Data Analysis Working Group

**Task 3.** Cumulative cross-range interference.

Author: Pasha Ponomarenko

Date: 18 November 2013

1. **Description**

Cross-range interference (CRI) results from the multi-pulse mode of operation when at any given time the received signal represents a combination of returns from different pulses at different ranges. At a given range, different pairs of receiver samples are used to calculate different ACF lags. Furthermore, different samples are affected by CRI from different sets of range gates so that the CRI effect should be estimated for each sample separately.

For example, the same sample can be used as a pulse #2 for one range gate but as a pulse #5 for another. In each case, the contribution from the desired range is rectified through averaging the cross-products of two samples (i.e. ACF lags). Coherent returns from the "correct" range are present in both samples while incoherent CRI returns come from different sets of ranges so that their contribution to the overall ACF variance decreases with increasing number of averaged pulse sequences ~1/sqrt(N). Therefore, a substantial averaging is required for statistically reliable estimates of ACFs. Currently, N~25-30 but this number can still be insufficient for negating a large-amplitude CRI.

In order to remove the data with excessive CRI levels, the FITACF package compares lag 0 power from the analysed range, P0_check, to that from each of the interfering ranges which contribute to CRI at this particular lag, P0_i. Currently, the acceptable CRI level is considered to be when P0_check > P0_i, i.e. each of the interfering ranges has lag 0 power lower that from the checked range. If this condition is not met, then this particular sample is marked as "bad", and all related ACF lags (i.e. its cross-products with other pulses) are excluded from further analysis (fitting).

2. **Implications**

The problem here is that there is usually more than one range gate contributing to CRI for a particular pulse at a given range gate. These components are incoherent so their effect is proportional to the cumulative power from all interfering ranges. Therefore, the CRI level is generally underestimated by the current software, sometimes significantly.

3. **Proposed actions**

Instead of the gate-by-gate power comparison, we have to estimate a cumulative effect from all interfering ranges, i.e. to compare lag 0 power from the analysed range gate with a sum of lag 0 powers form all ranges contributing to CRI for a given receiver sample.

This can be done by following changes in the respective C code, **rang_badlags.c,** which are highlighted by yellow (the original code is appended to this document):

```
void lag_overlap(int range,int *badlag,struct FitPrm *ptr) {

  int ck_pulse;
  int pulse;
  int lag;
  int ck_range;
  long min_pwr;
  long pwr_ratio;
  int bad_pulse[PULSE_SIZE];  /* 1 if there is a bad pulse */
  int i;
  double nave;
  double tot_cri; /* cumulative CRI power */
  --range;  /* compensate for the index which starts from 0 instead of 1 */

  nave = (double) (ptr->nave); /* Number of averaged  pulse sequences */
  /* Filling in bad pulse array with zeroes */
  for (pulse = 0; pulse < ptr->mppul; ++pulse)
      bad_pulse[pulse] = 0;
  /* Cycle for checked receiver samples (pulses) at a given range */
  for (ck_pulse = 0;  ck_pulse < ptr->mppul; ++ck_pulse) {
    tot_cri=(double) 0;  /* Zeroing total CRI power for the next pulse sample */
    for (pulse = 0; pulse < ptr->mppul; ++pulse) {
      ck_range = range_overlap[ck_pulse][pulse] + range;
      if ((pulse != ck_pulse) && (0 <= ck_range) &&
          (ck_range < ptr->nrang))
          tot_cri=tot_cri+ptr->pwr0[ck_range];  /* Accumulating CRI power */
    }
        pwr_ratio = (long) 1;  /* Power ratio threshold */
        min_pwr =  pwr_ratio * ptr->pwr0[range];
        if(min_pwr < tot_cri)    /* Comparing lag 0 power of the checked sample (pulse) with cumulative lag 0 power from all interfering ranges */
        bad_pulse[ck_pulse] = 1;
  }

  /* mark the bad lag */
  for (pulse = 0 ; pulse < ptr->mppul; ++pulse) {
    if (bad_pulse[pulse] == 1) {
      for (i=0; i < 2 ; ++i) {
        for (lag = 0 ; lag < ptr->mplgs ; ++lag) {
          if (ptr->lag[i][lag] == ptr->pulse[pulse])
            badlag[lag] = 1;  /* 1 for bad lag */
        }
      }
    }
  }
  return;
}
```

4. **Remarks:**

I did some basic testing for this task. First, I used AJ’s simulator to check if the magnitude of the CRI from multiple ranges is indeed determined by the sum of the respective lag 0 powers, and I found this assumption to be consistent with the simulation results. Second, I applied the modified code to two weeks of real data (Rankin Inlet, 01-16 January 2012). I analysed ionospheric scatter only with SNR ("power") exceeding 6 dB. As expected, the modified code produced lesser amount of valid ACFs (~92% as compared to the current procedure) but lower median velocity error (95% of the "unmodified" value).

# Appendix

```
/* rang_badlags.c
   ==============
   Author: R.J.Barnes & K.Baker & P.Ponomarenko
*/

/*
 Copyright 2004 The Johns Hopkins University/Applied Physics Laboratory.
 All rights reserved.

 This material may be used, modified, or reproduced by or for the U.S.
 Government pursuant to the license rights granted under the clauses at DFARS
 252.227-7013/7014.

 For any other permissions, please contact the Space Department
 Program Office at JHU/APL.

 This Distribution and Disclaimer Statement must be included in all copies of
 "Radar Software Toolkit - SuperDARN Toolkit" (hereinafter "the Program").

 The Program was developed at The Johns Hopkins University/Applied Physics
 Laboratory (JHU/APL) which is the author thereof under the "work made for
 hire" provisions of the copyright law.

 JHU/APL assumes no obligation to provide support of any kind with regard to
 the Program.  This includes no obligation to provide assistance in using the
 Program or to provide updated versions of the Program.

 THE PROGRAM AND ITS DOCUMENTATION ARE PROVIDED AS IS AND WITHOUT ANY EXPRESS
 OR IMPLIED WARRANTIES WHATSOEVER.  ALL WARRANTIES INCLUDING, BUT NOT LIMITED
 TO, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE ARE
 HEREBY DISCLAIMED.  YOU ASSUME THE ENTIRE RISK AND LIABILITY OF USING THE
 PROGRAM TO INCLUDE USE IN COMPLIANCE WITH ANY THIRD PARTY RIGHTS.  YOU ARE
 ADVISED TO TEST THE PROGRAM THOROUGHLY BEFORE RELYING ON IT.  IN NO EVENT
 SHALL JHU/APL BE LIABLE FOR ANY DAMAGES WHATSOEVER, INCLUDING, WITHOUT
 LIMITATION, ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR
 CONSEQUENTIAL DAMAGES, ARISING OUT OF THE USE OR INABILITY TO USE THE
 PROGRAM."
```

```

*/

/*
 $Log: rang_badlags.c,v $
 Revision 1.5  2007/02/02 21:40:15  code
 Changed cross-range interference threshold (pwr_ratio) from 0.3*nave to 1 (line
  122 from version 1.4)
  and commented out declaration of MIN_PWR_RATIO = .3


 Revision 1.4  2003/09/13 22:39:29  barnes
 Modifications to use the new data structures.

 Revision 1.3  2001/06/27 20:48:31  barnes
 Added license tag

 Revision 1.2  2001/01/29 18:11:53  barnes
 Added Author Name

 Revision 1.1  1998/06/05 19:56:46  barnes
 Initial revision

 */

#include <stdio.h>
#include <math.h>
#include "limit.h"
#include "fitblk.h"

/* #define MIN_PWR_RATIO   .3 */

static int range_overlap[PULSE_SIZE][PULSE_SIZE];

/*  r_overlap sets up the table r_overlap which keeps track of the
 *  ranges which might cause interference.
 */

void r_overlap(struct FitPrm *ptr) {
  int ck_pulse;
  int pulse;
  int tau;

  int diff_pulse;

  /* define constants */
  tau = ptr->mpinc / ptr->smsep;

  for (ck_pulse = 0; ck_pulse < ptr->mppul; ++ck_pulse) {
    for (pulse = 0; pulse < ptr->mppul; ++pulse) {
      diff_pulse = ptr->pulse[ck_pulse] -
                      ptr->pulse[pulse];
      range_overlap[ck_pulse][pulse] = diff_pulse * tau;
    }
  }
  return;
```

```
}


/* lag_overlap marks the badlag array for bad lags */

void lag_overlap(int range,int *badlag,struct FitPrm *ptr) {

  int ck_pulse;
  int pulse;
  int lag;
  int ck_range;
  long min_pwr;
  long pwr_ratio;
  int bad_pulse[PULSE_SIZE];  /* 1 if there is a bad pulse */
  int i;
  double nave;

  --range;  /* compensate for the index which starts from 0 instead of 1 */

  nave = (double) (ptr->nave);

  for (pulse = 0; pulse < ptr->mppul; ++pulse)
      bad_pulse[pulse] = 0;

  for (ck_pulse = 0;  ck_pulse < ptr->mppul; ++ck_pulse) {
    for (pulse = 0; pulse < ptr->mppul; ++pulse) {
      ck_range = range_overlap[ck_pulse][pulse] + range;
      if ((pulse != ck_pulse) && (0 <= ck_range) &&
          (ck_range < ptr->nrang)) {
        pwr_ratio = (long) 1;  /*pwr_ratio = (long) (nave * MIN_PWR_RATIO);*/
        min_pwr =  pwr_ratio * ptr->pwr0[range];
        if(min_pwr < ptr->pwr0[ck_range])
        bad_pulse[ck_pulse] = 1;
      }
    }
  }

  /* mark the bad lag */

  for (pulse = 0 ; pulse < ptr->mppul; ++pulse) {
    if (bad_pulse[pulse] == 1) {
      for (i=0; i < 2 ; ++i) {
        for (lag = 0 ; lag < ptr->mplgs ; ++lag) {
          if (ptr->lag[i][lag] == ptr->pulse[pulse])
            badlag[lag] = 1;  /* 1 for bad lag */
        }
      }
    }
  }
  return;
}
```