

第4回輪講資料

4321 野秋 琳太郎

2025 年 11 月 15 日

指導教員 宮田 尚起

1. はじめに

四年生ゼミでは高周波回路の勉強が始まった。これまでの輪講では先輩方の研究内容がつかみにくかったため、しばらくはそれらを自分で再現しつつ発表していく方針とした。今回は、前回のゼミ内容とも流れが近く、シミュレーション環境を整える手間も比較的少なく済みそうだったことから、三浦先輩の研究を題材にして調べることにした。

2. やりたいこと

電信方程式は、分布定数回路のキャパシタが線形な特性を持っていると仮定している。しかし、実際のキャパシタは非線形な特性を持っており、その非線形性が超高周波 (数百 GHz とか) においては無視できず、それによって信号線にソリトン波が発生することが知られている。そのため、非線形なキャパシタを考慮した電信方程式を導出する必要がある。非線形なキャパシタのモデルは先行研究でいくつか提案されているので、それで拡張した電信方程式を導出し、シミュレーションを行うことができれば研究としては完成ということになる。また、先輩の研究では”数値計算”も重きを置いているので、その方法も調べる必要がある。

3. KdV 方程式

KdV 方程式は、非線形波動方程式の一種であり、浅水波やプラズマ波動などの現象を記述するために用いられる。その一般的な形は式 1 の通りである。この解は解析的に求めることができ、図 1 に示すようなパルス状の波 (soliton) と呼ばれる特異な波動を記述することができる。ここで、式 1 の $u(x, t)$ を 2 とし、3 の初期条件を与えている。この波は、非線形性と分散性のバランスによって形成され、他の波と衝突しても形状を保つ特性を持つ。例えば、水面に 2 つの石を別の場所に投げ込んだときに発生する波は、衝突しても元の形状を保ちながら進む。これはこのソリトン波の特性によるものである。KdV 方程式が非線形はどうの中でも広い応用範囲を持つことと、ソリトン波が確認できていることから、非線形なキャパシタを考慮した電信方程式を解ける形で導出する際に

利用できると思われる。

$$\frac{\partial u(x, t)}{\partial t} + 6u \frac{\partial u(x, t)}{\partial x} + \frac{\partial^3 u(x, t)}{\partial x^3} = 0 \quad (1)$$

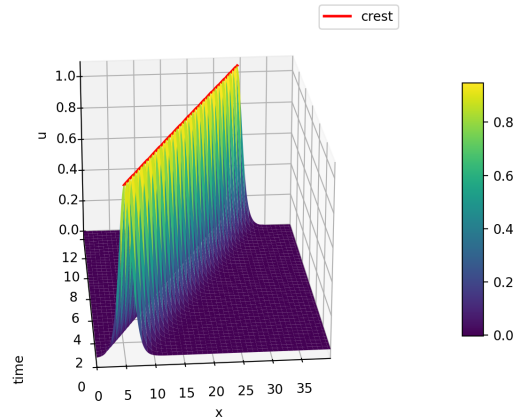


図 1. KdV 方程式の孤立波解の例

$$u(x, t) = 2k^2 \operatorname{sech}^2(k(x - 4k^2t - x_0)) \quad (2)$$

$$u(x, 0) = \frac{c}{2} \operatorname{sech}^2\left(\frac{\sqrt{c}}{2}(x - x_0)\right) \quad (3)$$

4. キャパシタモデルの改善

高周波回路、特に非線形伝送線路 (NLTL: Non-linear Transmission Line) の解析において、キャパシタのモデルとしてバラクタダイオード (Varactor Diode) が重要な役割を果たす。これは半導体の pn 接合を利用した素子であり、逆バイアス電圧を変化させることで静電容量を制御できる特性を持つ。一般にバラクタダイオードは集中定数素子であるが、これを伝送線路に周期的に装荷し、その間隔が信号波長に対して十分に短い場合、巨視的には電圧依存性を持つ分布定数線路として扱うことが可能である。

このキャパシタの静電容量 $C(V)$ は、印加電圧 V に対して非線形な特性を持つ。この特性を表現する

モデルとして、以下の式 (4) が広く用いられている。

$$C(V) = \frac{C_{J0}}{\left(1 + \frac{V}{\phi}\right)^\gamma} \quad (4)$$

ここで、 C_{J0} はバイアス電圧がゼロの時の接合容量、 ϕ は拡散電位（ビルトインポテンシャル）、 γ は接合容量係数である（例えば、シリコン階段接合では $\gamma = 0.5$ とされる）。本節では、この非線形キャパシタンスモデルを考慮した、より厳密な電信方程式を導出する。

5. 電信方程式の拡張

通常の線形電信方程式は以下の式 (5) で与えられる。

$$\frac{\partial^2 V}{\partial x^2} = LC \frac{\partial^2 V}{\partial t^2} + (RC + GL) \frac{\partial V}{\partial t} + GRV \quad (5)$$

ここで、 L 、 C 、 R 、 G はそれぞれ単位長さあたりのインダクタンス、キャパシタンス、抵抗、コンダクタンスであり、すべて定数である。しかし、キャパシタンスが電圧の関数 $C(V)$ である場合、線形の方程式 (式 5) の定数 C を、単に $C(V)$ に書き換えるだけではよくない。電圧 V が時間変化すれば、それに応じて容量 $C(V)$ 自体も時間変化するため、時間微分の計算において積の微分と連鎖律の影響を受けるからである。

伝送線路における電流 I の空間的な減少は、蓄積される電荷 Q の時間変化と、漏れ電流（コンダクタンス G ）によって記述される。

$$-\frac{\partial I}{\partial x} = \frac{\partial Q}{\partial t} + GV \quad (6)$$

ここで、変位電流の項に着目する。電荷の変化は「微分容量 $C(V) = dQ/dV$ 」を用いて次のように記述される。

$$\frac{\partial Q}{\partial t} = \frac{dQ}{dV} \frac{\partial V}{\partial t} = C(V) \frac{\partial V}{\partial t} \quad (7)$$

次に、電圧の波動方程式を導出するために、もう一つの基礎方程式 $-\frac{\partial V}{\partial x} = L \frac{\partial I}{\partial t} + RI$ を用いて、 I を消去する操作を行う。このとき、右辺にある $L \frac{\partial I}{\partial t}$ の項に上記の電流の式を代入すると、「関数の積の時間微分」が現れる。すなわち、 $C(V)$ と $\frac{\partial V}{\partial t}$ という「時間によって変化する 2 つの関数」の積を微分することになる。

$$\frac{\partial}{\partial t} \left(\underbrace{C(V)}_{f(t)} \cdot \underbrace{\frac{\partial V}{\partial t}}_{g(t)} \right) \quad (8)$$

これに積の微分公式 $(fg)' = f'g + fg'$ を適用すると、以下の 2 つの項に分かれる。

$$(\text{左辺}) = \underbrace{C(V) \frac{\partial^2 V}{\partial t^2}}_{\text{通常の波の項}(f \cdot g')} + \underbrace{\frac{\partial C(V)}{\partial t} \frac{\partial V}{\partial t}}_{\text{非線形による追加項}(f' \cdot g)} \quad (9)$$

ここで、第 2 項の $\frac{\partial C(V)}{\partial t}$ に注目する。容量 C は電圧 V の関数であり、 V は時間 t の関数であるため、ここでも連鎖律が適用される。

$$\frac{\partial C(V)}{\partial t} = \frac{dC(V)}{dV} \frac{\partial V}{\partial t} \quad (10)$$

これを元の式に戻すと、最終的に $\frac{\partial V}{\partial t}$ が 2 回掛け合わされた形（2 乗の項）が出現する。

$$\text{追加項} = \left(\frac{dC(V)}{dV} \frac{\partial V}{\partial t} \right) \cdot \frac{\partial V}{\partial t} = \frac{dC(V)}{dV} \left(\frac{\partial V}{\partial t} \right)^2 \quad (11)$$

以上を整理することで、非線形キャパシタンス $C(V)$ を考慮した拡張された電信方程式は次式となる。

$$\frac{\partial^2 V}{\partial x^2} = L \left[\underbrace{C(V) \frac{\partial^2 V}{\partial t^2}}_{\text{線形項の拡張}} + \underbrace{\frac{dC(V)}{dV} \left(\frac{\partial V}{\partial t} \right)^2}_{\text{非線形特有の項}} \right] + (RC(V) + GL) \frac{\partial V}{\partial t} + GRV \quad (12)$$

式 (12) の第 1 行目右辺にある第 2 項が、容量の電圧依存性によって生じる新たな項である。これは「電圧が急激に変化する ($\partial V / \partial t$ が大きい) ほど、波の挙動に強く影響を与える」ことを意味しており、衝撃波の発生やソリトン波の形成といった非線形現象の源となる要素である。

6. KdV 方程式にあてはめたい

7. 考えたこと

8. わからないこと

9. おわりに

10. 付録

10.1 KdV 方程式を解く Python コード

3 節で紹介した KdV 方程式を数値的に解く Python コードを以下に示す.

```
1 import os
2 import matplotlib.pyplot as plt
3 import numpy as np
4 from mpl_toolkits.mplot3d import Axes3D # noqa: F401 # needed for 3D
5
6 # るんげくったばらめた
7 def kdv_etdrk4(
8     Lx: float = 40.0,
9     N: int = 512,
10    dt: float = 0.005,
11    tmax: float = 12.0,
12    save_steps: int = 300,
13    c: float = 2.0,
14    x0: float = 5.0,
15 ):
16     # KdV方程式を解く!!
17     # Grid: 0 から始まる周期区間  $x$  in  $[0, Lx)$ 
18     x = np.linspace(0.0, Lx, N, endpoint=False)
19
20     # Wavenumbers for domain length Lx
21     k = 2 * np.pi * np.fft.fftfreq(N, d=Lx / N)
22     ik = 1j * k
23     L = 1j * k**3 # Linear operator in Fourier space:  $+ i k^3$ 
24
25     # Initial condition: 1-soliton for KdV (coefficient 6)
26     #  $u(x,0) = c/2 * \text{sech}^2(\text{sqrt}(c)/2 (x - x0))$ 
27     def sech(z):
28         return 1.0 / np.cosh(z)
29
30     u = 0.5 * c * sech(0.5 * np.sqrt(c) * (x - x0)) ** 2
31
32     # ETDRK4 precomputations (Kassam & Trefethen 2005)
33     E = np.exp(dt * L)
34     E2 = np.exp(dt * L / 2)
35     M = 16 # contour points for phi functions
36     r = np.exp(1j * np.pi * (np.arange(1, M + 1) - 0.5) / M)
37     LR = dt * L[:, None] + r[None, :]
38
39     Q = dt * np.mean((np.exp(LR / 2) - 1.0) / LR, axis=1)
40     f1 = dt * np.mean(
41         (-4 - LR + np.exp(LR) * (4 - 3 * LR + LR**2)) / LR**3,
42         axis=1,
43     )
44     f2 = dt * np.mean(
45         (2 + LR + np.exp(LR) * (-2 + LR)) / LR**3,
46         axis=1,
47     )
48     f3 = dt * np.mean(
```

```

49         (-4 - 3 * LR - LR**2 + np.exp(LR) * (4 - LR)) / LR**3,
50         axis=1,
51     )
52
53     # Nonlinear term N(u) = -6 u u_x (compute u_x via spectral derivative)
54     def nonlinear(u_phys: np.ndarray) -> np.ndarray:
55         U = np.fft.fft(u_phys)
56         ux = np.fft.ifft(ik * U).real
57         return -6.0 * u_phys * ux
58
59     # Time stepping & saving
60     Nt = int(np.round(tmax / dt))
61     if save_steps > Nt + 1:
62         save_steps = Nt + 1
63     save_every = max(1, Nt // (save_steps - 1))
64
65     t_save = []
66     U_save = []
67
68     def save(tn: float, u_phys: np.ndarray) -> None:
69         t_save.append(tn)
70         U_save.append(u_phys.copy())
71
72     t = 0.0
73     save(t, u)
74
75     U_hat = np.fft.fft(u)
76     for n in range(1, Nt + 1):
77         Nv = nonlinear(u)
78         a = np.fft.ifft(E2 * U_hat + Q * np.fft.fft(Nv)).real
79         Na = nonlinear(a)
80         b = np.fft.ifft(E2 * U_hat + Q * np.fft.fft(Na)).real
81         Nb = nonlinear(b)
82         c_stage = np.fft.ifft(
83             E * U_hat + Q * np.fft.fft(2 * Nb - Nv)
84         ).real
85         Nc = nonlinear(c_stage)
86
87         U_hat = (
88             E * U_hat
89             + np.fft.fft(Nv) * f1
90             + 2 * np.fft.fft(Na + Nb) * f2
91             + np.fft.fft(Nc) * f3
92         )
93         u = np.fft.ifft(U_hat).real
94         t = n * dt
95
96         if n % save_every == 0 or n == Nt:
97             save(t, u)
98
99     U_save = np.array(U_save)
100     t_save = np.array(t_save)
101

```

```

102     return x, t_save, U_save
103
104 # -----グラフ描画関数-----
105
106 def plot_surface(
107     x: np.ndarray,
108     t: np.ndarray,
109     U: np.ndarray,
110     out_path: str,
111     crest_line: bool = True,
112 ) -> None:
113     """3D サーフェスプロットを描画して保存する。"""
114     # Create mesh for surface: X: space, Y: time
115     X, Y = np.meshgrid(x, t)
116
117     fig = plt.figure(figsize=(6.3, 5.4))
118     ax = fig.add_subplot(111, projection="3d")
119
120     # 時間軸 (Y) を相対的に長く見せるためのアスペクト調整 (x:y:z)
121     try:
122         ax.set_box_aspect((1.0, 1.4, 0.9))
123     except Exception:
124         pass
125
126     surf = ax.plot_surface(
127         X,
128         Y,
129         U,
130         cmap="viridis",
131         linewidth=0,
132         antialiased=True,
133     )
134
135     # Z軸が見切れないように余白を持たせて設定
136     zmin = float(np.min(U))
137     zmax = float(np.max(U))
138     pad = 0.05 * (zmax - zmin + 1e-12)
139     ax.set_zlim(zmin - pad, zmax + pad)
140
141     # 範囲を明示
142     ax.set_xlim(float(x.min()), float(x.max()))
143     ax.set_ylim(float(t.min()), float(t.max()))
144
145     ax.set_xlabel("x")
146     # 標準の y ラベルは隠して独自配置 (余白削減)
147     ax.set_ylabel("")
148     ax.set_zlabel("u", labelpad=10)
149
150     # Z軸ラベルが視点で隠れないように画面基準で回転を固定
151     try:
152         ax.zaxis.set_rotate_label(False)
153         ax.zaxis.label.set_rotation(90)
154         ax.zaxis.label.set_color("black")

```

```

155     except Exception:
156         pass
157
158     fig.colorbar(surf, shrink=0.6, aspect=12, pad=0.1)
159
160     if crest_line:
161         idxs = np.argmax(U, axis=1)
162         x_crest = x[idxs]
163         z_crest = U[np.arange(len(t)), idxs]
164         ax.plot(
165             x_crest,
166             t,
167             z_crest,
168             color="red",
169             linewidth=2.0,
170             label="crest",
171         )
172         ax.legend(loc="upper right")
173
174     ax.view_init(elev=22, azim=-95)
175
176     fig.subplots_adjust(left=0.02, right=0.995, bottom=0.1, top=0.93)
177
178     try:
179         ax.zaxis.set_rotate_label(False)
180         ax.zaxis.label.set_rotation(90)
181     except Exception:
182         pass
183
184     # timeのラベルの位置!!
185     fig.text(
186         0.1,
187         0.2,
188         "time",
189         rotation=90,
190         va="center",
191         ha="center",
192         fontsize=10,
193     )
194
195     plt.savefig(out_path, dpi=400)
196     plt.close(fig)
197
198 def main() -> None:
199     x, t, U = kdv_etdrk4(
200         Lx=40.0,
201         N=512,
202         dt=0.005,
203         tmax=12.0,
204         save_steps=300,
205         c=2.0,
206         x0=5.0,
207     )

```

```
208     out_path = os.path.join(os.path.dirname(__file__), "KdV_surface.png")
209     plot_surface(x, t, U, out_path)
210     print(f"Saved 3D surface figure: {out_path}")
211
212
213 if __name__ == "__main__":
214     main()
```

10.2 波動方程式と Klein-Gordon 方程式と電信方程式

10.3 シュレーディンガー方程式と Klein-Gordon 方程式と KdV 方程式

10.4 参考文献

- 和達美樹『岩波講座現代の物理学 非線形波動』岩波書店 1992 年