MDI IP Velocity Algorithm

Philip H. Scherrer

SOI-TN-110 • December 1993

The following contains the definition of the IP velocity calculation. It begins with some discussion/ statements about the range of numbers expected and concludes with a c program that implements the algorithm. This document was used to generate the initial IP code.

The velocity observable is described in the IPFR document where the terms I1, I2, I3, I4, Sa, and Sc are defined.

```
Sa = I1 - I3
Sc = I2 - I4

Ssum = Sa + Sc = (I1 + I2) - (I3 + I4)

Ssum > 0 and Sa > 0 when V > 0
Ssum < 0 and Sc < 0 when V < 0

Sa = 0 at about -1800 m/s
Sc = 0 at about +1800 m/s

The max value of |Sa| or |Sb| is about 0.3 or 0.4 times I0.
The max value of |Ssum| is about 0.3 times I0.

if Ssum > 0
        alpha = Ssum/Sa
else
        alpha = Ssum/(-Sc)

alpha = 0 for V=0
```

For 12 bit a-d range of Ix is 0:4095

Assuming I0 disk center is set to 3/4 full well, I0 will be about 3000. Thus Sa max is about 1000. Alpha ranges from about +-2 for V=+-4000 m/s.

The expected range for velocity is about +-4000 m/s. We need a range of 8000 or 13 bits for V with 1 m/s resolution.

A 15 bit lookup table will be used. The full range of alpha will be set as described below, to +- 5.333 for a range of greater than +- 10,000 m/s with a resolution slightly better than 1 m/s. The scaling on alpha will be 5.333 * scale = 16384.

The division will be done by a multiply by a lookup reciprocal. The reciprocal will be determined by

dividing a BIGNUMBER by each possible denominator value. BIGNUMBER is chosen to keep the values for Sa, Sc of 100 to 1000 (1 pass) mapped to 3000 to 30000. Sa, Sc should never exceed 2000 (1 pass) so we can multiply the 1 pass numbers by 16. this means 1/1600 to 1/16000 should map into 30000 to 3000 respectively. Then BIGNUMBER should be 3000*16000 = 48,000,000. We can adopt 3 * 2^24 which is 50,331,648 for BIGNUMBER.

Then the calculated alpha will be BIGNUMBER * alpha. We want alpha of +- 2 to be scaled into say +-6000 to allow the velocity range described above. Thus we need to multiply alpha by about 3000. I will choose 3 * 1024 for this factor.

So now we have wanted_alpha_index = BIGNUMBER/(3*1024) too big.

But since the 16 bit multiply yeilds a 32 bit product, we can pick up the result in the high order word. So we WANT the target alpha index to be 2^16 times too big.

So we need to multiply by an additional (2^16) / (BIGNUMBER/(3*1024)).

Thus we need to multiply the 32 bit product by 4, then take the lower 15 of the upper 16 bits. This will give an alpha scaled +-2 into +-6144. The range for alpha is +-5.333 which corresponds to v = +-7183.

The V lookup table should be constructed for alpha_index to be in the range +-16383 in a 15 bit lookup table. The max alpha will then be 2 * (16384/6144) = 5.3333333.

The performance of the algorithm is shown as the rms values for test as follows:

vmax	I0	passes	noise	rms
4000	3000	1	no	1.2
4000	1000	1	no	3.2
4000	500	1	no	6.5
4000	400	1	no	7.9
4000	300	1	no	261.0
4000	3000	4	yes	9.5
4000	1000	4	yes	16.1
4000	500	4	yes	23.2
4000	400	4	yes	26.0
4000	300	4	yes	263.0

The code for the routine and test program follows:

REMOVE THIS AND ABOVE LINES

```
#include <stp.h>
```

```
/* GLOBAL VARIABLES DEFAULT VALUES AND DEFINED CONSTANTS */


/* spectral line specification */
#define LAMBDA_Ni        (6767.78 / 1.000291)

main()
{
double observe_MDI();
short Vcalc();
float makeVlookup();
int verbose = is_flag("-v");
int passes = (int)parameter("passes",1.0);
int ntest = (int)parameter("n",2001.0);
double I0 = parameter("I0",3000.0);
double v, l;
short I1,I2,I3,I4;
double dmin, dmax, dmean, dnoise;
int i;
double *vdiff = (double *)malloc(ntest*sizeof(double));
double *vobs = (double *)malloc(ntest*sizeof(double));
double maxvtest=parameter("vmax",2500.0);
double delta=((2*maxvtest)/(ntest-1));
float vcalib;
double noise();
int wantnoise = is_flag("-n");

makeVrecip();
vcalib = makeVlookup();

initMDI(NULL, 0, 1.0, 1.0, 0, verbose);

for (i=0; i<ntest; i++)
        {
        v = -maxvtest + delta*i;
        l = -LAMBDA_Ni * v / C;
        I1 = observe_MDI( l, 1) * I0 * passes;
        I2 = observe_MDI( l, 2) * I0 * passes;
        I3 = observe_MDI( l, 3) * I0 * passes;
        I4 = observe_MDI( l, 4) * I0 * passes;
        if (wantnoise)
                {       /* 4000->400,000 electrons so noise
                        * is sqrt(I*100 electrons) with
                        * scaling 1/100.
                        */
                I1 += sqrt((double)I1)*noise()/10.0;
```

```
                I2 += sqrt((double)I2)*noise()/10.0;
                I3 += sqrt((double)I3)*noise()/10.0;
                I4 += sqrt((double)I4)*noise()/10.0;
                }
        vobs[i] = Vcalc(I1,I2,I3,I4,passes)*vcalib;
        vdiff[i] = vobs[i] - v;
        }
statisticsd(vdiff,ntest,&dmin,&dmax,&dmean,&dnoise);
fprintf(stderr,"min=%f max=%f mean=%f stddev=%f\n",
        dmin,dmax,dmean,dnoise);
if (is_flag("-p"))
        {
        dataplotd(vobs,ntest,"vresponse",
                string("-%f:%f m/s",maxvtest,maxvtest),
                "V m/s (observed)");
        dataplotd(vdiff,ntest,"vresponse error",
                string("-%f:%f m/s",maxvtest,maxvtest),
                "delta V m/s (observed-input)");
        }
}

/*  START ONBOARD CODE  */

short Vrecip[32768];
short Vlookup[32768];

short Vcalc(short I1, short I2, short I3, short I4, short passes)
{
short Sa, Sc, Ssum, recip;
unsigned short denom, alpha;
short shift_count;
long prod_32;

shift_count =   (passes == 4 ? 2 :
                (passes == 2 ? 3 :
                (passes == 1 ? 4 : 0)));

Sa = I1 - I3;
Sc = I2 - I4;

Sa <<= shift_count;
Sc <<= shift_count;

Ssum = Sa + Sc;

if (Ssum > 0)
```

```
        denom = (unsigned short)Sa;
else
        denom = (unsigned short)-Sc;       /* note sign */

denom &= 077777;                           /* use 15-bit table */

recip = Vrecip[denom];

prod_32 = Ssum * recip;

prod_32 <<= 2;                      /* mult by 4 result in upper word */

alpha = ((unsigned)prod_32 >> 16) & 0077777;/* extract 15 bit alpha */

return(Vlookup[alpha]);
}


/* END ONBOARD CODE FOR V */

/* START TABLE GENERATION */

#define BIGNUMBER (0300000000) /* 3 * 2^24 */

makeVrecip()
{
double tmp;
short  recip;
unsigned short i,j;

for (i=1; i<=32767; i++)
        {
        tmp = (double)BIGNUMBER/i;
        if (tmp > 32767)
                tmp = 32767;
        Vrecip[i] = tmp + 0.5;
        }
Vrecip[0] = 0;
}

#include <gds.h>

float makeVlookup()
{
GDS *gVlook;
int i, nVlook;
```

```
unsigned j, k, mid;
short *tabl;
float vcalib;

gVlook = gds_in("/home0/soi/tables/MDI_Vlookup", gds_SHORT);
if (gVlook == NULL)
        fprintf(stderr,"oops\n");
vcalib = gds_info_value_d(gVlook,"BSCALE");
tabl = (short *)gVlook->data;
nVlook = gVlook->dims[0];
mid = nVlook/2;

for (i=0; i<mid; i++)
        {
        Vlookup[i] = tabl[mid + i];
        k = nVlook - i;
        Vlookup[k] = tabl[mid - i];
        }
gds_free(gVlook);

return(vcalib);
}
```