(filtered_combinations)

def generate_allcomb_Lcsv(number_of_rings):
    # CSV ファイルから値を読み込む
    values = _read_csv_values("MRR/data/FSR20nm_ring_combinations.csv")

    # すべての組み合わせを格納するリスト
    all_combinations = []

    # 組み合わせを生成
    for i in range(len(values)):
        combinations_array =
_generate_fixed_length_combinations(values[i][0], values[i][1],
number_of_rings)
        for combination in combinations_array:
            all_combinations.append(combination)

    # すべての組み合わせを一つの CSV ファイルに書き込む
    file_name = "MRR/data/combination_test2/allcomb_FSR20nm_L" +
str(number_of_rings) + ".csv"
    with open(file_name, "w", newline="") as file:
```

```
        writer = csv.writer(file)
        for combination in all_combinations:
            writer.writerow(combination)
```

ソースコード 10：ring_combination.py

```python
from re import A
import numpy as np
from typing import Any, Dict, List

class TransferFunction:
    """Simulator of the transfer function of the MRR filter.
    Args:
        L (List[float]): List of the round-trip length.
        K (List[float]): List of the coupling rate.
        config (Dict[str, Any]): Configuration of the MRR.
            Keys:
                eta (float): The coupling loss coefficient.
                n_eff (float): The effective refractive index.
                n_g (float): The group index.
                alpha (float): The propagation loss coefficient.
    Attributes:
        L (List[float]): List of the round-trip length.
        K (List[float]): List of the coupling rate.
        eta (float): The coupling loss coefficient.
        n_eff (float): The effective refractive index.
        n_g (float): The group index.
        a (List[float]): List of the propagation loss.
    """

    def __init__(
        self,
        L,
        K,
        config: Dict[str, Any]
    ) -> None:
        self.L: List[float] = L
        self.K: List[float] = K
        self.center_wavelength: float = config['center_wavelength']
        self.eta: float = config['eta']
        self.n_eff: float = config['n_eff']
        self.n_g: float = config['n_g']
```

```
        self.a: List[float] = np.exp(- config['alpha'] * L)

    def _C(self, K_k: float) -> np.array:
        return 1 / (-1j * self.eta * np.sqrt(K_k)) * np.array([
            [1, - self.eta * np.sqrt(self.eta - K_k)],
            [np.sqrt(self.eta - K_k) * self.eta, - self.eta ** 2]
        ])

    def _R(self, a_k: float, L_k: float, l: np.array) -> np.array:
        x = 1j * np.pi * L_k * self.n_g * (l - self.center_wavelength) /
self.center_wavelength / self.center_wavelength
        return np.array([
            [np.exp(x) / np.sqrt(a_k), 0],
            [0, np.exp(-x) * np.sqrt(a_k)]
        ], dtype="object")

    def _reverse(self, arr: List) -> List:
        return arr[::-1]

    def _M(self, l: List[float]) -> np.array:
        product = np.identity(2)
        for _K, _a, _L in zip(
            self._reverse(self.K[1:]),
            self._reverse(self.a),
            self._reverse(self.L)
        ):
            product = np.dot(product, self._C(_K))
            product = np.dot(product, self._R(_a, _L, l))
        product = np.dot(product, self._C(self.K[0]))
        return product

    def _D(self, l: List[float]) -> np.mat:
        return 1 / self._M(l)[0, 0]

    def print_parameters(self) -> None:
        print('eta:', self.eta)
```

```
        print('center_wavelength:', self.center_wavelength)
        print('n_eff:', self.n_eff)
        print('n_g:', self.n_g)
        print('a:', self.a)
        print('K:', self.K.tolist())
        print('L:', self.L.tolist())

    def simulate(self, l: List[float]) -> np.array:
        y = 20 * np.log10(np.abs(self._D(l)))
        return y.reshape(y.size)

class build_TransferFunction_Factory:
    def __init__(self, config):
        self.config = config

    def create(self, L, K):
        return TransferFunction(L, K, self.config)

def build_TransferFunction(config):
    """Partial-apply config to TransferFunction
    Args:
        config (Dict[str, Any]): Configuration of the TransferFunction
    Returns:
        TransferFunction_with_config: TransferFunction that is partial-
applied config to.
    """
    return build_TransferFunction_Factory(config).create
```

ソースコード 11：transfer_function.py