1　LU 分解を利用した電気回路の過渡現象の解き方

電気回路の過渡現象を求めるには，回路の微分方程式をたてて，それに基づいて数値計算をすればよい。問題は，どのような数値計算の方法を使うかである。runge-kutta 一本槍というのは，芸がない。ここでは，微分方程式をそのまま解く，LU 分解法を解説する。このやり方は，回路シミュレータ pspice, psim などで使われている方法である。

1.1 電圧形三相インバータの回路方程式

交流負荷が RL の電圧形三相インバータの回路は，図１のようになる。この回路の微分方程式を求めると次式となる。

$$L_u \frac{di_u}{dt} + R_u i_u - (L_v \frac{di_v}{dt} + R_v i_v) = e_u - e_v \quad \cdots\cdots (1)$$

$$L_v \frac{di_v}{dt} + R_v i_v - (L_w \frac{di_w}{dt} + R_w i_w) = e_v - e_w \quad \cdots\cdots (2)$$

$$\frac{d}{dt}(i_u + i_v + i_w) = 0 \quad \cdots\cdots (3)$$

上式で工夫したのは，(3)式である。この式は，三相星形結線で中性点が接地されていない場合に $i_u + i_v + i_w = 0$ になるが，その式を時間微分しただけである。（あとで分かるように，(3)式を(1)式や(2)式と同じ方針で立てると，失敗する。）

上式を次のように変形する。時間微分の項を左辺に残して，それ以外を右辺に移項する。さらに，両辺に dt を乗じて次式を得る。

$$\begin{pmatrix} L_u & -L_v & 0 \\ 0 & L_v & -L_w \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} di_u \\ di_v \\ di_w \end{pmatrix} = \begin{pmatrix} e_u - R_u i_u - (e_v - R_v i_v) \\ e_v - R_v i_v - (e_w - R_w i_w) \\ 0 \end{pmatrix} \times dt \quad \cdots\cdots (4)$$

ここで，三相星形結線で中性点が接地されていない場合に $i_u + i_v + i_w = 0$ だから，(3)式を(1)式や(2)式と同じ方針で立てると，(1)，(2)，(3)式は互いに独立ではない。その結果，(4)式の左辺の係数行列に相当する部分に逆行列が存在しなくなる。つまり，(3)式を作るときに，ちょっと工夫したのは，(4)式左辺の係数行列の逆行列があるようにするためだったのである。

ここで，(4)式の $(di_u \quad di_v \quad di_w)^T$ の計算は LU 分解を使うと高速に出来る。（LU 分解のプログラムを自分で作ってもよいが，添付のソースリスト中にあるように，インターネットで拾って来れば間違いが少ない。たとえば，http://math.nist.gov/tnt）

そして，$(di_u \quad di_v \quad di_w)^T$ が計算できてしまえば，次のように電流を更新すればよい。

$$i_u = i_u + di_u \quad \text{（v,w 相も同じような式だから省略）} \quad \cdots\cdots (5)$$

次ページ以降に，ソースリストを添付する。　＜＜＜＜おしまい＞＞＞＞

```
// triPWMinv100.cpp
// 三相ＰＷＭインバータを三相のまま解いて，中性点電圧を求める。
//                          三角波形発生関数。tiangleWave()
//                          version 1.00 2001.04.16 by SK 不要な部分を除いて簡単化
//                          version 0.02 2000.03.09 by SK
//                          ver. 0.01 1998.11.17

#include <cassert>
#include <cmath>

double      // -1〜+1 の間の三角波の値
triWave2(double t, double tp, double wid = 0.5)
        // PWM 用の三角波を作る関数　cf. 研究ノート vol.D-13 p.68
        // t = 時刻，tp = 周期，wid = 半周期の割合（wid=0.5 のとき対称波形）
        // 戻り値：[-1, 1] の値。
        // ver.0.00 2000.03.09
{
        double tm = fmod(t, tp); // t / tp の余り

        if( wid == 0.0)
                return 1.0 - 2.0/((1.0-wid)*tp) * tm;     // wid=0 なら mode2 だけ
        else if ( wid == 1.0 )
                return 1.0 + (2.0/wid)*(tm/tp - 1.0);     // mode3 だけ
        else
        {
                if ( tm < (1.0-wid)*tp )
                        return 1.0 - 2.0/((1.0-wid)*tp) * tm;     // mode2
                else
                        return 1.0 + (2.0/wid)*(tm/tp - 1.0);     // mode3
        }
}

inline double triangleWave(double t, double tp)
{
        //return triWave1(t, tp, 0.5);              // 昔はあったけど，今はない！
        return triWave2(t, tp);
}

// 三相 PWM インバータの電流・電圧を三相のまま解く
//          cf. 研究ノート vol.D-12 p.65
//
// TNT, LuDetInv.h を使用。for LU_factor(), LU_solve()

        // TNT はただでもらえる行列計算ライブラリ(C++専用)である。
        //      down load:   http://math.nist.gov/tnt

        // TNT のインクルードファイルにパスを通しておくこと！！
        //      visual C++ ver.6.0 の場合は次のようにする。
        //          ツール＞オプション＞ディレクトリ＞インクルードファイル
        //          とやって，そこに TNT のフォルダに一つ上のディレクトリ
        //          を書いておく。
        //          例：x:¥users¥kon¥C¥tnt なら，x:¥users¥kon¥C と書く。

#include <iostream>
#include <fstream>
#include <cstdlib>

#include "tnt/stopwatch.h"              // in TNT 2001.04.16 by SK

TNT::stopwatch Watch;                   // Watch のインスタンスを作る。

#include "tnt/tnt.h"        // TNT heder files
#include "tnt/vec.h"
#include "tnt/cmat.h"
#include "tnt/lu.h"

using namespace TNT;
using namespace std;

const double PI = 3.14159265358979323846;       //2.0 * asin(1.0);
const double PI2 = 2.0 * PI;

int main()
{
        double t, ts=0.000001, te=0.05, wave;  // ts=計算刻み
        double freq = 50.0;                                                     // 電圧指令の周波数
        double w = PI2 * freq;
        double tcarrier = 1.0 / 2000.0;                                         // 三角波キャリアの周期
        double Ed = 100.0, Ed2 = Ed/2.0;                        // Ed 直流リンク全電圧
        double phiV = PI2/3.0, phiW = 2.0*phiV;
        double eu, ev, ew                                                       ;                                                       // 相電圧，直流中間基準
        double refu, refv, refw;                                                // 単位振幅の電圧指令
        double iu=0.0, iv=0.0, iw=0.0;                                          // 相電流
        double diudt, divdt, diwdt;                                                     // 相電流の時間微分
        double enu=0.0, env=0.0, enw=0.0;                       // 中性点電位
        double Ru=20.0, Rv=Ru, Rw=Ru;                                           // たまたま，三相とも同じ
        double Lu=0.0100, Lv=Lu, Lw=Lu;                                         // 同上
```

```
	TNT::Matrix<double> z(3,3);                                    // 回路方程式左辺の係数行列
	z(1,1) = Lu;  z(1,2) = -Lv; z(1,3) = 0.0;
	z(2,1) = 0.0; z(2,2) = Lv;  z(2,3) = -Lw;
	z(3,1) = 1.0; z(3,2) = 1.0; z(3,3) = 1.0;

	TNT::Vector<int> index(3);                                     // LU 分解  LU_factor()用
	TNT::Matrix<double> zdcmp(3,3);                                // LU 分解  LU_factor()用
	TNT::Vector<double> di(3), edt(3);                 // i  の微小変化分

	zdcmp = z;

	int d = LU_factor(zdcmp, index);                   //  この例ではＬＵ分解は 1 回のみ！


	ofstream outFile("triangle.dat", ios::out);
	if(!outFile) {
		cerr << "triangle.dat  が開けません" << endl;
		exit(-100);
	}
	//  データファイルの 1 行目にタイトルを書く
	outFile << "t ¥t wave ¥t refu ¥t eu ¥t ev ¥t ew ¥t iu ¥t iv ¥t iw ¥t enu ¥t env ¥t enw ¥n";

	///////////////
	// main loop //
	///////////////
	long count = 0;
	long countCRT=2000;                                            // countCRT 回毎に画面にデータ出力する
	Watch.reset(); Watch.start();                      //  ストップウォッチをリセット

	for(t = 0.0; t < te; t += ts)
	{
		//  電圧指令値を作る
		double amp = 0.8;
		refu = amp * cos(w*t);                                     //  電圧指令値
		refv = amp * cos(w*t - phiV);
		refw = amp * cos(w*t - phiW);

		//  電圧指令値と三角波の値を比較して，on/off パタンを作る
		wave = triangleWave(t, tcarrier);
		eu = ( refu >= wave ? Ed2 : -Ed2);       // on/off  相電圧
		ev = ( refv >= wave ? Ed2 : -Ed2);
		ew = ( refw >= wave ? Ed2 : -Ed2);

		//  画面とファイルに出力する
		if (count % countCRT == 0) {
			cout << t << "¥t" << eu << "¥t" << iu << "¥t" << iu+iv+iw << endl;
		}
		if ( count % 2 == 0) {
			outFile << t << "¥t" << wave << "¥t" << refu << "¥t" << ¥
				eu << "¥t" << ev << "¥t" << ew << "¥t" <<          ¥
				iu << "¥t" << iv << "¥t" << iw << "¥t" <<          ¥
				enu << "¥t" << env << "¥t" << enw << "¥n";
		}

		//  回路方程式の右辺を計算する
		edt(1) = ( eu - Ru*iu - (ev - Rv*iv) ) * ts;               //  連立Ｅｑ右辺
		edt(2) = ( ev - Rv*iv - (ew - Rw*iw) ) * ts;
		edt(3) = 0.0;

		//  回路方程式（連立一次方程式）を LU 分解で解く。
		LU_solve(zdcmp, index, edt);                               // edt => di  が返る

		//  電流微分値の計算。edt() = di()であることに注意！
		diudt = edt(1)/ts;
		divdt = edt(2)/ts;
		diwdt = edt(3)/ts;

		//  確認のため，中性点電圧の計算
		enu = eu - Ru*iu - Lu * diudt;
		env = ev - Rv*iv - Lv * divdt;
		enw = ew - Rw*iw - Lw * diwdt;

		//  電流の更新
		iu += edt(1);
		iv += edt(2);
		iw += edt(3);
		count++;
	}

	//  計算を終了して，後始末。
	outFile << endl;
	cout << Watch.read() << "  秒です。" << endl;
	return 0;
}
// end of file
```