車両重量計測システム
（Bridge Weigh-in-Motion）

コード解説資料 Ver3.1

目　次

１．車両重量の計算手法について

１.１ 概要

車両が橋梁を通過する際に、その重量によって橋梁部材にひずみが生じる。主桁の下フランジに生じるひずみは、あらかじめ試験車の走行によって推定した各主桁の曲げ剛性(EZ)を用いて時間tの関数ε(t)として表わすことができる。

BWIM では、各主桁の曲げ剛性より計算される各主桁の推定ひずみε(t)と実測ひずみε’(t)との誤差が最小になるような軸重 P を、最適化の手法を用いて決定する。以下に重量計算の流れを示す。

・重量計算の流れ

1)計算に用いる入力データ

①橋長，ゲージ貼付位置，試験車両などの諸元

②主桁下フランジのひずみ応答

③床版下面のひずみ応答

2)主桁の曲げ剛性(EZ)の推定

荷重諸元（軸重，軸間距離）が既知である試験車を走行させた場合に生じる主桁下フランジのひずみ応答により、各主桁の曲げ剛性（ＥＺ）を推定する。

3)車両位置の判別

垂直補剛材上端部または床版下面のひずみより、走行車両の速度，軸数，軸間距離，位置（橋軸方向・橋軸直角方向）を判別する。（車両は橋梁通過時には加速、減速しないと仮定）

4)車両重量の計算

各主桁の曲げ剛性（ＥＺ）より計算される各主桁の推定ひずみε（ｔ）と実測ひずみε’（ｔ）との差が最小になるような軸重Ｐを、最適化の手法を用いて決定する。

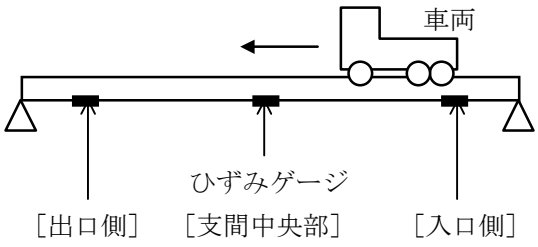


図１　測定イメージ図

1.2 BWＩMシステムの基本条件

本システムの基本条件を以下に示す。

表１　BWＩMシステムの基本条件

モデル化	・橋梁は単純桁とする。 ・斜角、キャンバーは無視する。 ・走行車両はｎ個の集中荷重が等速度で移動すると仮定。
車線数	最大４車線
適用車両	２軸から６軸（＊１）
連行・並走	一度に計算可能な台数は１車線２台連行×４車線の並走で最大８台（＊２）とする。
その他	・タンデム、トリプル軸の軸重は等しい。（＊３） 車両が橋梁を通過する時間は１０秒以内であること。１０秒以上かかる渋滞時は計算不能。

＊１：軸間隔が特殊な車両通過時は軸間隔と車両間隔を認識できない場合がある。

＊２：全ての連行・並走する車両が橋梁を通過する時間は１０秒以内であること。ただし、最初の車両が橋梁を通過後、後続の車両が橋梁に進入した場合は連行とは考えないで、別々に計算する。

＊３：計測誤差、モデル化誤差を含んだ実測ひずみを用いる場合は、未知数が多くなる程解が得られにくいので、未知数を減らすためにこの条件を設けた。

１.３ 計算理論

主桁は単純梁、また走行車両はn個の集中荷重が等速度で移動すると仮定する。また、各車線を通過する荷重はすべての桁に作用するので、並走時の桁ひずみ応答は各車線を通過するそれぞれの荷重によるひずみ応答の重ね合わせと考える。

本WIMシステムでは、初めに重量が既知の試験車両を走行させてその時の主桁ひずみ応答波形を計測し、各車線通過時における各主桁の曲げ剛性 EZij を推定する。次に、このEZij を用いて各車線の各軸重Pi を算定する。

車両が橋梁を通過する時に生じる主桁中央のひずみ ε (x)は次式で表される。

$$\varepsilon(x) = \frac{1}{EZ} \cdot \frac{P_i \cdot (x + l_i)}{2} \qquad \cdots式1$$

（$0 \le x + l_i \le 1/2\ell$ のとき）

$$\varepsilon(x) = \frac{1}{EZ} \cdot \frac{P_i \cdot (\ell - x - l_i)}{2} \qquad \cdots式2$$

（$1/2\ell \le x + l_i \le \ell$ のとき）

但し、

x:支点からの距離
EZ:曲げ剛性
Pi:i 番目の軸重
li:1軸とi 軸間の距離

速度の異なる複数の車両による桁ひずみ応答式を考える場合には、ひずみの関数は支点からの距離xとするよりも時刻tにした方が扱いやすい。さらに実測ひずみ応答値 ε ’(t)はdt秒間隔で得られるので、変数を支点からの距離 X ではなくて時刻tとする。

k主桁のひずみ応答 ε k(t)は次式で表される。

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\left(\frac{1}{EZ_{ki}} \cdot \frac{P_{ij} \cdot x'}{2}\right) = \varepsilon_k(t) \qquad \cdots式3$$

$$x = (t - t_i) \cdot V_i + l_{ij}$$

$$x' = x$$

（$0 \le x \le 1/2\ell$ のとき）

$$x' = l - x$$

（$1/2\ell \le x \le \ell$ のとき）

但し、

n:桁の数
m:軸の数

EZki:i 主桁上を通過した時の k 主桁の曲げ剛性
ε k(t):t時刻のひずみ応答値
ti:i 主桁上の1軸がIN側支点を通過する時刻(s)
Vi:i 主桁上車両の速度(m/s)
ij:i 主桁上車両のj番目の軸重(t)
lij:i 主桁上車両の1軸とj軸間の距離(m)

以下に式3を用いて計算した推定ひずみと実測ひずみの例を示す。

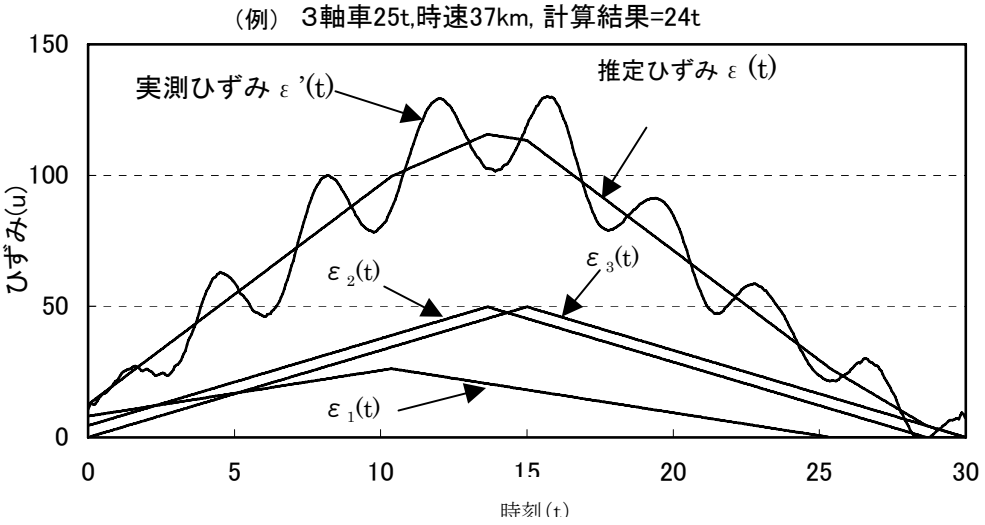


図2　推定ひずみと実測ひずみの例

2．主要ロジックの解説

2.1 概要

車両重量を精度良く算定するためには、橋梁上を通過している車両全てを認識する必要がある。しかし、現実的には乗用車等の軽い車両によるひずみは小さいので検出は難しく、また、小さなひずみを対象にするとノイズを軸と誤認識してしまうこともあるので、本システムでは誤認識を少なくするために試験車通過時のひずみを元に車両検出レベルを設定している。以降この車両検出レベル以下を不感帯と呼ぶ。

不感帯の初期値は以下の通りとする。

・主桁ひずみの不感帯は4t相当以下

（4t/試験車重量×試験車通過時の主桁ひずみ値）

・軸検出用ひずみの不感帯は試験車通過時の最大ひずみの33％

但し、上記不感帯を用いてこの試験車の車両認識処理を試みた結果、車両認識が正確にできない場合は、不感帯を90％に再設定して試験車の軸が全て正しく検出できるレベルまで繰り返し小さくする。

次に、橋長30m、車両は時速20kmで連行していることを想定すると、例えば3軸車2台連行の場合は（1軸目から最後の軸までの距離を20mとする）、1台目が橋梁に乗ってから2台目が抜けるまでに要する時間は約9秒間である。よって、この例では無限に続くデータの内で、1台目の軸が橋梁に入った時刻から、2台目が橋梁を抜けるまでの9秒間のデータを用いれば計算できる。さらに、車速が20km以上の場合は9秒以下の時間だけを対象にすればよい。故に、本システムでは1度に処理するデータ量を10秒間とした。計算は1秒間隔で、現在から過去10秒間のデータを用いることにした。以降この10秒間の計算データ領域を FIFO と呼ぶ。

FIFOを10秒と決定したので、車両が橋梁を通過するのに10秒以上かかるような低速走行している場合は計算不能となる。この状態を本システムでは渋滞時と定義する。

毎秒 WIM 計算処理を行う為に、1秒毎にタイマー割り込みを発生させて、この割り込みルーチン内でWIM計算を行う。このように1秒毎に処理するとFIFO の中には前回すでに車両認識できたデータが含まれていることがあるので、認識できた車両の最後の軸が出口側ひずみ計を通過した時刻（以降処理済みインデックスと呼ぶ）を記憶して、次回の計算はこの処理済みインデックス以降のデータを用いることにする。

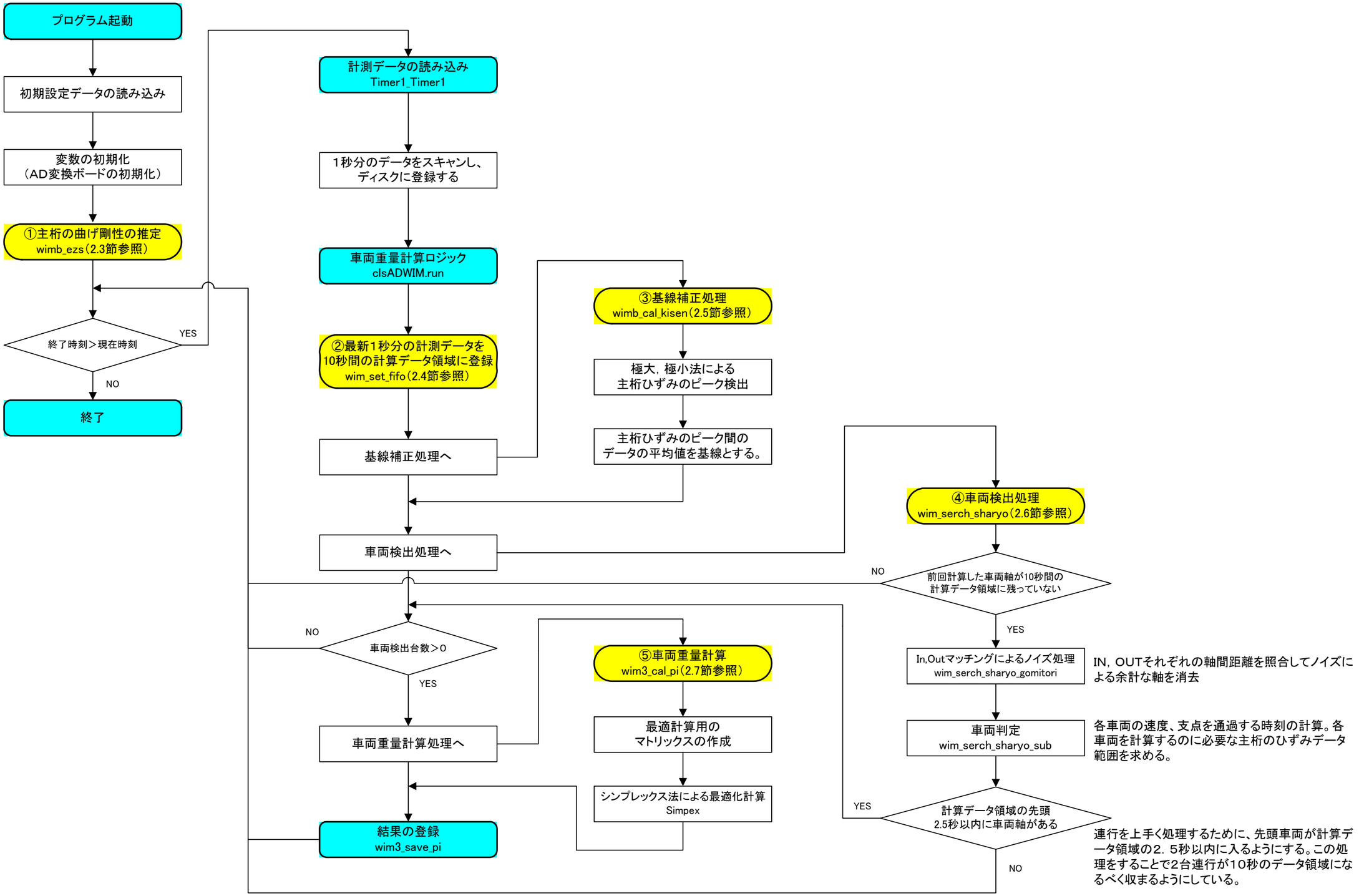
以下に本システムの核となる毎秒処理の流れを示す。

1. 1秒分の AD データをスキャンして FIFO に登録。
2. 処理済みインデックスが先頭から200（1秒）以上の場合は古い1秒分のデータを消去することができるので何もせずにリターン。
3. 基線補正処理。（温度変化によるひずみの除去）
4. 車両検出。車両検出によって得られた入口側の1軸目のインデックスが200以内ならリターン。（なるべく連行の始めから終わりまでをFIFOに入れる為。）
5. 車両重量計算。
6. 結果の登録。

2.2 主なサブルーチンの構成

主なサブルーチンの構成を次頁に示す。

＜主なサブルーチンの構成＞



２.３ 主桁の曲げ剛性の推定

（1）ロジックの解説

荷重諸元（軸重，軸間距離）が既知である試験車を走行させた場合に生じる主桁下フランジのひずみ応答と前章に示す式３を用いて計算した各主桁の推定ひずみとの誤差が最少となるような各主桁の曲げ剛性（ＥＺ）を推定する。

ｋ主桁の推定ひずみ（ε）と実測ひずみ（ε’）との誤差（ek）は次式で表される。

$$e_k = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \left(\frac{1}{EZ_{ki}} \cdot \frac{P_{ij} \cdot x'}{2} \right) - \varepsilon'_k(t) \right)^2 \quad \cdots式４$$

$$x = (t - t_i) \cdot V_i + l_{ij}$$

$x' = x$

（$0 \le x \le 1/2\ell$ のとき）

$x' = l - x$

（$1/2\ell \le x \le \ell$ のとき）

各主桁それぞれについて、誤差ｅｋ が最少になるような主桁の曲げ剛性ＥＺをニュートン法により求める。

(2)主要コードの解説

＜関連するコード＞	＜解　説＞
Public Sub wimb_ezs(npic%, Pic As PictureBox, offsetwave&, chd%(), offset&, num&)	試験車データからＥＺを推定する処理
j = offset * clsMyWaveData.ChLast0 st = wim_set_fifo(inum&, dat16(j)) If st Then GoTo errt st = wim_set_indexline(gIndexLine&)	ｆｉｆｏにひずみデータをセット。 （ｆｉｆｏ：先に入力されたデータから先に読み出されるデータ構造）

For I = 1 To 10 '不感帯の再調整を再度試みる
wim3_init_sharyo '車両データクリア
st = win_set_kajyuu(gNumberJiku&, gCar_L(0), gCar_P(0))
st = wim_serch_sharyo(0, inum&, 1)　'0からｉ個、バッファが足りなくても計算する。
'MsgBox st
If st = 1 Then
st = win_get_kajyuu(N, l(0), p(0))
Else
N = 0
End If
	不感帯の自動設定（再調整は１０回）
	（不感帯：軸検出するときの最低ひずみ値。この値以上のひずみから軸を検出する。）
	試験車データを登録。
	車両検出。

If N ◇ 3 Then
'MsgBox "車両認識ができませんでした。不感帯を再調整します。"
For j = 1 To 2　'不感帯の再調整
fukantai&(gIndexLine&, j) = fukantai&(gIndexLine&, j) * 0.9
clsMyWaveData.listindex = gIndexArray(gIndexLine&, j)
clsMyWaveData.tokusd = fukantai&(gIndexLine&, j)
Next j
st = wim_set_fukantai(gIndexLine&, fukantai&(gIndexLine&, 0), fukantai&(gIndexLine&, 1), fukantai&(gIndexLine&, 2))
'wimb_initialize
Else
Exit For
End If
	試験車として認識できなかったら不感帯の再調整を試みる。
	
	不感帯をセット。

Next I	
If N ◇ 3 Then GoTo errt For j = 1 To 2　'不感帯の再調整 MyDB.SaveSetting "fukantai-" & Format$(gIndexArray(gIndexLine&, j), "0"), fukantai&(gIndexLine&, j) Next j	
st = wim_save_peek(Len("c:¥wim_peek.txt"), "c:¥wim_peek.txt") DrawKeisanKukan npic%, Pic, offsetwave&, chd%() st = win_set_kajyuu(gNumberJiku&, gCar_L(0), gCar_P(0)) 'wim_cal_yによって書き換えられるので再セット If st Then GoTo errt	検出した軸をグラフィック表示。
'st = wim_set_hizumi_dat() st = wim_cal_ezs() 'st = wim_save_g(Len("c:¥wim_ez.csv"), "c:¥wim_ez.csv")'最後の主桁しかだめ	各主桁のＥＺを計算。
If st Then GoTo errt s$ = "" For j = 0 To num_line - 1 gEZS(gIndexLine&, j) = wim_get_ezs(gIndexLine&, j) s$ = s$ & "EZ(" & gIndexLine& & "," & j & ") = " & gEZS(gIndexLine&, j) & vbCr MyDB.SaveSetting gIndexLine& & "-" & j & "-line-ezs" & gkeisansikino, Format$(gEZS(gIndexLine&, j), "0.00") Next j fMainForm.Shape1.Visible = False MsgBox s$ Exit Sub	ＥＺの表示。
errt: wim_errcode2msg (st) End Sub	エラーの表示。

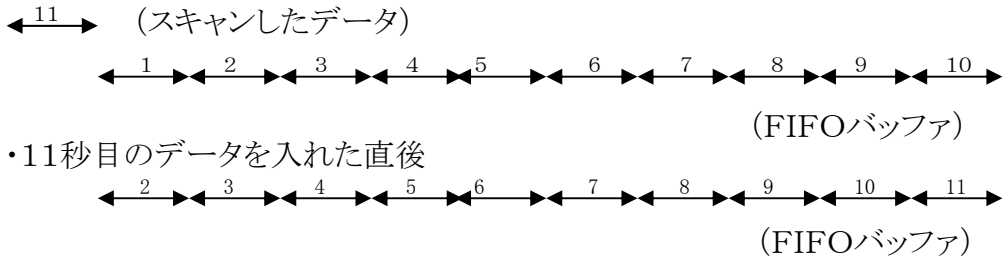
2.4 ＦＩＦＯバッファ

（1）FIFO バッファの解説

FIFO の Full スペルは「 First-In First-Out」である。 FIFOの書き込みアドレスは必ず"0"から始まり、"1"、"2"、"3"、……と順次アドレスが進んでいきます。また、 読み出しアドレスも同様に必ず"0"から始まり、"1"、"2"、"3"、……と順次アドレスが進んでいきます。このように、アドレス"0"から書き込んだデータが書き込んだのと同じ順番で、アドレス"0"から出力するため、「First-In First-Out」と呼ばれる。

このFIFOバッファには以下に示すように常に最新10秒分のAD変換したひずみデータが入っている。

・11秒目のデータを入れる時



・11秒目のデータを入れた直後

(2)主要コードの解説

＜関連するコード＞	＜解　説＞
//----------	ＦＩＦＯバッファにスキャンしたデータを入れる
//ＦＩＦＯバッファセット	
//Num : ADDatの個数	
//----------	
NOMANGLE int CCONV wim_set_fifo(int Num, short *ADDat)	
{	
int iNum;	
short *pFif0, *pFif02, *pADDat;	
gone_sample = Num;	
if(Num >= gNFif0){//Fif0より大きい場合は切り捨て	ＦＩＦＯバッファより入れようとするデータの方が大きい場合は切り捨て
Num = gNFif0;	
gIndexFif0 = 0;	
}else{	
iNum = gIndexFif0 + Num - gNFif0;	新しいデータを入れる分、既存データを移動する。
if(iNum > 0){	
register int i, j;	
gIndexFif0 = gNFif0 - Num;	
for(i = 0; i < gNumCh; i++){	
if(FlagAlloc[i] & F_ALLOC){	
for (j=0, pFif0 = &Fif0[i][0], pFif02=&Fif0[i][iNum]; j<gIndexFif0; j++, pFif0++, pFif02++){	
*pFif0 = *pFif02;	
}	
}	
}	
}	
}	
//新しいデータを入れる	新しいデータをＦＩＦＯバッファに入れる。
register int i, j;	
for(i = 0; i < gNumCh; i++){	
if(FlagAlloc[i] & F_ALLOC){	
FlagAlloc[i] |= F_CALFFT;	
FlagAlloc[i] ^= F_CALFFT;	
for (j=0, pFif0 = &Fif0[i][gIndexFif0], pADDat = &ADDat[i]; j<Num; j++, pFif0++, pADDat += gNumCh){	
*pFif0 = *pADDat;	
}	
}	
}	
}	
gIndexFif0 += Num;//次に書き込むインデックス	
gnum_fifo += Num;//いままでの総数	
if(gNFif0 <= gIndexFif0) fifo_full = 1; //一回はフルになった！	ＦＩＦＯがフルになった場合はfifo_full = 1にする。
//入れたぶん処理済を戻す	
{	
register int i;	処理済みインデックス（前回の計算で車両判定、重量計算が済んでいるインデックス）を修正する。
for(i = 0; i < gNumline; i++){	次回はこのインデックス以降のデータから車両を検出することになる。
if (gline_dat[i].fin_index<Num)	
gline_dat[i].fin_index = 0;	
else	
gline_dat[i].fin_index -= Num;	
}	
}	
return ret_ok;	
}	

2.5 基線補正処理

（1）ロジックの解説

本システムは主桁ひずみの大きさによって車両重量を算定しているので、温度変化によるひずみを除去する必要がある。本基線補正処理では車線に車両が乗っていない時間を見つけて、その時の主桁ひずみを基線（ひずみゼロ）とすることを目標にしている。

以下に2つのアルゴリズムを示す。

a）再計算時の基線補正処理

下図に示す主桁ひずみの極大点a、c点を見つけ、そのa－c間の中点bの前後０．５秒の平均値を基線（ゼロ点）とする。

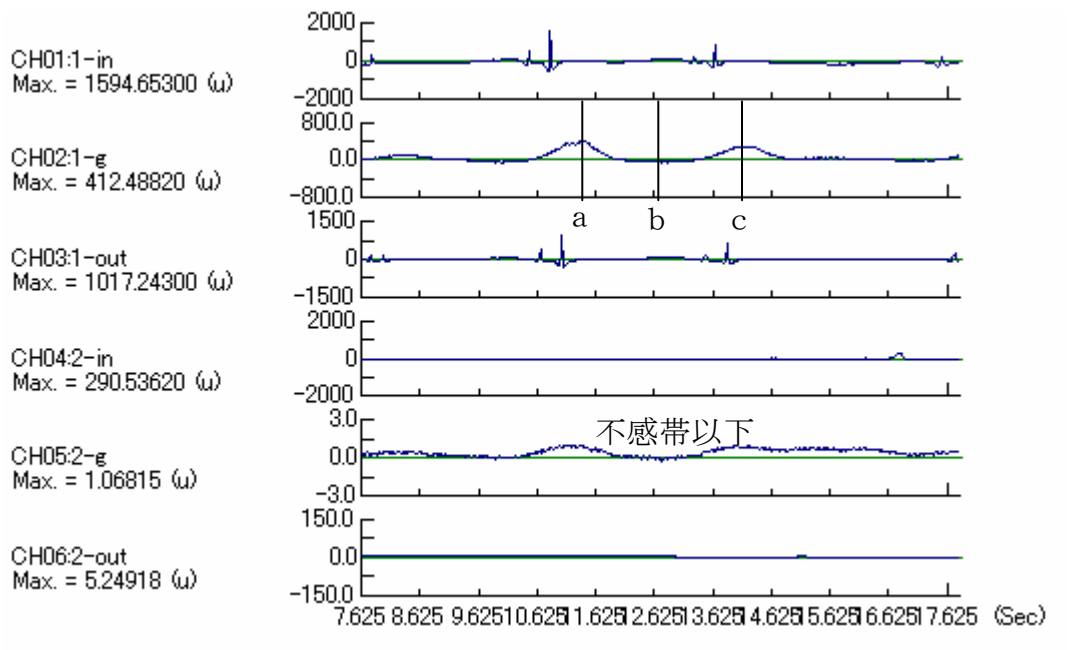


図4　ひずみ波形

複数の車線を考慮する為に以下の条件を付加する。

・主桁ひずみが不感帯以上ある車線においてa－c間の時間が9秒以上の場所を見つけ、さらにこの区間の他の車線の主桁ひずみが不感帯以下だったら、中点bの前後0.5秒の平均値を基線（ゼロ点）とする。この条件に満たない時は基線補正処理を行わない。（初期値もしくは前回の基線値を用いる。）

・全ての車線の主桁ひずみが不感帯以下の場合は先頭から一秒間の平均値を基線（ゼロ点）とする。

◇リアルタイム処理用

　10秒間のデータ全てにおいてP－P値が不感帯以下だったら、この10秒間の平均値を基線とする。このアルゴリズムでは車両がとぎれないような交通量の多い時間帯では基線補正ができないという欠点があるが、計算量が少ないのでリアルタイム処理時に採用している。

(2)主要コードの解説

＜関連するコード＞	＜解　説＞

'主桁の基線補正処理
'バッチ処理用
'indexJ%:基線補正する場所
'   <0:基線補正する場所を自動計算する
'   >0:indexJから１秒間で基線補正する
Function wimb_cal_kisen(indexJ%) As Long
    Dim l%, INDEX&, ch&, nch%, j&, k&, idim&
    Dim ic&, NumCh&
    Dim st&, si&
    Dim numheikin%
    Dim Indexd&(10), vald&(10), count&, NumDat&
 	
    numheikin% = 200
 	
    NumDat = clsMyWaveData.NumDat0
    If NumDat& = 0 Then Exit Function
    If NumDat > 100000 Then NumDat = 100000
    NumCh = clsMyWaveData.ChLast0
 	
    ReDim dat16b%(NumDat& - 1)
    INDEX = gIndexArray(gIndexLine, 0) 'G
	再計算時の基線補正処理
	再計算時で５分間のデータがあることを想定している。

For j = 0 To NumDat& - 1
        dat16b(j) = dat16(INDEX)
        INDEX = INDEX + NumCh&
    Next j
	極大極小検出ルーチンに主桁ひずみデータを渡すために、２次元配列になっているひずみデータを１次元配列に代入する。

If indexJ < 0 Then clsMyWaveData.listindex = gIndexArray(gIndexLine, 0)	基線補正する場所を示すindexJが０以下の場合は自動計算する。

st = GetPeekmaxmin(0, NumDat, dat16b(0), 0, clsMyWaveData.max0, dat16b(0) - clsMyWaveData.max0, 10, (clsMyWaveData.max1 - clsMyWaveData.max2) / 3, Indexd&(0), vald&(0), count&)
        k = 0
        For l = 0 To count - 1 Step 2
            If k < (Indexd&(l + 2) - Indexd&(l)) Then
                k = (Indexd&(l + 2) - Indexd&(l))
                idim = l
            End If
        Next l
        j = 0
	極大極小検出ルーチン

If count < 3 Then
            j = 0
        Else
	５分間でＰ－Ｐが３以下だったら、車両が居ないと判断して、データの先頭を使う。

j = (Indexd&(idim) + Indexd&(idim + 2)) / 2 - numheikin% / 2
        End If
        If j < 0 Then j = 0
        If j + numheikin% > NumDat Then j = NumDat - numheikin - 1
    Else
        j = indexJ
    End If
 	
    wimb_cal_kisen = j
 	
    l = gIndexLine
    si = 0
	ピークの中点を求める。

INDEX = j * NumCh& + gIndexArray(l, 0) 'G
    For k = 1 To numheikin%
        si = si + dat16(INDEX)
        INDEX = INDEX + NumCh&
    Next k
    si = si / numheikin%
    clsMyWaveData.listindex = gIndexArray(l, 0) 'G
    clsMyWaveData.max0 = si
	平均値の計算。

st = wim_set_indexline(l) wim_set_kisen 0, clsMyWaveData.max0	求まった基線補正値をセットする。

End Function

＜関連するコード＞	＜解　説＞
//----------	リアルタイム処理時の基線補正処理
// 基線補正（桁用）	
// FIFOの全個数で基線補正をする。	
// 車両が見つからなかったタイミングでこの関数を呼ぶ。	
// Ｐ－Ｐが不感体の１／４以下じゃないと実行しない	
//----------	
NOMANGLE int CCONV wim_zerohoseiG(int Index)	
{	
int i, j, flag;	
short *p, min,max;	
double sum, x[4];	
flag = ret_ok;	
for(j = 0; j < gNumline; j++){	
sum = 0;	
min = 32767;	
max = -32767;	
for(i = 0, p=gline_dat[j].sp[Index]; i<gNFifO; i++, p++){	FIFOデータの最大値、最少値を求める。
sum += (double)*p;	
if (min > *p) {	
min = *p;	
}	
if (max < *p) {	
max = *p;	
}	
}	
x[j] = (double)max - min;	
if (x[j] < (double)gline_dat[j].fukantai[Index]/4.0){	Ｐ－Ｐが不感帯の１／４以下だったらリターン
}else{	
flag = ret_err;	
}	
#if defined(TRACE)	
sprintf(gmsg,"ZerohoseiG100 : %d ch, p-p :%d¥n"	
, gline_dat[j].ch[Index], x[j]);	
g_print(gmsg);	
#endif	
x[j] = sum /(double)gNFifO;	平均値を不感帯とする。
}	
if(flag == ret_ok){	
for(j = 0; j < gNumline; j++){	
gline_dat[j].tzero_index = 0;	
gline_dat[j].zerohosei[Index] = (short)x[j];	求まった基線補正値をセットする。
#if defined(TRACE)	
sprintf(gmsg,"ZerohoseiG100 : %d ch=%d set¥n"	
, gline_dat[j].ch[Index], gline_dat[j].zerohosei[Index]);	
g_print(gmsg);	
#endif	
}	
}	
return flag;	
}	

2.6 車両検出

（1）ロジックの解説

橋軸方向及び幅員方向の複数箇所で測定した RC 床版のひずみ応答値を計測して、その波形の大小から、走行車線の判別，走行車両の速度・位置，各車両の軸数及び軸間距離の計算を行う。

・軸検出

床板に設置したひずみ波形（図3ひずみ波形参照）のピークを軸通過と判断する。

・速度計算

入口と出口のひずみ計間の距離とこの距離を車両が通過するのにかかった時間（Ta）から計算する。

・軸間距離

車両速度と各ピーク間の時間（Ti）から軸間距離を計算する。

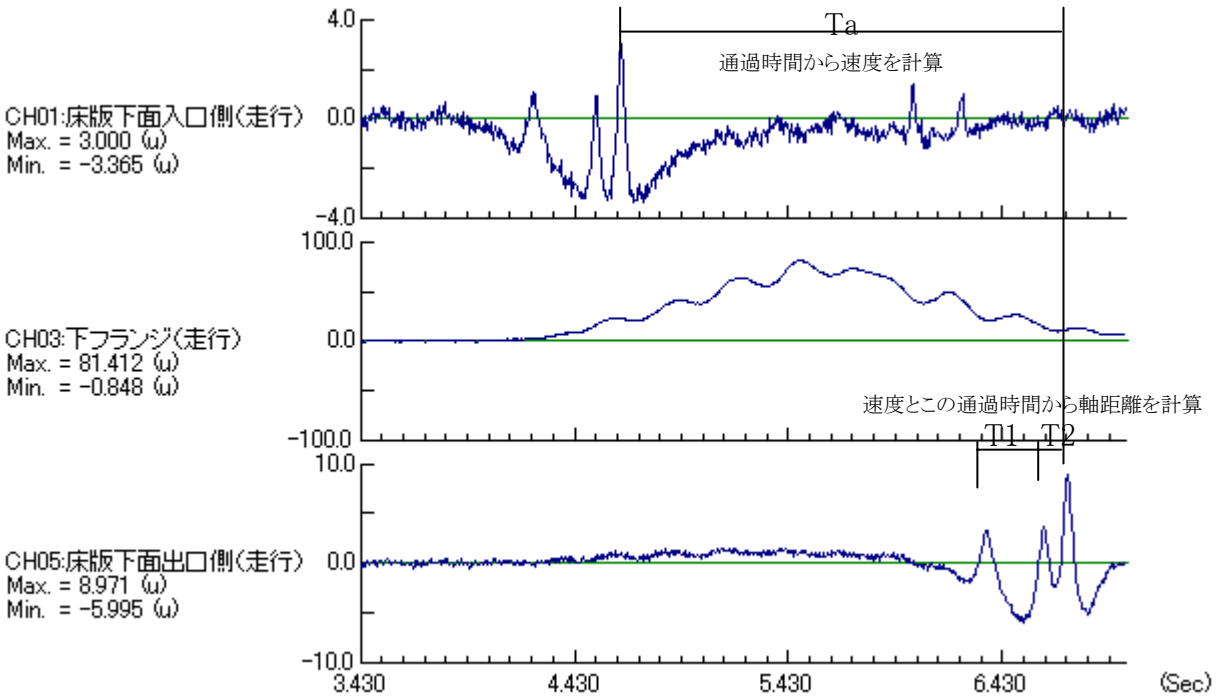


図3　ひずみ波形

(2)主要コードの解説

＜関連するコード＞	＜解　説＞
//----------	車両検出処理
// 縦ひずみから車軸を検出	
// (通常の区間は0からバッファ個数)	
//	
// wim_set_fifoでデータを入れておく	
// ひずみデータからピークを取り出す	
// wim_serch_sharyo_gomitori ごみとり	
// wim_serch_sharyo_sub　　ごみとり後のピークデータから車両検出	
//	
// Flag＝1：バッファに溜まってなくて実行	
// Flag＝2：桁ひずみ使う	
//----------	
NOMANGLE int CCONV wim_serch_sharyo(int iskukan, int inum, int Flag)	
{	
int st,offset,i;	
#if defined(TRACE)	
int j, k, l;	
#endif	
offset = gnum_fifo - gIndexFif0;	
#if defined(TRACE2)	デバック用コード
sprintf(gmsg, "%d line wim_serch：offset,fin：%d,%d (is,num：%d,%d)"	
,gIndexline	
,gnum_fifo	
,gpline->fin_index	
,iskukan, inum	
);	
g_message(gmsg);	
#endif	
#if defined(TRACE)	
g_print(gmsg); g_print("¥n");	
#endif	
st = 0;	
Indexd[i_g] = 0;	
Indexd[i_in] = 0;	
Indexd[i_out] = 0;	
gline_dat[gIndexline].tzero_index=0;	
if ((Flag & 1)=0)	
if ((fifo_full=0) || gone_sample＜gpline->fin_index){//次のデータが来ても大丈夫	前回計算した車両がＦＩＦＯデータに残っている場合はリターン
st = WIM_ERR_CAL_FIN_INDEX;	
goto errt;	
}	
iskukan += gpline->fin_index;	引数チェック
inum -= gpline->fin_index;	
if (iskukan + inum ＞ gNFif0){	
_itoa(WIM_ERR_OVER_FIFO,gmsg, 10);g_error(gmsg);	
return WIM_ERR_OVER_FIFO;	
}	
//桁ピーク検出----------	桁ひずみから極大極小法による軸検出
if (Flag & 2){	
#if defined(TRACE)	
g_print("g¥n");	
#endif	
GetPeekmaxmin(offset,inum, gpline->sp[i_g], iskukan, gpline->zerohosei[i_g], 0, MaxPeek	
,gpline->fukantai[i_g], Indexjiku[i_g], Valjiku[i_g], &Indexd[i_g]);	
if(Indexd[i_g]＜1){	
st=WIM_ERR_DAT_KETA;	
goto errt;	
}	
}	
//inピーク検出----------	入り側ひずみから極大極小法によるピーク検出

#if defined(TRACE)
g_print("in¥n");
#endif
GetPeekmaxmin(offset,inum, gpline->sp[i_in], iskukan, gpline->zerohosei[i_in], 0, MaxPeek
, gpline->fukantai[i_in], Indexjiku[i_in], Valjiku[i_in], &Indexd[i_in]);
if(Indexd[i_in] < 3){
st=WIM_ERR_DAT_IN;
goto errt;
}	

//out ピーク検出----------	出口側ひずみから極大極小法による軸検出

//桁ピーク以降のデータを使う
//01/5/15:inと同じ位置から
#if defined(TRACE)
g_print("out¥n");
#endif
GetPeekmaxmin(offset,inum-Indexjiku[i_in][0], gpline->sp[i_out], Indexjiku[i_in][0] ,
gpline->zerohosei[i_out], 0, MaxPeek
, gpline->fukantai[i_out],Indexjiku[i_out], Valjiku[i_out], &Indexd[i_out]);
if(Indexd[i_out] < 2){
st=WIM_ERR_DAT_OUT;
goto errt;
}
if(Indexd[i_in]/2*2 == Indexd[i_in]){
Indexd[i_in] = Indexd[i_in]-1;
}
if(Indexd[i_out]/2*2 == Indexd[i_out]){
Indexd[i_out] = Indexd[i_out]-1;
}	

#if defined(TRACE)
sprintf(gmsg, "count1 : %d:%d¥n", Indexd[i_in] ,Indexd[i_out]);
g_print(gmsg);
for (i=0; i<Indexd[i_in]; i+=2){
j = Indexjiku[i_in][i+2] - Indexjiku[i_in][i];
k = Indexjiku[i_out][i+2] - Indexjiku[i_out][i];
l = abs(j-k);
if (l >= jikugosa){
sprintf(gmsg, "%d¥t%d : %d >%d no match¥n"
,j
,k
,l
,jikugosa
);
}else{
sprintf(gmsg, "%d¥t%d : %d =< %d match¥n"
,j
,k
,l
,jikugosa
);
}
g_print(gmsg);
}
#endif
	デバッグ用コード

wim_serch_sharyo_gomitori();
#if defined(TRACE)
sprintf(gmsg, "count2 : %d:%d¥n", Indexd[i_in] ,Indexd[i_out]);
g_print(gmsg);
#endif
	ノイズ処理

st = wim_serch_sharyo_sub(1);
if ((Flag & 1)==0){
struct ssharyo_dat *p;
p = &gsharyo_dat[0];
offset = 20000;
for(i=0; i<gNumsharyo; i++){
if(offset>p->tin_index)offset = p->tin_index;
	車両判定

```
      p++;
    }
    if(offset>500){//次のデータを待つ
      st = WIM_ERR_CAL_FIN_INDEX;
      goto errt;
    }
  }
//軸が正常に認識されていない場合がある
//  gpline->fin_index = Indexjiku[i_out][Indexd[i_out] - 1];
#if defined(TRACE)
  sprintf(gmsg,"LINE : %d : gNumsharyo : %d¥n",gIndexline+1, gNumsharyo); g_print(gmsg);
#endif
  return gNumsharyo;

errt:
//一度完了したインデックスを戻したらだめ
//  gpline->fin_index = 0;
#if defined(TRACE)
  sprintf(gmsg,"wim_serch st : %d¥n",st);g_print(gmsg);
#endif
  return st;

}

void dim_remove(int inout,int index,int n)
{
  int i;

#if defined(TRACE)
  sprintf(gmsg,"dim_remove      %d:[%d]      <=      [%d]:%d      kill¥n",inout,      index,
index+n,Indexjiku[inout][index] );g_print(gmsg);
#endif

  Indexd[inout] = Indexd[inout] - n;
  for (i = index; i<Indexd[inout]; i++){
    Valjiku[inout][i] = Valjiku[inout][i + n];
    Indexjiku[inout][i] = Indexjiku[inout][i + n];
  }
}

//---------------------------------------------------------------------------
// 車輪数判定前処理
//in,out 両方の計算結果を照らし合わせて余計なピーク（車軸）を削除
//---------------------------------------------------------------------------
NOMANGLE int CCONV wim_serch_sharyo_gomitori(void)
{
//  int ifin,ifout,iin,iout,j;
  int i,i2;
  int ivin,ivout;
  int iind[30],ioutd[30],isad[30];
  int nmatch, flag;
  double lvel, x,l1;

    lvel = (gpline->l[0] - gpline->l[1] - gpline->l[2])/gdt;
top1:
  isad[0] = Indexjiku[i_out][0] - Indexjiku[i_in][0];
  iind[0] = Indexjiku[i_in][2] - Indexjiku[i_in][0];
  gvel = lvel / isad[0];//速度
  l1 = gvel * gdt * iind[0];//１軸目の長さ
#if defined(TRACE)
    sprintf(gmsg, "vel = %.1f¥n", gvel);g_print(gmsg);
#endif
  nmatch = 0;
  if (Indexd[i_in]<3 || Indexd[i_out]<3) goto exitt;
#if defined(TRACE)
  g_print("top¥n");
```

連行を上手く処理するために、先頭車両がＦＩＦＯの２．５秒（５００個）以内に入るようにする。この処理をすることで２台連行が１０秒のＦＩＦＯになるべく収まるようにしている。

極大、極小値配列の index から n 個削除する関数

車輪数判定前処理
ノイズ除去処理

```
#endif
  flag=1;
  for(i=0,i2=0; i2+2<=Indexd[i_in] || i2+2<=Indexd[i_out]; i++, i2+=2){

    iind[i] = Indexjiku[i_in][i2+2] - Indexjiku[i_in][i2];
    ioutd[i] = Indexjiku[i_out][i2+2] - Indexjiku[i_out][i2];
    isad[i] = Indexjiku[i_out][i2] - Indexjiku[i_in][i2];
    iind[i+1] = 0;
    ioutd[i+1] = 0;
    isad[i+1] = 0;
    if ((abs(iind[i] -ioutd[i]) <jikugosa) && flag) nmatch++;
    else                  flag=0;
#if defined(TRACE)
    sprintf(gmsg, "%4d : %4d¥n"
     , iind[i]
     , ioutd[i]
     );g_print(gmsg);
#endif
  }

  if ((nmatch == 0)                                          インｍ側とアウト側の軸間距離が１つも合ってない時
  && (5<=Indexd[i_in] && 9<=Indexd[i_out])
  && (abs(iind[0] - ioutd[1])<jikugosa)                      インｍ側の１つ目とアウト側の２つ目が一致
  && (abs(iind[1] - ioutd[2]-ioutd[3])<jikugosa)             インｍ側の１つ目とアウト側の２つ目＋３つ目が一致
  ){
#if defined(TRACE)
    g_print("典型的なパターン消去１¥n");
#endif
    dim_remove(i_out, 6, 2);
    dim_remove(i_out, 0, 2);
    goto top1;
  }

  if ((nmatch < 2)
  && (9<=Indexd[i_in] && 5<=Indexd[i_out])
  && (abs(iind[1]       - ioutd[0])<jikugosa)
  && (abs(iind[2]+iind[3] - ioutd[1])<jikugosa)
  && (abs(iind[4]       - ioutd[2])<jikugosa)
  ){
#if defined(TRACE)
    g_print("典型的なパターン消去２¥n");
#endif
    dim_remove(i_in, 6, 2);
    dim_remove(i_in, 0, 2);
    goto top1;
  }
  if (
     (abs(iind[0] - ioutd[0])             < jikugosa)
  && (abs(iind[1] - ioutd[1])             < jikugosa)
  && (abs(iind[2] - ioutd[2])             > jikugosa)
  && (abs(iind[3] - ioutd[3])             > jikugosa)
  && (abs(iind[2]+iind[3] -ioutd[2]-ioutd[3]) < jikugosa)
  && (abs(iind[4] - ioutd[4])<jikugosa)
  ){
#if defined(TRACE)
    g_print("典型的なパターン消去３¥n");
#endif
    dim_remove(i_in, 6, 2);
    dim_remove(i_out, 6, 2);
    goto top1;
  }

  if (Indexd[i_in] >=13 && Indexd[i_out] >=13
  && (abs(iind[0] - ioutd[0])             < jikugosa)
  && (abs(iind[1] - ioutd[1])             < jikugosa)
  && (abs(iind[2]+iind[3] -ioutd[2]) < jikugosa)
```

```
  && (abs(iind[4] - ioutd[3])<jikugosa)
  && (abs(iind[5] - ioutd[4])<jikugosa)
  ){
#if defined(TRACE)
    g_print("典型的なパターン消去4¥n");
#endif
    dim_remove(i_in, 6, 2);
    goto top1;
  }

  if (Indexd[i_in] >=13 && Indexd[i_out] >=13
  && (abs(iind[0] - ioutd[0])              < jikugosa)
  && (abs(iind[1] - ioutd[1])              < jikugosa)
  && (abs(iind[2]-ioutd[2] -ioutd[3]) < jikugosa)
  && (abs(iind[3] - ioutd[4])<jikugosa)
  && (abs(iind[4] - ioutd[5])<jikugosa)
  ){
#if defined(TRACE)
    g_print("典型的なパターン消去5¥n");
#endif
    dim_remove(i_out, 6, 2);
    goto top1;
  }
  if (Indexd[i_in] >=13 && Indexd[i_out] >=13
  && (abs(iind[0] - ioutd[0])              < jikugosa)
  && (abs(iind[1] - ioutd[1])              < jikugosa)
  && (abs(iind[2] - ioutd[2]) > jikugosa)
  && (abs(iind[3] - ioutd[3]) > jikugosa)
  && (abs(iind[2]+iind[3] - ioutd[2]-ioutd[3]) < jikugosa)
  && (abs(iind[4] - ioutd[4])<jikugosa)
  && (abs(iind[5] - ioutd[5])<jikugosa)
  ){
#if defined(TRACE)
    g_print("典型的なパターン消去6¥n");
#endif
    dim_remove(i_in, 6, 2);
    dim_remove(i_out, 6, 2);
    goto top1;
  }

  if (
     (Indexd[i_in] >=5 && Indexd[i_out] >=11)
  && (abs(iind[0] - ioutd[0])                 > jikugosa)
  && (abs(iind[1] - ioutd[1])                 > jikugosa)
  && (abs(iind[2] - ioutd[2])                 > jikugosa)
  && (abs(iind[0] - ioutd[1]-ioutd[2]-ioutd[3])<jikugosa)
  && (abs(iind[1] - ioutd[4]) < jikugosa)
  )
  {
#if defined(TRACE)
    g_print("典型的なパターン消去7¥n");
#endif
    dim_remove(i_out, 6, 2);
    dim_remove(i_out, 4, 2);
    dim_remove(i_out, 0, 2);
    goto top1;
  }
  if (
     (Indexd[i_in] >=5 && Indexd[i_out] >=5)
  && (abs(iind[0] - ioutd[0])                 > jikugosa)
  && (abs(iind[1] - ioutd[1])                 > jikugosa)
  && (abs(iind[2] - ioutd[2])                 < jikugosa)
  && (abs(iind[0]+iind[1]-ioutd[0]-ioutd[1])<jikugosa)
  )
  {
#if defined(TRACE)
```

```
    g_print("典型的なパターン消去 8¥n");
#endif
    dim_remove(i_in, 2, 2);
    dim_remove(i_out, 2, 2);
    goto top1;
  }
  if (
     (Indexd[i_in] >=7 && Indexd[i_out] >=7)
  && (abs(iind[0] - ioutd[0])              > 4)
  && (abs(iind[1] - ioutd[1])              > 4)
  && (abs(iind[2] - ioutd[2])              > 4)
  && (abs(iind[1] - ioutd[0])              < 2)
  && (abs(iind[2] - ioutd[1])              < 2)
  && (abs(iind[3] - ioutd[2])              < 2)
  )
  {
#if defined(TRACE)
    g_print("典型的なパターン消去 9¥n");
#endif
    dim_remove(i_in, 0, 2);
    goto top1;
  }
  if (
     (Indexd[i_in] >=7 && Indexd[i_out] >=7)
  && (abs(iind[0] - ioutd[0])              > 4)
  && (abs(iind[1] - ioutd[1])              > 4)
  && (abs(iind[2] - ioutd[2])              > 4)
  && (abs(iind[0] - ioutd[1])              < 2)
  && (abs(iind[1] - ioutd[2])              < 2)
  && (abs(iind[2] - ioutd[3])              < 2)
  )
  {
#if defined(TRACE)
    g_print("典型的なパターン消去 10¥n");
#endif
    dim_remove(i_out, 0, 2);
    goto top1;
  }
  if (
     (Indexd[i_in] >=11 && Indexd[i_out] >=7)
  && (abs(iind[0] - ioutd[0])                > 2)
  && (abs(iind[1] - ioutd[1])                > 4)
  && (abs(iind[2] - ioutd[2])                > 4)
  && (abs(iind[3] - ioutd[3])                > 4)
  && (abs(iind[4] - ioutd[4])                > 4)
  && (abs(iind[5] - ioutd[5])                > 4)
  && (abs(iind[1]+iind[2] - ioutd[0])        < 2)
  && (abs(iind[3]+iind[4] - ioutd[1])        < 2)
  && (abs(iind[5] - ioutd[2])                < 2)
  )
  {
#if defined(TRACE)
    g_print("典型的なパターン消去 11¥n");
#endif
    dim_remove(i_in, 4, 2);
    dim_remove(i_in, 0, 2);
    goto top1;
  }
  if (
     (Indexd[i_in] >=8 && Indexd[i_out] >=6)
  && (abs(iind[0] - ioutd[0])                > 2)
  && (abs(iind[1] - ioutd[1])                > 2)
  && (abs(iind[2] - ioutd[2])                > 2)
  && (abs(iind[3] - ioutd[3])                > 2)
  && (abs(iind[1] - ioutd[0])               < 2)
  && (abs(iind[2]+iind[3] - ioutd[1]-ioutd[2])< 2)
```

```
  && (abs(iind[4] - ioutd[3])              < 2)
  )
  {
#if defined(TRACE)
    g_print("典型的なパターン消去 12¥n");
#endif
    dim_remove(i_in, 6, 2);
    dim_remove(i_in, 0, 2);
    dim_remove(i_out, 4, 2);
    goto top1;
  }
  if (
     (Indexd[i_in] >=5 && Indexd[i_out] >=5)
  && (abs(iind[0] - ioutd[0])               > 2)
  && (abs(iind[1] - ioutd[1])               > 2)
  && (abs(iind[2] - ioutd[2])               > 2)
  && (abs(iind[3] - ioutd[3])               > 2)
  && (abs(iind[0] - ioutd[1])               < 2)
  && (abs(iind[1] - ioutd[2])               < 2)
  )
  {
#if defined(TRACE)
    g_print("典型的なパターン消去 13¥n");
#endif
    dim_remove(i_out, 0, 2);
    goto top1;
  }

  if(Indexjiku[i_in][0] == Indexjiku[i_out][0]){
    ivin = Valjiku[i_in][0]-Valjiku[i_in][1];
    ivout = Valjiku[i_out][0]-Valjiku[i_out][1];

#if defined(TRACE)
    sprintf(gmsg, "同時削除? %d : %d, %d : %d¥n"
      , ivin, gpline->fukantai[i_in]
      , ivout, gpline->fukantai[i_out]
      );g_print(gmsg);
#endif
  if (
     (Indexd[i_in] >=5 && Indexd[i_out] >=5)
  && (abs(iind[0] - ioutd[0])               > 3)
  && (abs(iind[1] - ioutd[1])               < 3)
  && (abs(iind[2] - ioutd[2])               < 3)
  && (abs(iind[3] - ioutd[3])               < 3)
  )
  {
#if defined(TRACE)
    g_print("同時削除 inout0¥n");
#endif
    dim_remove(i_in, 0, 2);
    dim_remove(i_out, 0, 2);
    goto top1;
  }
  if (
     (Indexd[i_in] >=7 && Indexd[i_out] >=7)
  && (abs(iind[0] - ioutd[0])               > 3)
  && (abs(iind[2] - ioutd[2])               > 3)
  && (abs(iind[3] - ioutd[3])               > 3)
  && (abs(iind[1] - ioutd[1])               < 3)
  && (abs(iind[2]+iind[3] - ioutd[2]-ioutd[3]) < 3)
  && (abs(iind[4] - ioutd[4])               < 3)
  )
  {
#if defined(TRACE)
    g_print("同時削除 inout01¥n");
#endif
```

```
    dim_remove(i_in, 6, 2);
    dim_remove(i_out, 6, 2);
    dim_remove(i_in, 0, 2);
    dim_remove(i_out, 0, 2);
    goto top1;
  }

    //同時の場合はＯＵＴを消したくなるが、
    //in側がおかしい時もある
    x = (double)ivout/ivin;
    if(x > 3.0){
#if defined(TRACE)
      sprintf(gmsg, "同時削除 inout¥n");g_print(gmsg);
#endif
      dim_remove(i_in,0,2);
      dim_remove(i_out,0,2);
      goto top1;
    }
#if defined(TRACE)
    sprintf(gmsg, "同時削除 out¥n");g_print(gmsg);
#endif
    dim_remove(i_out,0,2);
    goto top1;

  }
  //速度が早い場合は削除
  if(gvel > 35.0){
#if defined(TRACE)
    sprintf(gmsg, "out 早い out 削除¥n");g_print(gmsg);
#endif
    dim_remove(i_out,0,2);
    goto top1;
  }
  //速度が遅い場合は削除
  if(0.0 < gvel  && gvel < 1.3){
#if defined(TRACE)
    sprintf(gmsg, "時速１０ｋｍ以下 in 削除¥n");g_print(gmsg);
#endif
    dim_remove(i_in,0,2);
    goto top1;
  }

  for(i=0; iind[i+2] > 0; i++){
#if defined(TRACE)
    sprintf(gmsg, "ina : %d %d %d¥nouta: %d %d %d¥n"
     ,iind[i],iind[i+1],iind[i+2]
     ,ioutd[i],ioutd[i+1],ioutd[i+2]
     );g_print(gmsg);
#endif
    if (//１軸目を削除
      (abs(iind[i] - ioutd[i])>=jikugosa) &&
      (abs(iind[i+1] - ioutd[i+1])<jikugosa) &&
      (abs(iind[i+2] - ioutd[i+2])<jikugosa)
     ){
#if defined(TRACE)
       g_print("1ng 2&3ok 1del¥n");
#endif
       dim_remove(i_in, i*2, 2);
       dim_remove(i_out, i*2, 2);
       goto top1;
    }
    if (
      (abs(iind[i]  - ioutd[i]) >= jikugosa) &&
      (abs(iind[i+1]  - ioutd[i+1]) >= jikugosa) &&
      (abs(iind[i+1] - ioutd[i]) < jikugosa) &&
      (abs(iind[i+2] - ioutd[i+1]) < jikugosa)
```

```
    ){
#if defined(TRACE)
       g_print("in2 = out1 del¥n");
#endif
       dim_remove(i_in, i*2, 2);
       goto top1;
   }
   if (
     (abs(iind[i]   - ioutd[i]) >= jikugosa) &&
     (abs(iind[i+1]   - ioutd[i+1]) >= jikugosa) &&
     (abs(iind[i]   - ioutd[i+1]) < jikugosa) &&
     (abs(iind[i+1] - ioutd[i+2]) < jikugosa)
    ){
#if defined(TRACE)
       g_print("in1 = out2 del¥n");
#endif
       dim_remove(i_out, i*2, 2);
       goto top1;
   }

   if (
     (abs(iind[i]+iind[i+1] - ioutd[i])  < jikugosa) &&
     (abs(iind[i+2]         - ioutd[i+1]) < jikugosa)
    ){
#if defined(TRACE)
       g_print("in1+2 = out1 del¥n");
#endif
       dim_remove(i_in, i*2+2, 2);
       goto top1;
   }
   if (
     (abs(iind[i]-ioutd[i] - ioutd[i+1]) < jikugosa) &&
     (abs(iind[i+1]        - ioutd[i+2]) < jikugosa)
    ){
#if defined(TRACE)
       g_print("out1+2 = in1 del¥n");
#endif
       dim_remove(i_out, i*2+2, 2);
       goto top1;
   }
   if (
     (abs(iind[i+1]+iind[i+2] - ioutd[i]) < jikugosa) &&
     (abs(iind[i+3]           - ioutd[i+1]) < jikugosa)
    ){
#if defined(TRACE)
       g_print("in2+3 = out1 del¥n");
#endif
       dim_remove(i_in, i*2+4, 2);
       dim_remove(i_in, i*2, 2);
       goto top1;
   }
   if (
     (abs(ioutd[i+1]+iind[i+2] - iind[i]) < jikugosa) &&
     (abs(ioutd[i+3]           - iind[i+1]) < jikugosa)
    ){
#if defined(TRACE)
       g_print("out2+3 = in1 del¥n");
#endif
       dim_remove(i_out, i*2+4, 2);
       dim_remove(i_out, i*2, 2);
       goto top1;
   }
  }

  i=0;
  if(l1 > 10.5){
```

```
#if defined(TRACE)
    sprintf(gmsg, "in l1>10.0¥n");g_print(gmsg);
#endif
    dim_remove(i_in,0,2);
    i=1;
  }
  if(gvel * gdt * ioutd[0] > 10.5){
#if defined(TRACE)
    sprintf(gmsg, "out l1>10.5¥n");g_print(gmsg);
#endif
    dim_remove(i_out,0,2);
    i=1;
  }
  if(i) goto top1;

  if(Indexd[i_in] >= 4 && Indexd[i_out] >= 6 ){
    if( (abs(iind[0]-ioutd[0]) < jikugosa)
    && (abs(iind[1]-ioutd[1]-ioutd[2]) < jikugosa)
    )
    {
      sprintf(gmsg, "xx:out¥n");g_print(gmsg);
      dim_remove(i_out,4,2);
      goto top1;
    }
  }
  if(Indexd[i_in] >= 6 && Indexd[i_out] >= 4 ){
    if( (abs(iind[0]-ioutd[0]) < jikugosa)
    && (abs(iind[1]+iind[2]-ioutd[1]) < jikugosa)
    )
    {
      sprintf(gmsg, "xx:in¥n");g_print(gmsg);
      dim_remove(i_in,4,2);
      goto top1;
    }
  }




  if (nmatch == 0){
#if defined(TRACE)
    sprintf(gmsg, "2軸?¥n");g_print(gmsg);
#endif
    if((abs(iind[0]-ioutd[0]) < jikugosa) && (gvel > 2.7) && (gvel < 38.9) && l1 > 2.0){
#if defined(TRACE)
      sprintf(gmsg, "2軸ok¥n");g_print(gmsg);
#endif
      nmatch++;
    }else{//マッチしていない場合
#if defined(TRACE)
      sprintf(gmsg, "2軸x¥n");g_print(gmsg);
#endif
      if(Indexd[i_in] >= 7 && Indexd[i_out] >= 5){
        if( abs(iind[1]+iind[2]-ioutd[1]) < jikugosa){
#if defined(TRACE)
          sprintf(gmsg, "xx:0¥n");g_print(gmsg);
#endif
          dim_remove(i_in,4,2);
          dim_remove(i_in,0,2);
          dim_remove(i_out,0,2);
          goto top1;
        }
      }
      if(Indexd[i_out] >= 4 && Indexd[i_in] >= 4){
        if( abs(iind[0]+iind[1]-ioutd[0]-ioutd[1]) < jikugosa){
#if defined(TRACE)
```

```
        sprintf(gmsg, "xx:1¥n");g_print(gmsg);
#endif
        dim_remove(i_in,2,2);
        dim_remove(i_out,2,2);
        goto top1;
       }
     }
     if(Indexd[i_out] >= 3 && Indexd[i_in] >= 5){
       if( abs(ioutd[0]-iind[0]-iind[1]) < jikugosa){
#if defined(TRACE)
        sprintf(gmsg, "xx:3¥n");g_print(gmsg);
#endif
        dim_remove(i_in,2,2);
        goto top1;
       }
     }

     if(Indexd[i_out] >= 5 && Indexd[i_in] >= 3){
       if( abs(iind[0]-ioutd[0]-ioutd[1]) < jikugosa){
#if defined(TRACE)
        sprintf(gmsg, "xx:4¥n");g_print(gmsg);
#endif
        dim_remove(i_out,2,2);
        goto top1;
       }
     }
     if(Indexd[i_in] >= 5 && Indexd[i_out] >= 3){

       i2 = Valjiku[i_in][0] - Valjiku[i_in][1];
       if( (abs(iind[1]-ioutd[0]) < jikugosa)
       &&  (i2 < gpline->fukantai[i_out] * 3)

        ){
#if defined(TRACE)
        sprintf(gmsg, "xx:5¥n");g_print(gmsg);
#endif
        dim_remove(i_in,0,2);
        goto top1;
       }
     }
     if(Indexd[i_in] >= 3 && Indexd[i_out] >= 5){
       if( abs(iind[0]-ioutd[1]) < jikugosa){
#if defined(TRACE)
        sprintf(gmsg, "xx:6¥n");g_print(gmsg);
#endif
        dim_remove(i_out,0,2);
        goto top1;
       }
     }
   }
  }

  if(nmatch == 0){//マッチしなかった時は無条件に先頭１軸を削除
#if defined(TRACE)
    sprintf(gmsg, "マッチしなかった時は無条件に先頭１軸を削除¥n");g_print(gmsg);
#endif
    dim_remove(i_in,0,2);
    dim_remove(i_out,0,2);
    goto top1;
  }

//後ろを削除：inとoutの間隔が等しくない軸を削除
//  sprintf(gmsg, "ein,out処理¥n");g_print(gmsg);
  for(i=0; iind[i]>0; i++){
    if(abs(iind[i]-ioutd[i]) >= jikugosa){
      if(abs(iind[i]+iind[i+1]-ioutd[i]) < jikugosa){
```

```
#if defined(TRACE)
        sprintf(gmsg, "最後のあがき in を削除¥n");g_print(gmsg);
#endif
        dim_remove(i_in,i*2+2,2);
        goto top1;
      }
      if(abs(iind[i]-ioutd[i]-ioutd[i+1]) < jikugosa){
#if defined(TRACE)
        sprintf(gmsg, "最後のあがき out を削除¥n");g_print(gmsg);
#endif
        dim_remove(i_out,i*2+2,2);
        goto top1;
      }
      Indexd[i_in] = (i+1)*2;
      Indexd[i_out] = (i+1)*2;
      break;
    }
  }
exitt:

  if(Indexd[i_in] >=11){
    if(
       (x * (Indexjiku[i_in][2] - Indexjiku[i_in][0]) > 2.0)
    &&  (x * (Indexjiku[i_in][4] - Indexjiku[i_in][2]) < 1.5)
    &&  (x * (Indexjiku[i_in][6] - Indexjiku[i_in][4]) > 4.0)
    &&  (x * (Indexjiku[i_in][8] - Indexjiku[i_in][6]) > 4.0)
    &&  (x * (Indexjiku[i_in][10] - Indexjiku[i_in][8]) < 1.5)
    ){
      dim_remove(i_in, 6, 2);
      dim_remove(i_out, 6, 2);
      goto top1;
    }
  }
  if (Indexd[i_in] > Indexd[i_out] )Indexd[i_in] = Indexd[i_out];
  if (Indexd[i_in] < Indexd[i_out] )Indexd[i_out] = Indexd[i_in];


#if defined(TRACE)
  sprintf(gmsg,"nPeek = %d,%d¥n", Indexd[i_in], Indexd[i_out]);g_print(gmsg);
#endif

  return 0;
}

//---------------------------------------------------------------------
// 速度の計算
// FlagJikuSet : 1で車両の軸間距離をセットする
//---------------------------------------------------------------------
NOMANGLE int CCONV wim_serch_sharyo_sub(int FlagJikuSet)
{
  int i1,i2;
  int st,max;
    int l, njiku, i, n, ild, idai;
  double x, xl;
    double sx, xx;
  double xm1, xm2;

  static struct ssharyo_dat *p;

//速度を計算
  //  l = Indexjiku[i_g][0];//桁ひずみのピーク
    xx = Indexjiku[i_out][0] - Indexjiku[i_in][0];//in_out 間の通過に掛かった時間
  xx = xx * gdt;
    gvel = (gpline->l[0] - gpline->l[1] - gpline->l[2]) / xx;

#if defined(TRACE)
```

極大、極小値配列データを用いて車両判定処理を行い、車両と判断できた軸間距離を計算用車両データ構造体に登録する

```
  sprintf(gmsg,"v=l/t(km/h): %f = %f/%f*3.6¥n"
    , gvel*3.6
    , (gpline->l[0] - gpline->l[1] - gpline->l[2])
    , xx
    );
  g_print(gmsg);
  sprintf(gmsg,"tg1,tg2-tg1: %f,%f¥n"
    , gdt*(Indexjiku[i_in][0]  + gnum_fifo - gIndexFif0)-gpline->l[1]/gvel
    , gpline->l[0]/gvel
    );
  g_print(gmsg);
#endif
  if((gvel < 1.3) || (gvel  > 35.0)){//時速１０ｋ以下はエラー
    st = WIM_ERR_CAL_VEL;
    goto errt;
  }
    gDx = gvel * gdt;//１ドットあたりに進む距離
    if( !FlagJikuSet )  return ret_ok;

    if( Indexd[i_in] < 3 || Indexd[i_out] < 3 ){  //３個以上のピークがないとエラー
#if defined(TRACE)
    g_print("n<3");
#endif
    st = WIM_ERR_CAL_NUM_JIKU;
        goto errt;
    }
  if (Indexjiku[i_out][0] < Indexjiku[i_in][0]){//OUT側の方が早いとエラー
#if defined(TRACE)
    g_print("inout");
#endif
    st = WIM_ERR_CAL_NUM_JIKU;
        goto errt;
  }

  for (i = 0; i < MaxJiku; i++){
    gCar_L[i] = 0.0;
  }
    gNumberJiku = 0;
  njiku = Indexd[i_in]/2;//ピークの数から軸数を求める
  if (Indexd[i_in]/2*2 != Indexd[i_in]){
    njiku++;
  }
#if defined(TRACE)
  sprintf(gmsg,"njiku1 = %d(%d/2)¥n", njiku, Indexd[i_in]);g_print(gmsg);
#endif
//in側
//01 23 45 67
//+-,+-,+-,+-
//軸間距離の計算
    l = 2;
    for( i = 1; i< njiku; i++){
        xm1 = (Indexjiku[i_in][l] - Indexjiku[i_in][l-2]) * gDx;
        xm2 = (Indexjiku[i_out][l] - Indexjiku[i_out][l-2]) * gDx;
    x = fabs(xm1-xm2);
    if((1.0 < x)
    ){
#if defined(TRACE)
      sprintf(gmsg,
        "%.1f = (%d - %d) * %.1f¥n%.1f = (%d - %d) * %.1f¥n"
        ,xm1, Indexjiku[i_in][l] , Indexjiku[i_in][l-2] , gDx
        ,xm2, Indexjiku[i_out][l] , Indexjiku[i_out][l-2] , gDx
        );
      g_print(gmsg);
#endif
      if (i == 1){
        st=WIM_ERR_CAL_LEN_JIKU;
```

ここではＩＮ、ＯＵＴの軸間距離誤差が１m以上の時に軸間距離配列をセットする。

```
        goto errt;
      }
      njiku = i;
      break;
    }
        if( FlagJikuSet ){
            gCar_L[i - 1] = xm1;
#if defined(TRACE)
      sprintf(gmsg,"l(%d) : %.1f¥n",i-1, xm1);g_print(gmsg);
#endif
        }
        l = l + 2;
    }
#if defined(TRACE)
  sprintf(gmsg,"njiku2 = %d¥n", njiku);g_print(gmsg);
#endif

  idai = 0;
  ild = 0;
  while (true){//車両検出
#if defined(TRACE)
  sprintf(gmsg,"%d : idai = %d¥n", ild, idai);g_print(gmsg);
#endif
    n = SharyoHantei(njiku, &gCar_L[ild]);//車両の区切りを求める
    if (n < 2){//車両見つからない。。。
#if defined(TRACE)
    g_print("n<2");
#endif
      st = WIM_ERR_CAL_NUM_JIKU;
      break;
    }
    if((n-1)*2 >= Indexd[i_out]){//アウト側のデータと合わない
      st = WIM_ERR_CAL_LEN_JIKU;
      break;
    }
    if (Indexjiku[i_in][ild*2] >= Indexjiku[i_out][(ild + n-1)*2] ){
#if defined(TRACE)
      sprintf(gmsg,"ild,n %d:%d¥nNot t1 < t2 (it1,it2) :%d,%d¥n"
       ,ild, n
       ,Indexjiku[i_in][ild*2] ,Indexjiku[i_out][(ild + n-1)*2] );g_print(gmsg);
#endif
      st=WIM_ERR_CAL_LEN_JIKU;
      break;
    }
////////////////////////////////////////////////////////////////////////////////
//ラインデータに見つけた車両データを登録

#if defined(TRACE)
    sprintf(gmsg,"LINE:%d %d:p[%d]=%d 軸¥n"
      ,glndexline, idai, glndexsharyo, n);g_print(gmsg);
#endif
    gpline->Indexsharyo[idai] = glndexsharyo;
    p = &gsharyo_dat[glndexsharyo];
    p->Indexline = glndexline;
    p->v = gvel;
    p->n = n;
    p->t1 = Indexjiku[i_in][ild*2]  * gdt;          //1軸がINに乗った時刻
      p->t2 = Indexjiku[i_out][(ild + n-1)*2] * gdt; //n軸がoutに乗った時刻
    x = p->t1 / gdt;  p->t1_index=(int)x;
    x = p->t2 / gdt;  p->t2_index=(int)x;

    sx = 0;
    for(i=0; i<n; i++){
//      p->l[i] = x - gCar_L[i];//後輪からの距離から1軸からの距離にして代入
      p->l[i] = sx;//1軸からの距離
      sx += gCar_L[ild+i];
```

車両判定処理を実行

```
    }

#if defined(TRACE)
    sprintf(gmsg,"p->tin : %.3f,%.3f¥n", Indexjiku[i_in][ild*2] * gdt , gpline->l[1] / gvel);
    g_print(gmsg);
#endif
    x = Indexjiku[i_in][ild*2] * gdt - gpline->l[1] / gvel; //1軸がIN側の支点に乗った時刻
    x = x / gdt;
    p->tin_index = (int)x;
//    x = x +  (gpline->l[0]) / gvel / gdt;//1軸が通過
    x = x +  (gpline->l[0]+p->l[p->n-1]) / gvel / gdt;//n軸が通過 2002/11/07
    p->tout_index = (int)x;

#if defined(TRACE)
    sprintf(gmsg,"v2,njiku : %.1f,%d", gvel*3.6, n);
    g_print(gmsg);
#endif

//無視していい車両かの判断
    i1 = (p->tin_index + p->tout_index)/2;
    i2 = i1+100;
    if (gNFif0 <= i2){
      break;
    }
      max = -32767;
      for (i=i1; i<i2; i++){//桁が不感体１／４以下だったら以降の計算をしない
        if (max < gpline->sp[i_g][i]) max = gpline->sp[i_g][i];
      }
      max -= gpline->zerohosei[i_g];
      if (max > gpline->fukantai[0]/4){
        idai++;
        gIndexsharyo++;
        gNumsharyo++;
#if defined(TRACE)
        sprintf(gmsg,"%d軸見つけた¥n", n );g_print(gmsg);
#endif
      }else{
#if defined(TRACE)
        sprintf(gmsg,"無視¥n");g_print(gmsg);
#endif
      }
      ild += n;//n軸目から再検索
      njiku -= n;
     if (idai == 2) break;//２台以上の連行は無視
   }

//一台目を旧データに登録
//後輪からの距離を代入(軸距離：後軸からの距離,xl:最後輪からの距離)
   p = &gsharyo_dat[gpline->Indexsharyo[0]];
   l = 0;
   xl = p->l[p->n-1];
   for( i = 0; i<p->n; i++){
      gCar_L[i] = xl - p->l[i];
    }
   gNumberJiku = p->n;

#if defined(TRACE)
   g_print("¥n");
#endif
//  gpline->fin_index = Indexjiku[i_out][Indexd[i_out] - 1];
   gpline->nsharyo = idai;//idai台見つけた
   if (idai == 0) goto errt;
   return ret_ok;

errt:
#if defined(TRACE)
```

```
  sprintf(gmsg,"wim_cal_v : %d¥n",st);g_print(gmsg);
#endif
  return st;
}

//-----------------------------------------------------------------------
// 1台判別処理
// FlagJikuSet : 車両の軸間距離をセットする
// Indexd[3];   //ピーク数
// *Indexjiku[3];//ピークのインデックス
// *Valjiku[3];  //ピークインデックスの値
// gCar_L[i] = 0.0;
// gNumberJiku = 0;
//-----------------------------------------------------------------------
NOMANGLE int CCONV SharyoHantei(int n, double *ld)
{
  int st, flag;

#if defined(TRACE)
     g_print("3¥n");
#endif

  if (10.5 < ld[0]) {
    st = WIM_ERR_CAL_LEN_JIKU;
    goto errt;
  }

  for(st=0; st<n; st++)
    if(ld[st] > 16.0){
      n = st+1;
      break;
    }

//  goto cont1;
  st=0;
  flag = 0;
  if(
     (2.8 < ld[0]) && (ld[0]< 3.4)
  && (1.1 < ld[1]) && (ld[1]<1.6)
  && (6 < ld[2])
  && (1.1 < ld[3]) && (ld[3]<1.6) && (n >= 5)
  ){//5軸車判定
    if( (n >= 6) && (1.1 < ld[4]) && (ld[4]<1.75)){
#if defined(TRACE)
      g_print("66666¥n");
#endif
      n = 6;
      return n;
    }else{
#if defined(TRACE)
      g_print("55555¥n");
#endif
      n = 5;
      return n;
    }
  }

  if(
     (3.0 < ld[0]) && (ld[0]< 3.4)
  && (1.1 < ld[1]) && (ld[1]<1.6)
  && (1.6 < ld[2])
  ){//4軸以上あるけど3軸車とみなす
    if((n >= 6) && (ld[3] < 1.8) && (ld[4] < 1.8)){//甲子橋を通った
#if defined(TRACE)
      g_print("6666¥n");
#endif
```

1台判別処理
図4－2ＷＩＭＶｅｒ３．１の車両判別処理ロジック参照

```
      n = 6;
      return n;
    }else if((n >= 5) && (ld[3] < 1.8)){
#if defined(TRACE)
      g_print("5555¥n");
#endif
      n = 5;
      return n;

    }else if((ld[2] < 10.0)){
#if defined(TRACE)
      g_print("4444¥n");
#endif
      n = 4;
      return n;
    }else{
#if defined(TRACE)
      g_print("3333¥n");
#endif
      n = 3;
      return n;
    }
  }

  if(n >= 6){//6軸の場合
    if((ld[3] < 1.8) && (ld[4] < 1.8)){//甲子橋を通った
      n = 6;
    }else{
      if((ld[2] < 2.0) && (ld[3] < 2.0) && (ld[4] < 2.0)){
        n = 6;

      }else if((ld[1] < 1.5) && (ld[4] < 1.5)){
        n = 3; flag=1;
#if defined(TRACE)
        g_print("6->3¥n");
#endif
      }else if((ld[1] > 4) && (ld[2] < 1.6)){
        n = 4; flag=1;
#if defined(TRACE)
        g_print("6->4¥n");
#endif
      }else{
        n = 5; flag=1;
#if defined(TRACE)
        g_print("6->5¥n");
#endif
      }
    }
  }
  if(n == 5){//5軸の場合
    if(ld[2] > 10.0  && ld[3] > 1.5 ) {
      n = 3;flag=1;
#if defined(TRACE)
      g_print("5->3¥n");
#endif
    }else if(ld[3] > 6.0 ) {
      n = 4;flag=1;
#if defined(TRACE)
      g_print("5->4¥n");


#endif
    }
  }
  if(n == 4){//4軸の場合
    if(ld[2] > 2.1 ) {
```

```
      n = 3;flag=1;
#if defined(TRACE)
      g_print("4->3¥n");
#endif
    }
  }
  if(n==3){//3軸の場合
    if(ld[0] < 1.5 && ld[1] >10.5){
#if defined(TRACE)
      g_print("3->0 err¥n");
#endif
      st = WIM_ERR_CAL_LEN_JIKU;
      goto errt;
    }

    if(ld[1] > 12.0){
      if(ld[0] < 1.3){
#if defined(TRACE)
        g_print("3->0 err¥n");
#endif
        st = WIM_ERR_CAL_LEN_JIKU;
        goto errt;
      }else{
        n = 2;flag=1;
#if defined(TRACE)
        g_print("3->2¥n");
#endif
      }
    }
  }

  if(n==2){//2軸の場合
    if(ld[0] < 2.0){
#if defined(TRACE)
      g_print("err:l < 2.0¥n");
#endif
      st = WIM_ERR_CAL_LEN_JIKU;
      goto errt;
    }
  }
#if defined(TRACE)
  sprintf(gmsg,"SharyoHantei : %d¥n",n);g_print(gmsg);
#endif
  return n;

errt:
#if defined(TRACE)
  sprintf(gmsg,"SharyoHantei : %d¥n",st);g_print(gmsg);
#endif
  return 0;

}
```

2.7 車両重量計算

（1）ロジックの解説

各主桁の推定ひずみ（ε）と実測ひずみ（ε’）との誤差（ek）は前項までに示した式４で表される。

$$e_k = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \left(\frac{1}{EZ_{ki}} \cdot \frac{P_{ij} \cdot x'}{2} \right) - \varepsilon'_k(t) \right)^2 \quad \cdots式４$$

$$x = (t - t_i) \cdot V_i + l_{ij}$$

$$x' = x$$

（$0 \le x \le 1/2\ell$ のとき）

$$x' = l - x$$

（$1/2\ell \le x \le \ell$ のとき）

車両検出によって、車両の軸数、軸間距離は既知となっているので、ここで未知数となるのは各車両の軸重Ｐij であり、本車両重量計算では全ての主桁の誤差Σｅｋ が最少になるような各軸重 Pij をシンプレックス法により求める。

(2)主要コードの解説

＜関連するコード＞	＜解　説＞

NOMANGLE int CCONV wim3_cal_pi(void)
{
int i, j, it;
double Xarr[128];
double pn, qn;
int michisu;
struct ssharyo_dat *p;
 	
if(gpline->l[0] < 10.0){
_itoa(WIM_ERR_DAT_L, gmsg, 10);g_error(gmsg);
return WIM_ERR_DAT_L;
}
gNumberJikuAll = 0;
michisu = 0;
git1 = gNFif0;
git2 = 0;
for(j=0; j<gNumsharyo; j++){
p = &gsharyo_dat[j];
 	
#if defined(TRACE)
sprintf(gmsg,"p[%d]=%d¥n"
, j, p->n);
g_print(gmsg);
#endif
 	
if (p->tin_index < git1) git1 = p->tin_index;
if (p->tout_index > git2) git2 = p->tout_index;
switch (gFlagKeisanSeido){
case 0://総重量未知数
for(i=0; i<p->n-1; i++){
Flaginc[gNumberJikuAll++] = 0;
}
Flaginc[gNumberJikuAll++] = 1;
michisu++;
break;
case 1://タンデム条件
switch (p->n){
case 2://2
Flaginc[gNumberJikuAll++] = 1;
Flaginc[gNumberJikuAll++] = 1;
michisu += 2;
break;
case 3://2
Flaginc[gNumberJikuAll++] = 1;
Flaginc[gNumberJikuAll++] = 0;
Flaginc[gNumberJikuAll++] = 1;
michisu += 2;
break;
case 4://3
Flaginc[gNumberJikuAll++] = 1;
Flaginc[gNumberJikuAll++] = 1;
Flaginc[gNumberJikuAll++] = 0;
Flaginc[gNumberJikuAll++] = 1;
michisu += 3;
break;
case 5://3
Flaginc[gNumberJikuAll++] = 1;
Flaginc[gNumberJikuAll++] = 1;
Flaginc[gNumberJikuAll++] = 0;
Flaginc[gNumberJikuAll++] = 0;
Flaginc[gNumberJikuAll++] = 1;
michisu += 3;
break;
	車両重量計算処理
	
	データのチェック。
	
	総重量が未知数の場合の処理。
	
	タンデム条件を考慮した時の処理。

```
      case 6://4
/*
        Flaginc[gNumberJikuAll++] = 1;
        Flaginc[gNumberJikuAll++] = 1;
        Flaginc[gNumberJikuAll++] = 1;
        Flaginc[gNumberJikuAll++] = 0;
        Flaginc[gNumberJikuAll++] = 0;
        Flaginc[gNumberJikuAll++] = 1;
        michisu += 4;
*/
        //2002/11/07
        Flaginc[gNumberJikuAll++] = 1;
        Flaginc[gNumberJikuAll++] = 0;
        Flaginc[gNumberJikuAll++] = 1;
        Flaginc[gNumberJikuAll++] = 0;
        Flaginc[gNumberJikuAll++] = 0;
        Flaginc[gNumberJikuAll++] = 1;
        michisu += 3;

        break;
      }
      break;
    case 2://すべて未知数
      for(i=0; i<p->n; i++){
        Flaginc[gNumberJikuAll++] = 1;
      }
      michisu += p->n;
      break;
  }
 }

 if (git1 < 0) git1 = 0;
 if (git2>gNFif0) git2 = gNFif0;
#if defined(TRACE)
 sprintf(gmsg,"wim3_cal_pi lter,L,gNumberJikuAll : %d,%.3f,%d¥n"
  , gmaxIter, gpline->l[0] , gNumberJikuAll);
 g_print(gmsg);
 sprintf(gmsg,"Flaginc : %d¥t%d¥t%d¥n"
  ,Flaginc[0] ,Flaginc[1] ,Flaginc[2]);
 g_print(gmsg);
 sprintf(gmsg,"git1, git2 : %d¥t%d¥n"
  , git1
  , git2
  );
 g_print(gmsg);
#endif


#if defined(TRACE)
 sprintf(gmsg,"¥n未知数:%d¥n基線:%d¥n"
      ,michisu
      ,gline_dat[0].zerohosei[i_g]
      );
    g_print(gmsg);

 p = &gsharyo_dat[0];
 for (i = 0; i < gNumsharyo; i++) {
  sprintf(gmsg,"%d¥t%d¥t%d¥t%d¥t%d¥n"
    ,i
    ,p->n
    ,p->Indexline
    ,p->tin_index
    ,p->tout_index
    );
  g_print(gmsg);
  p++;
```

デバッグ用コード

```
  }
  g_print("¥n");
#endif

//計算区間からはみ出る車両は削除
  for(i=0; i<gNumline; i++){
    if(gline_dat[i].nsharyo){
      j = gline_dat[i].Indexsharyo[gline_dat[i].nsharyo-1];
      p = &gsharyo_dat[j];
      if((p->tin_index > 1500) && (p->tout_index > 2000)){
        gline_dat[i].nsharyo--;
        gNumsharyo--;
        for(; j<gNumsharyo; j++){
          gsharyo_dat[j] = gsharyo_dat[j+1];
        }
      }
    }
  }
  if(gNumsharyo<1){
    return WIM_ERR_CAL_T;
  }
#if defined(TRACE)
  p = &gsharyo_dat[0];
  for (i = 0; i < gNumsharyo; i++) {
    sprintf(gmsg,"%d¥t%d¥t%d¥t%d¥t%d¥t%d¥n"
      ,i
      ,p->n
      ,p->Indexline
      ,p->tin_index
      ,p->tout_index
      );
    g_print(gmsg);
    p++;
  }
#endif

//初期値の定義
  if (gFlagKeisanSeido == 1){//タンデム条件
  pn = 1;
  qn = 2;
  for(j=0; j<(MaxJiku+1)*MaxJiku; j++)
    gmat[j] = 4.0;

  for(i=1; i<michisu+1; i++)
    for(j=0; j<michisu; j++)
      gmat[i+j*MaxJiku] = ((i-1)==j? pn : qn);

  i = 0;  gmat[i*MaxJiku] = 6.29;
  i = 1;  gmat[i*MaxJiku] = 9.36;
   i = 2;  gmat[i*MaxJiku] = 3.00;
   i = 3;  gmat[i*MaxJiku] = 6.00;

  i = 0;  gmat[i*MaxJiku] = 6.29;
  i = 1;  gmat[i*MaxJiku] = 9.36;
   i = 2;  gmat[i*MaxJiku] = 5.00;
   i = 3;  gmat[i*MaxJiku] = 5.00;

  i = 0;  gmat[i*MaxJiku] = 5.7;
  i = 1;  gmat[i*MaxJiku] = 5.7;
  i = 2;  gmat[i*MaxJiku] = 7.7;
  i = 3;  gmat[i*MaxJiku] = 7.7;

  i = 0;  gmat[i*MaxJiku] = 4.375;
  i = 1;  gmat[i*MaxJiku] = 4.375;

  /* y[]をsumX[]から計算 */
```

先頭車両の計算に関係ないと思われる後続の車両を、今回の計算から削除する。

```
  for (i = 0; i < michisu+1; i++) {
    for (j = 0; j < michisu; j++)
      Xarr[j] = gmat[i+j*MaxJiku];
    gY[i] = fx4(Xarr);
  }
//  g_print("wim3_cal_pi-4¥n");
  it = Simplex(MaxJiku, gmat, gY, gNumberJikuAll, 1.0e-3, 0., 0., 0., gmaxIter, fx4);
  for (i = 0; i < michisu; i++){
    gY[i] = gmat[i*MaxJiku];
  }
  fx4_sub(gY);
//  g_print("wim3_cal_pi-5¥n");
////////////////////////////////////////////////////
  }

  if (gFlagKeisanSeido == 2){//すべて未知数

////////////////////////////////////////////////////
//  pn=(sqrt(gNumberJikuAll+1)-1+gNumberJikuAll)*10/(gNumberJikuAll*sqrt(2.0));
//  qn=(sqrt(gNumberJikuAll+1)-1)*10/(gNumberJikuAll*sqrt(2.0));
  pn = 1;
  qn = 2;
  for(j=0; j<(MaxJiku+1)*MaxJiku; j++)
    gmat[j] = 4.0;

  for(i=1; i<gNumberJikuAll+1; i++)
    for(j=0; j<gNumberJikuAll; j++)
      gmat[i+j*MaxJiku] = ((i-1)==j? pn : qn);

  i = 0;  gmat[i*MaxJiku] = 6.29;
  i = 1;  gmat[i*MaxJiku] = 9.36;
  i = 2;  gmat[i*MaxJiku] = 9.36;

  i = 0;  gmat[i*MaxJiku] = 4.00;
  i = 1;  gmat[i*MaxJiku] = 5.00;
  i = 2;  gmat[i*MaxJiku] = 5.00;

  i = 3;  gmat[i*MaxJiku] = 5.00;
  i = 4;  gmat[i*MaxJiku] = 9.00;
  i = 5;  gmat[i*MaxJiku] = 9.00;

  i = 0;  gmat[i*MaxJiku] = 5.14;
  i = 1;  gmat[i*MaxJiku] = 4.48;

  /* y[]をsumX[]から計算 */
  for (i = 0; i < gNumberJikuAll+1; i++) {
    for (j = 0; j < gNumberJikuAll; j++) Xarr[j] = gmat[i+j*MaxJiku];
    sprintf(gmsg,"Xarr : %.3f¥t%.3f¥t%.3f¥n"
      ,Xarr[0] ,Xarr[1] ,Xarr[2]);
    g_print(gmsg);
    gY[i] = fx4(Xarr);
  }
  g_print("wim3_cal_pi-4¥n");
  it = Simplex(MaxJiku, gmat, gY, gNumberJikuAll, 1.0e-3, 0., 0., 0., gmaxIter, fx4);
  for (i = 0; i < gNumberJikuAll; i++){
    gY[i] = gmat[i*MaxJiku];
  }
  fx4_sub(gY);
//  g_print("wim3_cal_pi-5¥n");
////////////////////////////////////////////////////

  }

  if (gFlagKeisanSeido == 0){//総重量未知数

////////////////////////////////////////////////////
```

シンプレックス法による最適化の計算。

```
//  pn=(sqrt(gNumsharyo+1)-1+gNumsharyo)*10/(gNumsharyo*sqrt(2.0));
//  qn=(sqrt(gNumsharyo+1)-1)*10/(gNumsharyo*sqrt(2.0));
  pn=1;
  qn=2;
  for(j=0; j<(MaxJiku+1)*MaxJiku; j++)
    gmat[j] = 4;

  for(i=1; i<gNumsharyo+1; i++)
    for(j=0; j<gNumsharyo; j++)
      gmat[i+j*gNumsharyo] = ((i-1)==j? pn : qn);

  for (i = 0; i < gNumsharyo+1; i++) {
    for (j = 0; j < gNumsharyo; j++)
      Xarr[j] = gmat[i+j*gNumsharyo];
    gY[i] = fx4(Xarr);
  }
  g_print("wim3_cal_pi-4¥n");
  sprintf(gmsg,"maxIter : %d¥n", gmaxIter);
  g_print(gmsg);
  it = Simplex(gNumsharyo, gmat, gY, gNumsharyo, 1.0e-3, 0., 0., 0., gmaxIter, fx4);
  for (i = 0; i < gNumsharyo; i++){
    gY[i] = gmat[i*gNumsharyo];
  }
  fx4_sub(gY);
/*
  for(i=0; i<gNumsharyo+1; i++){
    sprintf(gmsg,"%.3f¥t"
      ,gY[i]
    );
    g_print(gmsg);
    for(j=0; j<gNumsharyo; j++){
      sprintf(gmsg,"¥t%.3f"
        ,gmat[i+j*gNumsharyo]
      );
      g_print(gmsg);
    }
    g_print("¥n");
  }
*/
//  g_print("wim3_cal_pi-5¥n");
////////////////////////////////////////////////////

  }

  for(i=0; i<gNumline; i++){
    j = gline_dat[i].nsharyo;
    if(j){
      it = gsharyo_dat[gline_dat[i].Indexsharyo[j-1]].t2_index;
      if(gline_dat[i].fin_index < it) gline_dat[i].fin_index = it;
    }
  }

#if defined(TRACE)
//  wim3_write_sharyo_dat();
#endif
  return ret_ok;
}
```

シンプレックス法による最適化の計算。

今回計算できた車両の桁通過時刻を記録する。

３．主要なソースリスト

次頁以降に主要なソースリストを添付する。