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. これ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 として,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考えられる.

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

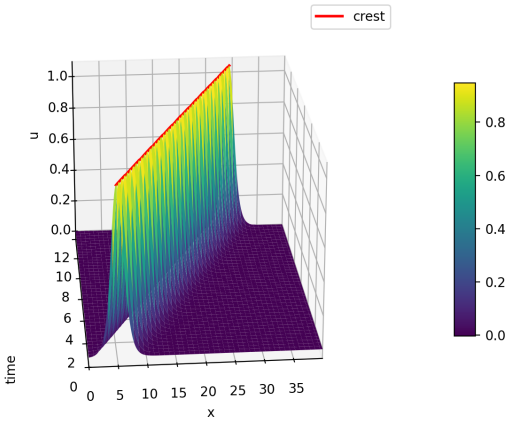

図 1. KdV 方程式の孤立波解の例

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

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

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: Nonlinear 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。

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

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。

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

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。

非線形な電信方程式を導出するためには、完成された式（2 階微分方程式）に代入するのではなく、その導出元となる**2 つの基礎方程式（1 階連立偏微分方程式）**に立ち返る必要がある。分布定数回路における電圧 V と電流 I の関係は、以下の 2 式で記述される。

- 電圧降下の式（線路の直列インピーダンスによる）：

$$-\frac{\partial V}{\partial x} = L\frac{\partial I}{\partial t} + RI \tag{6}$$

- 電流減少の式(線路の並列アドミタンスによる)：

$$-\frac{\partial I}{\partial x} = \frac{\partial Q}{\partial t} + GV \tag{7}$$

ここで、式 7 に含まれる変位電流項 $\frac{\partial Q}{\partial t}$ について考える。キャパシタンスが電圧依存性 $C(V)$ を持つ場合、電荷 Q の時間変化は連鎖律により以下のように展開される。

$$\frac{\partial Q}{\partial t} = \frac{dQ}{dV}\frac{\partial V}{\partial t} = C(V)\frac{\partial V}{\partial t} \tag{8}$$

これを式 7 に代入すると、電流の変化は次式となる。

$$-\frac{\partial I}{\partial x} = C(V)\frac{\partial V}{\partial t} + GV \tag{9}$$

次に、電圧と電流を一本化して V だけの式（波動方程式）にするため、もう一方の基礎方程式である式 6 を変形する。式 6 の両辺を x で偏微分すると、以下の形になる。

$$-\frac{\partial^2 V}{\partial x^2} = L\frac{\partial}{\partial t}\left(\frac{\partial I}{\partial x}\right) + R\frac{\partial I}{\partial x} \tag{10}$$

この式の右辺にある $\frac{\partial I}{\partial x}$ の箇所に、先ほど導いた式 9 を代入することで、電流 I を消去できる。ここで特に注意すべきは、第 1 項の時間微分である。代入を行うと、積の微分と連鎖律により非線形項が現れる。

$$\frac{\partial}{\partial t}\left(C(V)\frac{\partial V}{\partial t}\right) = C(V)\frac{\partial^2 V}{\partial t^2} + \frac{dC(V)}{dV}\left(\frac{\partial V}{\partial t}\right)^2 \tag{11}$$

以上を整理することで、以下の拡張された電信方程式が得られる。

$$\frac{\partial^2 V}{\partial x^2} = L\left[C(V)\frac{\partial^2 V}{\partial t^2} + \frac{dC(V)}{dV}\left(\frac{\partial V}{\partial t}\right)^2\right] + (RC(V) + GL)\frac{\partial V}{\partial t} + GRV \tag{12}$$

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.

```python
import os
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401  # needed for 3D

# るんげくったぱらめた
def kdv_etdrk4(
    Lx: float = 40.0,
    N: int = 512,
    dt: float = 0.005,
    tmax: float = 12.0,
    save_steps: int = 300,
    c: float = 2.0,
    x0: float = 5.0,
):
# KdV方程式を解く！！
    # Grid: 0 から始まる周期区間 x in [0, Lx)
    x = np.linspace(0.0, Lx, N, endpoint=False)

    # Wavenumbers for domain length Lx
    k = 2 * np.pi * np.fft.fftfreq(N, d=Lx / N)
    ik = 1j * k
    L = 1j * k**3  # Linear operator in Fourier space: + i k^3

    # Initial condition: 1-soliton for KdV (coefficient 6)
    # u(x,0) = c/2 * sech^2( sqrt(c)/2 (x - x0) )
    def sech(z):
        return 1.0 / np.cosh(z)

    u = 0.5 * c * sech(0.5 * np.sqrt(c) * (x - x0)) ** 2

    # ETDRK4 precomputations (Kassam & Trefethen 2005)
    E = np.exp(dt * L)
    E2 = np.exp(dt * L / 2)
    M = 16  # contour points for phi functions
    r = np.exp(1j * np.pi * (np.arange(1, M + 1) - 0.5) / M)
    LR = dt * L[:, None] + r[None, :]

    Q = dt * np.mean((np.exp(LR / 2) - 1.0) / LR, axis=1)
    f1 = dt * np.mean(
        (-4 - LR + np.exp(LR) * (4 - 3 * LR + LR**2)) / LR**3,
        axis=1,
    )
    f2 = dt * np.mean(
        (2 + LR + np.exp(LR) * (-2 + LR)) / LR**3,
        axis=1,
    )
    f3 = dt * np.mean(
```

```
        (-4 - 3 * LR - LR**2 + np.exp(LR) * (4 - LR)) / LR**3,
        axis=1,
    )

    # Nonlinear term N(u) = -6 u u_x (compute u_x via spectral derivative)
    def nonlinear(u_phys: np.ndarray) -> np.ndarray:
        U = np.fft.fft(u_phys)
        ux = np.fft.ifft(ik * U).real
        return -6.0 * u_phys * ux

    # Time stepping & saving
    Nt = int(np.round(tmax / dt))
    if save_steps > Nt + 1:
        save_steps = Nt + 1
    save_every = max(1, Nt // (save_steps - 1))

    t_save = []
    U_save = []

    def save(tn: float, u_phys: np.ndarray) -> None:
        t_save.append(tn)
        U_save.append(u_phys.copy())

    t = 0.0
    save(t, u)

    U_hat = np.fft.fft(u)
    for n in range(1, Nt + 1):
        Nv = nonlinear(u)
        a = np.fft.ifft(E2 * U_hat + Q * np.fft.fft(Nv)).real
        Na = nonlinear(a)
        b = np.fft.ifft(E2 * U_hat + Q * np.fft.fft(Na)).real
        Nb = nonlinear(b)
        c_stage = np.fft.ifft(
            E * U_hat + Q * np.fft.fft(2 * Nb - Nv)
        ).real
        Nc = nonlinear(c_stage)

        U_hat = (
            E * U_hat
            + np.fft.fft(Nv) * f1
            + 2 * np.fft.fft(Na + Nb) * f2
            + np.fft.fft(Nc) * f3
        )
        u = np.fft.ifft(U_hat).real
        t = n * dt

        if n % save_every == 0 or n == Nt:
            save(t, u)

    U_save = np.array(U_save)
    t_save = np.array(t_save)
```

```
    return x, t_save, U_save

# ------------------グラフ描画関数------------------

def plot_surface(
    x: np.ndarray,
    t: np.ndarray,
    U: np.ndarray,
    out_path: str,
    crest_line: bool = True,
) -> None:
    """3D サーフェスプロットを描画して保存する。"""
    # Create mesh for surface: X: space, Y: time
    X, Y = np.meshgrid(x, t)

    fig = plt.figure(figsize=(6.3,5.4))
    ax = fig.add_subplot(111, projection="3d")

    # 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)
    try:
        ax.set_box_aspect((1.0, 1.4, 0.9))
    except Exception:
        pass

    surf = ax.plot_surface(
        X,
        Y,
        U,
        cmap="viridis",
        linewidth=0,
        antialiased=True,
    )

    # Z軸が見切れないように余白を持たせて設定
    zmin = float(np.min(U))
    zmax = float(np.max(U))
    pad = 0.05 * (zmax - zmin + 1e-12)
    ax.set_zlim(zmin - pad, zmax + pad)

    # 範囲を明示
    ax.set_xlim(float(x.min()), float(x.max()))
    ax.set_ylim(float(t.min()), float(t.max()))

    ax.set_xlabel("x")
    # 標準の y ラベルは隠して独自配置（余白削減）
    ax.set_ylabel("")
    ax.set_zlabel("u", labelpad=10)

    # 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
    try:
        ax.zaxis.set_rotate_label(False)
        ax.zaxis.label.set_rotation(90)
        ax.zaxis.label.set_color("black")
```

```
    except Exception:
        pass

    fig.colorbar(surf, shrink=0.6, aspect=12, pad=0.1)

    if crest_line:
        idxs = np.argmax(U, axis=1)
        x_crest = x[idxs]
        z_crest = U[np.arange(len(t)), idxs]
        ax.plot(
            x_crest,
            t,
            z_crest,
            color="red",
            linewidth=2.0,
            label="crest",
        )
        ax.legend(loc="upper right")

    ax.view_init(elev=22, azim=-95)

    fig.subplots_adjust(left=0.02, right=0.995, bottom=0.1, top=0.93)

    try:
        ax.zaxis.set_rotate_label(False)
        ax.zaxis.label.set_rotation(90)
    except Exception:
        pass

    # timeのラベルの位置！！
    fig.text(
        0.1,
        0.2,
        "time",
        rotation=90,
        va="center",
        ha="center",
        fontsize=10,
    )

    plt.savefig(out_path, dpi=400)
    plt.close(fig)

def main() -> None:
    x, t, U = kdv_etdrk4(
        Lx=40.0,
        N=512,
        dt=0.005,
        tmax=12.0,
        save_steps=300,
        c=2.0,
        x0=5.0,
    )
```

```
    out_path = os.path.join(os.path.dirname(__file__), "KdV_surface.png")
    plot_surface(x, t, U, out_path)
    print(f"Saved 3D surface figure: {out_path}")


if __name__ == "__main__":
    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