第 14 回報告書　　2022 年 1 月 18 日

分布定数線路の周波数特性

愛媛大学工学部
8531037m
祖父江匠真

1　はじめに

前回, 分布定数線路の周波数特性を調べたが, ゲインの値が正しくなかったのでプログラムを修正した.

2　分布定数線路の周波数特性

伝搬定数 γ は

$$\gamma = \sqrt{(R + j\omega L)(G + j\omega C)} \tag{1}$$

より求めた.
図 1 の回路図について, 周波数特性を調べる.

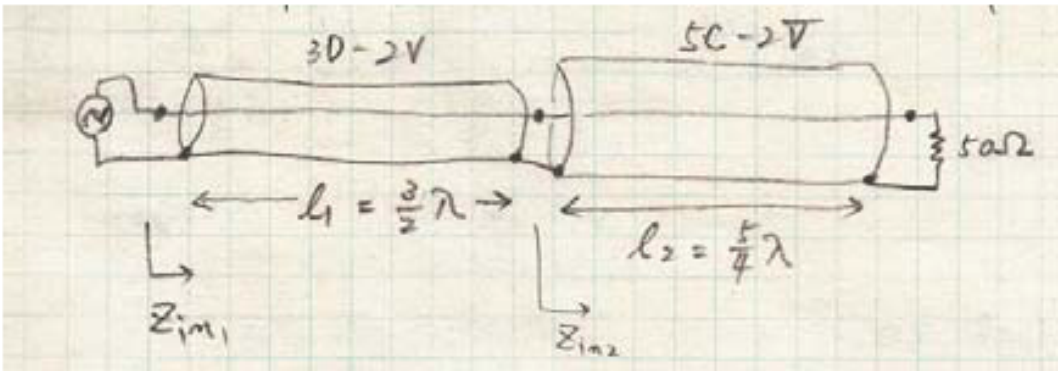

図 1: 回路図

周波数特性のグラフを出力するプログラムをソースコード 1 に示す.

ソースコード 1: 周波数特性

```
1 import cmath
2 import math
3 import math
```

```python

import matplotlib

matplotlib.rc("font", family="Noto Sans CJK JP")
import matplotlib.pyplot as plt

import numpy as np
from tqdm import tqdm

import cable
from snippet import drawFrequencyResponse
import util


def calculateTheta(frequency_Hz, cable):
    """
    伝搬定数 γ と同軸ケーブルの長さ *l* の積を求める

    *Parameters*
    ----------
    *frequency_Hz : float*
        周波数 *(Hz)*
    *cable : instance*
        *Cable* クラスのインスタンス
    """
    omega = 2 * np.pi * frequency_Hz

    R = cable.resistance # Ω/m
    L = cable.inductance # H/m
    G = cable.conductance # S/m
    C = cable.capacitance # F/m

    gamma = cmath.sqrt((R + 1j * omega * L) * (G + 1j * omega * C))
    theta = gamma * cable.length

    return theta


def createFMatrixForDcc(frequency_Hz, theta, cable):
    """
    分布定数回路の *F* 行列を求める

    *Parameters*
    ----------
    *frequency_Hz : float*
        周波数 *(Hz)*
    *theta : float*
        伝搬定数 γ と同軸ケーブルの長さ *l* の積
    *cable : instance*
        *Cable* クラスのインスタンス
    """

    cosh = cmath.cosh(theta)
    sinh = cmath.sinh(theta)
    return np.array(
        [
            [cosh, cable.calcCharacteristicImpedance(frequency_Hz) * sinh],
```

```
                [sinh / cable.calcCharacteristicImpedance(frequency_Hz), cosh],
            ]
        )


def createTransferFunction(frequency_Hz, endCondition, cable):
    """
    受電端に抵抗を接続した分布定数回路の伝達関数を求める

    Parameters
    ----------
    frequency_Hz : float
        周波数 (Hz)
    endCondition: dict
        受電端の抵抗の条件
    cable : instance
        Cable クラスのインスタンス
    """
    # 分布定数線路のF 行列を求める
    f_matrix_dcc = createFMatrixForDcc(
        frequency_Hz, calculateTheta(frequency_Hz, cable), cable
    )

    if endCondition["shouldMatching"]:
        # 線路の特性インピーダンスと、受電端の抵抗のインピーダンスを同じにする
        endImpedance = cable.calcCharacteristicImpedance(frequency_Hz)
    else:
        endImpedance = endCondition["impedance"]

    # F 行列と末端のインピーダンスから伝達関数を計算する
    return util.createTransferFunctionFromFMatrix(endImpedance, f_matrix_dcc)


def drawBodePlot(frequencies_Hz, endCondition, cable, fileName=""):
    """
    分布定数線路のボード線図をグラフに表示する

    Parameters
    ----------
    frequencies_Hz : list
        周波数 (Hz)のリスト
    endCondition: dict
        受電端の抵抗の条件
    cable : instance
        Cable クラスのインスタンス
    fileName: string
        表示するグラフを保存する際のファイル名
    """

    tfs = []
    tfs_nthPwrOf10 = []
    # 周波数ごとに伝達関数を求める
    for frequency_Hz in tqdm(frequencies_Hz, leave=False):
        tf = createTransferFunction(frequency_Hz, endCondition, cable)
        tfs.append(tf)

```

```
117             # 周波数が 10のn 乗ごとに、伝達関数の計算結果を控えておく
118             if frequency_Hz > 1 and math.log10(frequency_Hz).is_integer():
119                 tfs_nthPwrOf10.append({"frequency_Hz": frequency_Hz, "tf": tf})
120
121     # ゲインの傾きを求める
122     slope = util.calcMinimumSlope(tfs_nthPwrOf10)
123     print(f"傾き: {slope}[dB/dec]")
124
125     fig, axes = plt.subplots(1, 2)
126     axes[0].plot(
127         frequencies_Hz,
128         list(map(util.convertGain2dB, tfs)),
129     )
130     axes[0].set_xlabel("frequency [Hz]")
131     axes[0].set_ylabel("Gain [dB]")
132     axes[0].set_xscale("log")
133     if max(np.abs(tfs)) - min(np.abs(tfs)) < 1e-6:
134         axes[0].set_ylim(-5, 5)
135
136     axes[1].plot(
137         frequencies_Hz,
138         list(map(lambda tf: math.atan2(tf.imag, tf.real) * 180 / np.pi, tfs)),
139     )
140     axes[1].set_xlabel("frequency [Hz]")
141     axes[1].set_ylabel("phase [deg]")
142     axes[1].set_xscale("log")
143
144     if fileName != "":
145         fig.savefig(f"{fileName}")
146     plt.show()
147
148
149 def drawFrequencyResponse(frequencies_Hz, cable, fileName=""):
150     """
151     分布定数線路の周波数特性をグラフに表示する
152
153     Parameters
154     ----------
155     frequencies_Hz : list
156         周波数 (Hz)のリスト
157     endCondition: dict
158         受電端の抵抗の条件
159     cable : instance
160         Cable クラスのインスタンス
161     fileName: string
162         表示するグラフを保存する際のファイル名
163     """
164
165     # open
166     # 共振周波数の分母
167     resonance_denominator_open = (
168         2 * cable.length * math.sqrt(cable.inductance * cable.capacitance)
169     )
170     # 反共振周波数の分母
171     antiresonance_denominator_open = (
172         4 * cable.length * math.sqrt(cable.inductance * cable.capacitance)
173     )
```

```
174
175     # 開放
176     resonance_freqs_open = [] # 共振周波数
177     antiresonance_freqs_open = [] # 反共振周波数
178     # 整数n
179     n_length = range(10)
180     for n in n_length:
181         # 共振周波数
182         resonance_freq_open = n / resonance_denominator_open
183         resonance_freqs_open.append(resonance_freq_open)
184         # 反共振周波数
185         antiresonance_freq_open = (2 * n + 1) / antiresonance_denominator_open
186         antiresonance_freqs_open.append(antiresonance_freq_open)
187     # 引数のfrequencies_Hzに一番近い
188     # 短絡
189     resonance_freqs_short = antiresonance_freqs_open # 共振周波数
190     antiresonance_freqs_short = resonance_freqs_open # 反共振周波数
191
192     conditions = [
193         {"shouldMatching": True, "impedance": 0},
194         {"shouldMatching": False, "impedance": 1e6},
195         {"shouldMatching": False, "impedance": 1e-6},
196     ]
197     # fig, axes = plt.subplots(1, 3)
198     for (i, condition) in enumerate(conditions):
199         fig, ax = plt.subplots()
200         axes = [ax, ax, ax]
201
202         tfs = []
203         for frequency_Hz in tqdm(frequencies_Hz, leave=False):
204             tf = createTransferFunction(frequency_Hz, condition, cable)
205             tfs.append(tf)
206
207         # if i == 2:
208         # print(list(map(abs, tfs)))
209
210         axes[i].plot(
211             frequencies_Hz,
212             list(map(abs, tfs)),
213             # list(map(util.convertGain2dB, tfs)),
214         )
215         if cable.resistance == 0 and cable.conductance == 0:
216             # 無損失ケーブル
217             if i == 1:
218                 # open
219                 axes[i].plot(
220                     resonance_freqs_open,
221                     list(
222                         map(
223                             abs,
224                             calcTfsBySomeFreqs(resonance_freqs_open, condition,
                                cable),
225                         )
226                     ),
227                     marker="v",
228                     color="blue",
229                     linestyle="",
```

```
                )
                axes[i].plot(
                    antiresonance_freqs_open,
                    list(
                        map(
                            abs,
                            calcTfsBySomeFreqs(
                                antiresonance_freqs_open, condition, cable
                            ),
                        )
                    ),
                    marker="o",
                    color="red",
                    linestyle="",
                )
                axes[i].legend(["全ての周波数", "共振周波数", "反共振周波
                    数"], loc="best")
            elif i == 2:
                # short
                axes[i].plot(
                    resonance_freqs_short,
                    list(
                        map(
                            abs,
                            calcTfsBySomeFreqs(resonance_freqs_short, condition,
                                cable),
                        )
                    ),
                    marker="v",
                    color="blue",
                    linestyle="",
                )
                axes[i].plot(
                    antiresonance_freqs_short[1:],
                    list(
                        map(
                            abs,
                            calcTfsBySomeFreqs(
                                antiresonance_freqs_short[1:], condition, cable
                            ),
                        )
                    ),
                    marker="o",
                    color="red",
                    linestyle="",
                )
                axes[i].legend(["全ての周波数", "共振周波数", "反共振周波
                    数"], loc="best")
        text = (
            "matching"
            if condition["shouldMatching"]
            else "open"
            if condition["impedance"] >= 1e6
            else "short"
        )
        axes[i].set_title(f"{text}")
        axes[i].set_ylabel("|H(f)|") # y 軸は、伝達関数の絶対値
```

```
284             axes[i].set_xlabel("frequency␣[Hz]")
285             axes[i].set_yscale("log") # y 軸は log スケールで表示する
286             axes[i].ticklabel_format(style="sci", axis="x", scilimits=(0, 0))
287             if cable.resistance == 0 and cable.conductance == 0:
288                 if max(np.abs(tfs)) − min(np.abs(tfs)) < 1e−6:
289                     axes[i].set_ylim(1e−1, 1e3)
290 
291     if fileName != "":
292         fig.savefig(util.createImagePath(fileName))
293 
294     plt.subplots_adjust(
295         left=0.0625, bottom=0.1, right=0.98, top=0.9, wspace=0.2, hspace=0.35
296     )
297     plt.show()
298 
299 
300 def calcTfsBySomeFreqs(frequencies_Hz, endCondition, cable):
301     tfs = []
302     for frequency_Hz in tqdm(frequencies_Hz, leave=False):
303         # 5C−2V + Zr の回路の入力インピーダンスを受電端側の抵抗 Zr とする
304         tf = createTransferFunction(frequency_Hz, endCondition, cable)
305         tfs.append(tf)
306     return tfs
307 
308 
309 # ケーブルのインスタンスを作成
310 cable_vertual = cable.Cable(
311     resistance=1e−6,
312     inductance=1.31e−7,
313     conductance=1e−4,
314     capacitance=67e−12,
315     length=1000,
316 )
317 
318 # 無損失ケーブル
319 cable_noLoss_vertual = cable.Cable(
320     resistance=0,
321     # inductance=1.31e−7,
322     # 特性インピーダンスの計算結果が 50[Ω]になるように意図的に値を設定
323     inductance=100e−12 ∗ 50 ∗∗ 2, # C ∗ Zo ∗∗ 2
324     conductance=0,
325     # capacitance=67e−12,
326     capacitance=100e−12,
327     length=1000,
328 )
329 
330 # ケーブルの周波数特性をグラフにする
331 frequencies_Hz = list(range(0, 10000, 10))
332 frequencies_Hz.extend(list(range(10000, 200 ∗ 10 ∗∗ 6, 10000)))
333 # drawBodePlot(
334 # np.logspace(4, 6, 1000, base=10),
335 # {"shouldMatching": False, "impedance": 1e6},
336 # cable_vertual,
337 # )
338 # drawBodePlot(
339 # np.logspace(4, 6, 1000, base=10),
340 # {"shouldMatching": True, "impedance": 1e6},
```

```
# cable_noLoss_vertual,
# )

drawFrequencyResponse(
    # list(range(10000, 1000000, 100)),
    list(range(0, 1000000, 100)),
    cable_noLoss_vertual,
    # cable.Cable(
    # resistance=1e-3,
    # inductance=100e-12 * 50 ** 2, # C * Zo ** 2
    # conductance=1e-4,
    # capacitance=100e-12,
    # length=1000,
    # ),
)


def squareWaveFftAndIfft(cable, endCondition):
    # サンプリング周期の逆数が入力波形の周波数?
    f = 1000
    rate = 44100 # サンプリング周波数(1秒間に何回サンプリングするか、ナイキスト周波数は 44100 / 2)
    # 方形波
    T = np.arange(
        0, 0.0087, 1 / rate # 0.0087は単パルスが真ん中に来るよう調整した
    ) # len(T) => 441, 1 / rate はサンプリング周期(何秒おきにサンプリングするか)
    # 足し合わされる波は、入力波の周波数の整数倍の周波数を持つ
    squareWaves_time = np.sign(np.sin(2 * np.pi * f * T))
    prevIndex = 0
    indexChunk = []
    chunks = []
    for index, discreteValue in enumerate(squareWaves_time):
        if discreteValue == 1:
            if prevIndex + 1 == index:
                # 同じ-1の塊に現在いる
                indexChunk.append(index)
            else:
                # 次の-1の塊に移動した
                chunks.append(indexChunk)
                indexChunk = []
            prevIndex = index

    single_palse = []
    # print(chunks[4])
    # print("単パルス波形の周波数? => ", 1 / (len(chunks[4]) * (1 / rate)))
    for index, y in enumerate(squareWaves_time):
        if index in chunks[4]:
            single_palse.append(y)
        else:
            single_palse.append(0)
    inputWaves_time = list(single_palse)

    # sinc 関数
    # T = np.linspace(-10, 10, 1000)
    # sincWaves_time = np.sinc(T)
    # inputWaves_time = sincWaves_time
```

```
396
397     fig, axes = plt.subplots(3, 2)
398     axes = axes.flatten()
399     # fig, axes = plt.subplots()
400     # fig, axes1 = plt.subplots()
401     # fig, axes2 = plt.subplots()
402     # fig, axes3 = plt.subplots()
403     # fig, axes4 = plt.subplots()
404     # fig, axes5 = plt.subplots()
405     # axes = [axes, axes1, axes2, axes3, axes4, axes5]
406
407     axes[0].plot(T, inputWaves_time)
408     axes[0].set_title("input(t)")
409     axes[0].set_xlabel("Time")
410     axes[0].set_ylabel("Gain")
411     axes[0].set_xlabel("time␣[s]")
412
413     # フーリエ変換
414     # 各離散値は、それぞれlen(離散信号列)個の複素正弦波の一次結合で表される(
            DFT)
415     # 実数をFFTする場合、
416     # 負の周波数のフーリエ係数の値は、
417     # 対応する正の周波数のフーリエ係数の虚数部分を打ち消すために共役な値を
            とる為、
418     # 情報としては正の周波数部分のみで十分
419     # numpy.fft.fft(
420     # FFTを行う配列,
421     # FFTを行うデータ点数。Noneとするとaの長さに等しくなる,
422     # FFTを行う配列の軸方向。指定しなければ、配列の最大次元の方向となる,
423     # "ortho"とすると正規化する。正規化すると変換値が1/√
            Nになる（Nはデータ点数
424     # )
425     # numpy.fft.fft()の戻り値は、長さnの複素数配列
426     # inputWaves_fft = np.fft.fft(inputWaves_time)
427     # 工学系の用途向けに、実数のFFTに特化したnp.fft.rfftが用意されている。
428     inputWaves_fft = np.fft.rfft(inputWaves_time)
429
430     # 離散フーリエ変換のサンプル周波数を返す(rfft, irfftで使用するため)
431     # np.fft.fftfreq(FFTを行うデータ点数, サンプリング周期) # サンプリング周期
            次第で時系列データの時間軸の長さが決定する(100点, 0.01)なら100 ∗
            0.01[s]の時系列データということになる?
432     # frequencies = np.fft.fftfreq(len(inputWaves_time), 1.0 / rate)
433     frequencies = np.fft.rfftfreq(
434         len(inputWaves_time), 1.0 / rate
435     ) # len(frequencies) => 193, 1/rateはサンプリング周期(何秒おきにサンプリン
            グするか)
436     # print(frequencies, len(frequencies))
437
438     axes[1].plot(frequencies, np.abs(inputWaves_fft)) # absで振幅を取得
439     axes[1].set_title("abs(F[input(t)])")
440     axes[1].set_xlabel("Frequency")
441     axes[1].set_ylabel("|F[input(t)]|")
442     axes[1].set_xlabel("Frequency␣[Hz]")
443
444     tfs = calcTfsBySomeFreqs(
445         frequencies,
```

```
446             endCondition,
447             cable,
448         )
449
450         axes[2].plot(frequencies, list(map(lambda tf: abs(tf), tfs)))
451         axes[2].set_title("abs(H(f))")
452         axes[2].set_xlabel("Frequency")
453         axes[2].set_ylabel("|H(f)|")
454         axes[2].set_xlabel("Frequency␣[Hz]")
455         # axes[2].set_yscale("log")
456
457         convolution = np.array(inputWaves_fft) * np.array(
458             tfs
459         ) # 時間軸の畳み込み積分 = フーリエ変換した値同士の積 (の値も周波数軸のもの)
460         # convolution = np.array(inputWaves_fft, dtype=np.complex) * np.array(
461         # list(map(lambda tf: abs(tf), tfs)), dtype=np.complex
462         # )
463
464         # 入力波形のフーリエ変換 * 伝達関数
465         axes[3].plot(frequencies, np.abs(convolution)) # abs で振幅を取得
466         axes[3].set_title("abs(F[input(t)]␣*␣H(f))")
467         axes[3].set_xlabel("Frequency")
468         axes[3].set_ylabel("|F[input(t)]␣*␣H(f)|")
469         axes[3].set_xlabel("Frequency␣[Hz]")
470
471         # 逆フーリエ変換
472         # r = np.fft.ifft(convolution, len(T))
473         # 入力波形が実数データ向けの逆FFT np.fft.irfft が用意されている。
474         # 33点のrfft 結果を入力すれば 64 点の時間領域信号が得られる。
475         # 入力波形が実数値のみなので、出力波形も虚数部分は捨ててよい?
476         r = np.fft.irfft(convolution, len(T))
477         axes[4].plot(T, np.real(r))
478         axes[4].set_title("output(t).real")
479         axes[4].set_xlabel("Time")
480         axes[4].set_ylabel("Gain")
481         axes[4].set_xlabel("time␣[s]")
482
483         r = np.fft.irfft(convolution, len(T))
484         axes[5].plot(T, np.imag(r))
485         axes[5].set_title("output(t).imag")
486         axes[5].set_xlabel("Time")
487         axes[5].set_ylabel("Gain")
488
489         plt.tight_layout()
490         plt.show()
491
492
493 # squareWaveFftAndIfft(
494 # cable.Cable(
495 # resistance=1e-3, # 無損失ケーブルを考える
496 # # ケーブルの特性インピーダンスの計算結果が 50[Ω]になるように意図的に値を設定
497 # inductance=100e-12 * 50 ** 2,
498 # conductance=1e-4, # 無損失ケーブルを考える
499 # capacitance=100e-12, # シートの値を参考に設定?
500 # length=1000,
```

```
# ),
# {"shouldMatching": False, "impedance": 1e-6}, # 受電端の抵抗が 0のとき、断線
    していない正常のケーブル?
# )
```

ソースコード 1 によって得られた周波数特性を図 2 に示す. 図 2 では γ の計算に, G = 0, R は図 3, 図 4 の導体抵抗 [20 ℃] の値を使用している.

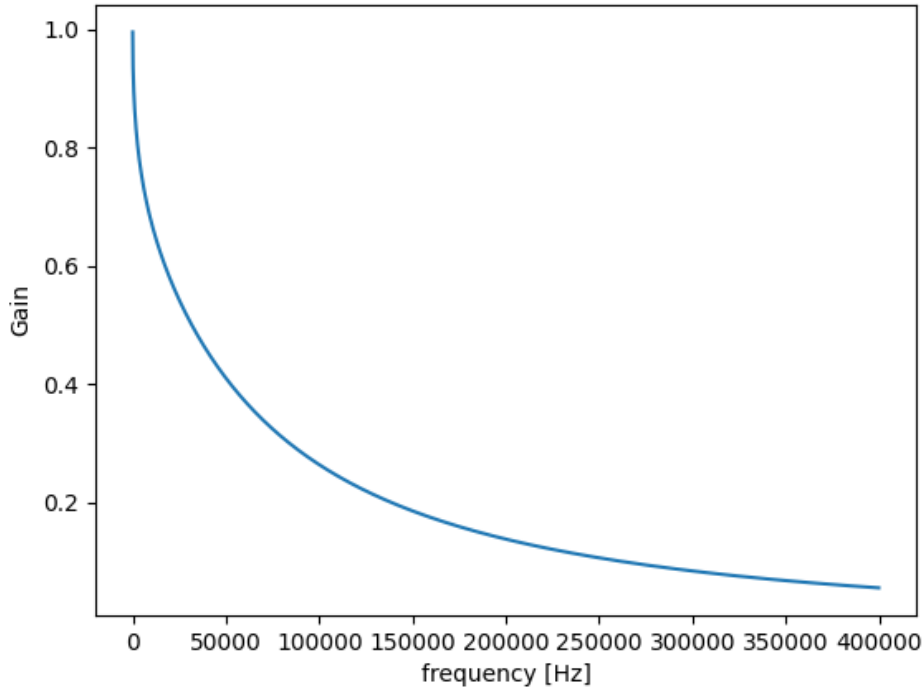

図 2: 周波数特性

項目 / 記号	導体抵抗 [20℃] MΩ-km (以下)	耐電圧 AC.V/ 1分間	静電容量 nF/km (約)	絶縁抵抗 MΩ-km (以下)	特性インピーダンス Ω	標準減衰量 dB/km 400MHz	標準減衰量 dB/km 900MHz
3D-2V	33.30	1000	100	1000	50±2	309	471

図 3: 3D-2V ケーブルの仕様

		5C-2V
両端処理		
内部導体	材質	
	素線本数/外径	1本/0.8mm
絶縁体	材質	
	外径	
外部導体編組	材質	
	外径	
シース	材質	
	外径	7.2mm
	標準色	
	他色	白
支持線標準サイズ		—
概算質量		65kg/km
導体抵抗[20℃]		35.9MΩ-km
耐電圧		
静電容量		
絶縁抵抗		
特性インピーダンス		
標準減衰量	10MHz	—
	220MHz	131dB/km
	770MHz	263dB/km

図 4: 5C-2V ケーブルの仕様

また, R = 0, G = 0 で γ を計算した際の周波数特性を図 5 に示す.

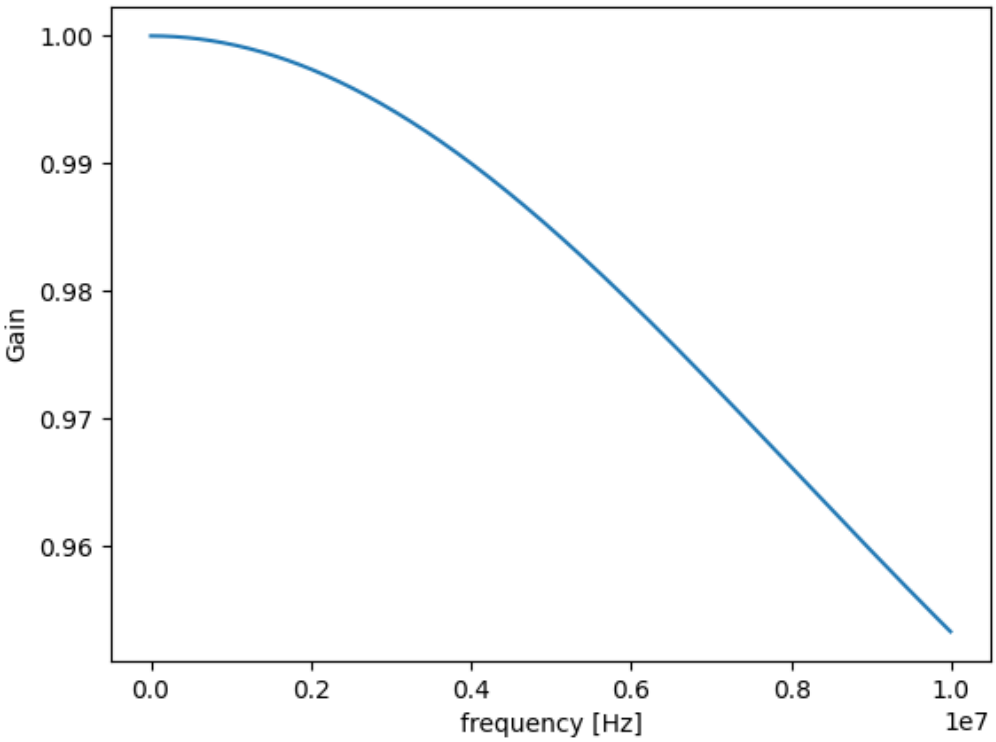

図 5: 無損失線路の周波数特性

3　おわりに

今回は, 分布定数線路の周波数特性のグラフの縦軸を修正した.

参考文献

[1] 都築, ”2020Q4-応用通信工学 II-都築 ”, moodle 内, 参照 December 8,2021.

[2] システムギアダイレクト, ”3D-2V　無線用同軸ケーブル”, https://www.systemgear.jp/kantsu/3d2v.php, 参照 December 8,2021.

[3] システムギアダイレクト, ”5C-2V　同軸ケーブル”, https://www.systemgear.jp/kantsu/5c2v.php, 参照 December 8,2021.