The Frame Vector Library (Frv)

By D. Buskulic, I. Fiori, F. Marion, B. Mours

Version v4r22
May 29th, 2014

Summary

1 INTRODUCTION 4

2 THE FRVECT STRUCTURE 5

3 USEFUL FRAMELIB VECTOR FUNCTIONS 6

3.1 FrFileIGetVect 6
3.2 FrVectCopy, FrVectCopyTo, FrVectCopyToD,… 6
3.3 FrVectDump 6
3.4 FrVectFill 6
3.5 FrVectFindQ 6
3.6 FrVectFree 6
3.7 FrVectGetValueI 7
3.8 FrVectNew, FrVectNewTS, FrVectNew1D 7
3.9 FrVectMean 7
3.10 FrvDecimate 7

4 VECTOR COPY AND TYPE CONVERSION AND DECIMATION (FILE FRVCOPY.C) 8

4.1 FrvClone(vect, newName) 8
4.2 FrvDecimateF, FrvDecimateD, FrvDecimateI, FrvDecimateS 8

5 VECTOR ARITHMETIC (FILE FRVMATH.C) 9

5.1 FrvBias 9
5.2 FrvAdd 9
5.3 FrvCombine2 9
5.4 FrvCombine 9
5.5 FrvDelta 9
5.6 FrvDivide 9
5.7 FrvFlatten 10
5.8 FrvIntegrate 10
5.9 FrvModulus 10
5.10 FrvMult 10
5.11 FrvMultConj 10
5.12 FrvPhase 10
5.13 FrvRms 11
5.14 FrvSub 11
5.15 FrvZeroMean 11

6 THE VECTOR BUFFERING (FRVBUF) 12

6.1 Using FrvBuf 12
6.2 FrvBufFeed 12
6.3 FrvBufGetNext 12
6.4 FrvBufNew 12
6.5 FrvBufFree 13

7 THE VECTOR CORRELATION: FRVCORR 14

7.1 FrvCorrNew 14
7.2 FrvCorrFree 14
7.3 FrvCorrProc 14
7.4 FrvCorrSetPast 14

8 THE SECOND ORDER FILTER: FRVFILTER2 15

8.1 FrvFilter2New 15
8.2 FrvFilter2Proc 15
8.3 FrvFilter2Free 15

9 THE BUTTERWORTH FILTER: FRVFILTERBUT 16

9.1 FrvFilterButNew 16
9.2 FrvFilter2Proc 16
9.3 FrvFilter2Free 16

10 THE FAST FOURIER TRANSFORM (FFT) FOR REAL VECTOR (FRVRFFT) 17

10.1 FrvRFFTFor ... 17
10.2 FrvRFFTNewT ... 17
10.3 FrvRFFTNew ... 17
10.4 FrvRFFTSetDecay ... 18
10.5 FrvRFFTFree ... 18

11 THE TRANSFER FUNCTION COMPUTATION (FRVTF) ... 19

11.1 FrvTFNewT ... 19
11.2 FrvTFProc ... 19
11.3 FrvTFFree ... 19

12 LIBRARY INSTALLATION ... 20

13 THE ROOT INTERFACE ... 21

14 COPYRIGHT AND LICENSING AGREEMENT ... 22

15 HISTORY ... 23

15.1 From v4r20p1 to 4r22 (May 29, 2014) ... 23
15.2 From v4r20p1 to 4r21 (Jul 12, 2013) ... 23
15.3 From 4r20 to v4r20p1 (May 24, 2013) ... 23
15.4 From 4r19p1 to v4r20 (May 10, 2013) ... 23
15.5 From v4r19 to 4r19p1 (September 30, 2012) ... 23
15.6 From v4r18p3 to 4r19 (June 24, 2012) ... 23
15.7 From v4r18p2 to 4r18p3 (Jan 15, 2012) ... 23
15.8 From v4r18p1 to 4r18p2 (December 19, 2011) ... 23
15.9 From v4r18 to 4r18p1 (June 20, 2011) ... 23
15.10 From v4r17 to 4r18 (May 31, 2011) ... 23
15.11 From v4r16 to 4r17 (March 21, 2010) ... 24
15.12 From v4r15 to 4r16 (February 20, 2010) ... 24
15.13 Version v4r13 (April 3, 2005) ... 24
15.14 Version v4r12 (February 21, 2005) ... 24
15.15 Version v4r11 (March 03, 2004) ... 24
15.16 Version v4r10 (January 12, 2004) ... 24
15.17 Version v4r02 (February, 2003) ... 24
15.18 Version v4r01 (December 2, 2002) ... 24
15.19 Version v4r00 (August 12, 2002) ... 25
15.20 Version v3r30 (July 15, 2002) ... 25
15.21 Version v3r20 (May 2, 2002) ... 25
15.22 Version v3r10 (March 20, 2002) ... 25
15.23 Version v3r00 (November 22, 2001) ... 26
15.24 Version v2r10 (April 22, 2001) ... 26
15.25 Version v2r02 (Jan 16, 2001) ... 26
15.26 Version v2r01 (Jan 11, 2001) ... 26
15.27 Version v2r0 (Jan 10, 2001) ... 26

1 Introduction

This document describes a utility library dedicated to vector manipulation. The vectors used in this package are the Frame Library vectors (FrVect) described in the frame specification. The only part frame library part used in this library is the vector definition from the FrameL.h.

This library is intended to simplify the access of date for the various vectors. It provides most of the basic tools to do vector algebra and basic signal processing.

This code is written in C. It is thread safe and could be easily used in C or C++ software.
.

2 The FrVect Structure

The FrVect structure defined in the Frame Format specification and in the file FrameL.h contains many fields. We list here the fields that a non expert user may access in a read mode. Some fields reserved for internal management have been omitted

```
struct FrVect {
  char  *name;            /* vector name                                */
  unsigned short compress;/* 0 = no compression; 1 = gzip, ...          */
  unsigned short type;   /* vector type  (see below)                  */
  unsigned int  nData;    /* number of elements=nx[0].nx[1]..nx[nDim] */
  unsigned int  nBytes;   /* number of bytes                            */
  char  *data;            /* pointer to the data area.                  */
  unsigned int  nDim;     /* number of dimension                        */
  unsigned int *nx;       /* number of element for this dimension       */
  double *dx;             /* step size value (express in above unit)    */
  double *startX;         /* offset for the first x value               */
  char   **unitX;         /* unit name for (used for printout)          */
  char    *unitY;         /* unit name for (used for printout)          */
  FrVect *next;           /* hook for additional data                   */
  short  *dataS;          /* pointer to the data area (same as *data)  */
  int    *dataI;          /* pointer to the data area (same as *data)  */
  FRLONG *dataL;          /* pointer to the data area (same as *data)  */
  float  *dataF;          /* pointer to the data area (same as *data)  */
  double *dataD;          /* pointer to the data area (same as *data)  */
  unsigned char  *dataU;  /* pointer to the data area (same as *data)  */
  unsigned short *dataUS; /* pointer to the data area (same as *data)  */
  unsigned int   *dataUI; /* pointer to the data area (same as *data)  */
  FRULONG        *dataUL; /* pointer to the data area (same as *data)  */
  char  **dataQ;          /* pointer to the data area (same as *data)  */
  int  wSize;             /* size of one data element                   */
```

The following fields are filled only when a vector have been extract from a Frame. They may not been always available.

```
  unsigned int GTimeS;    /* vector time origin(GPS:s)                  */
  unsigned int GTimeN;    /* vector time origin(nsec modulo 1 sec)      */
  unsigned short ULeapS;  /* leap seconds between GPS and UTC           */
  int localTime;          /* Time offset = Local time - UTC (sec)       */
```

The vector type is one of the following:

```
      FR_VECT_C,     /* vector of char                                 */
      FR_VECT_2S,    /* vector of short                                */
      FR_VECT_8R,    /* vector of double                               */
      FR_VECT_4R,    /* vector of float                                */
      FR_VECT_4S,    /* vector of int                                  */
      FR_VECT_8S,    /* vector of long                                 */
      FR_VECT_C8,    /* vector of complex float                        */
      FR_VECT_C16,   /* vector of complex double                       */
      FR_VECT_STRING,/* vector of string                               */
      FR_VECT_2U,    /* vector of unsigned short                       */
      FR_VECT_4U,    /* vector of unsigned int                         */
      FR_VECT_8U,    /* vector of unsigned long                        */
      FR_VECT_1U,    /* vector of unsigned char                        */
```

Additional fields exist but are used only for management purpose.

3 Useful FrameLib vector functions

3.1 FrFileIGetVect

```
FrVect *FrFileIGetVect(FrFile *file, char *name,
                       double tStart, double duration);
```

It returns the vector of the time series for a given channel (FrAdcData, FrProcData or FrSimData) named ‘name’, starting at time tStart and lasting duration seconds. The vector boundaries are independent of the frame boundaries.
If the channel does not exist, it returns NULL.

3.2 FrVectCopy, FrVectCopyTo, FrVectCopyToD,…

```
FrVect* FrVectCopy(FrVect *vect);
```

It returns a copy of the original vector.

```
FrVect * = FrVectCopyTo(FrVect *vect, double scale,  FrVect *copy);
```

This function copies the data from vector vect to the vector copy using the scale factor 'scale' and casted to the copy vector type. This function returns NULL in case of error (malloc failed, no input vectors).
The supported output types are signed short, signed int, float, double, complex, hermitian.

```
FrVect * = FrVectCopyToD(FrVect *vect, double scale,  char *newName);
```

This function creates a copy of the input vector vect, changing the data type to double, rescaling the data by the factor scale and setting its name to newName.
This function returns NULL in case of error (malloc failed, no input vectors).

```
FrVect * = FrVectCopyToF(FrVect *vect, double scale,  char *newName);
```

Same as previous but the output is of type float (32 bits floating point).

```
FrVect * = FrVectCopyToI(FrVect *vect, double scale,  char *newName);
```

Same as previous but the output is of type int.

```
FrVect * = FrVectCopyToS(FrVect *vect, double scale,  char *newName);
```

Same as previous but the output is of type short (16 bits integer).

3.3 FrVectDump

```
void FrVectDump(FrVect *vect, FILE *fp, long debugLvl);
```

This function dumps on fp (like stdout) the vector information.
If debugLvl = 0, then nothing is printed
If debugLvl = 1 or 2, then only statistic information is printed beside the vector metadata.
If debugLvl = 3 or more, then all vector content in addition to the vector metadata.

3.4 FrVectFill

```
void FrVectFillC(FrVect *vect, char value);
```

This function adds one data element to an existing vector. The vector size is adjusted. This function is used if the vector is created with a 0 length, and then elements are added. If the vector has been created with the right size, the vector element could be directly added to the arrays like vect->dataD[4] = 3.1416;

```
void FrVectFillI(FrVect *vect, int value);   Same as FrVectFillC but for integer.
void FrVectFillS(FrVect *vect, short value); Same as FrVectFillC but for short.
```

3.5 FrVectFindQ

```
int FrVectFindQ(FrVect *vect, char *name);
```

This function returns for a vector of string the index corresponding to the given name.

3.6 FrVectFree

```
void FrVectFree(FrVect *vect);
```

This function frees all the vector space.

3.7 FrVectGetValueI

```
double = FrVectGetValueI(FrVect *vect, int index);
```

This function returns the vector value for a given index, converting to double time if needed.
It returns 0 if the vector does not exist or if the index is out of range.
This function is not as efficient as a direct access but it provides some protection and automatic type conversion.
Supported types: all non complex numbers

3.8 FrVectNew, FrVectNewTS, FrVectNew1D

```
FrVect *FrVectNew(int type, int ndim, ...)
```

This function creates a vector of any dimension. (see the Fr doc for more details).

```
FrVect *FrVectNewTS(char *name, double sampleRate,  long nData,long nBits);
```

This function creates a time series.

```
FrVect *FrVectNew1D(char *name, long type, long nData, double dx,
                    char *unitx, char *unity);
```

This function creates a one dimension vector.

3.9 FrVectMean

```
double = FrVectMean(FrVect* vect)
```

This function returns the mean value of the input vector vect.

3.10 FrvDecimate

```
FrVect * = FrVectDecimate(FrVect *vect, int nGroup,  FrVect *vOut);
```

This function decimates the data from the vector vect by averaging nGroup values together. The result is put in the vector vOut. The size of the vector vOut should be nGroup time smaller than the size of the input vector vect.
If vOut = NULL, the output result is put in the input vector.
This function returns NULL in case of error (malloc failed, no input vector).
Supported types: all types vect except complex.

4 Vector Copy and type Conversion and Decimation (file FrvCopy.c)

4.1 FrvClone(vect, newName)

```
FrVect *=FrvClone(FrVect, char *newName)
```

This function creates a new vector (FrVect structure and data area). It copies the header information but not the data.
If `newName = NULL`, the original vector name is used.
This function returns null in case of problems like malloc failed.
Supported types: all types.

4.2 FrvDecimateF, FrvDecimateD, FrvDecimateI, FrvDecimateS

```
FrVect * = FrvDecimateF(FrVect *vect, int nGroup, char* newName);
FrVect * = FrvDecimateD(FrVect *vect, int nGroup, char* newName);
FrVect * = FrvDecimateI(FrVect *vect, int nGroup, char* newName);
FrVect * = FrvDecimateS(FrVect *vect, int nGroup, char* newName);
```

These functions decimate the data from the vector vect by averaging nGroup values together. The result is put in a new vector named `'newName'` of type float, double, int or short. The size of the output vector is nGroup time smaller than the size of the input vector vect. If newName = NULL, the name of the output vector is the same as the input one.
These functions return NULL in case of error (malloc failed, no input vector).
They support all input types except complex.

Remark: FrVectDecimate could be used if there is no need to change the vector type.

5 Vector arithmetic (file FrvMath.c)

All the following functions works only for the vectors of same type. They check that both vectors have the same size. In case of error, they return a NULL pointer. Usually the output vector could be past as argument.

5.1 FrvBias

```
FrVect *FrvBias(FrVect *vect1, double bias, FrVect *vectOut)
```

This function adds bias to vector vect1.
The output is stored in vector vectOut, which should have the same size and type of vect1.
If vectOut==NULL, vect1 is modified.
Supported types: only signed data types.

5.2 FrvAdd

```
FrVect* = FrvAdd(FrVect* vect1, FrVect* vect2, FrVect* vectOut,
                                 char *newName)
```

This function computed `vectOut= vect1+vect2`. Both vectors must be of same type and size. The vector vectOut could be vect1 or vect2.
If `vectOut = NULL`, a new one is created named `newName`.
It returns the output vector or NULL in case of error. If `vectOut != NULL`, the newName argument is ignored.
Supported types: all.

Examples:

- `v3 = FrvAdd(v1, v2, NULL,"vector3");` Create and fill a new vector called vector3.
- `FrVectAdd(v1, v2, v1, NULL);` Put in v1 the sum of v1 and V2

5.3 FrvCombine2

```
FrVect* = FrvCombine(double s1, vect1, double s2, vect2, vectOut,
                                 char *newName)
```

This function computed vectOut = s1*vect1+ s2*vect2 (elements by elements). Both vectors must be of same type and size. The vector vectOut could be vect1 or vect2. If vectOut = NULL, a new one is created. Its name is then newName. It returns the vector result or NULL in case of error. If vectOut != NULL, the newName argument is ignored.
Supported types: all.
Examples: `v3 = FrvCombine2(1.3, v1, 3.3, v2, NULL, "result");`
is equivalent `to v3= 1.3*v1 + 3.3*v2`

5.4 FrvCombine

```
FrVect* = FrvCombine(int nVector, scale1, vect1, ... ,scalen, vectn,
                                 FrVect* vectOut, char* newName)
```

This function computed vectOut = scale1*vect1+ ... up to n vectors (elements by elements). All vectors must be of same type and size. The vector vectOut could be any of the input vectors. If vectOut = NULL, a new one is created. Its name is newName. It returns the vector result or NULL in case of error. If vectOut != NULL, the newName argument is ignored.
Supported types: all.
Examples: `v3 = FrvCombine(2, 1.3, v1, 3.3, v2, NULL);`
is equivalent to v3= 1.3*v1 + 3.3*v2

5.5 FrvDelta

```
int = FrvDelta(FrVect* vect, double *delta, double *previous)
```

This function computed maximum value of the differentiate vector:
`delta = max(abs(data[i+1] - data[i])).`
The result is return in delta. If `previous != NULL`, the corresponding value is used to differentiate the first vector element. On return, previous holds the value of the last vector element. This function returns zero for successful completion and a non zero value in case of error.
Supported types: all non complex.

5.6 FrvDivide

```
FrVect* = FrvDivide(FrVect* vect1, FrVect* vect2, FrVect* vectOut,
```

```
                                    char* newName)
```
This function computed vectOut= vect1/vect2 elements by elements after checking the division by 0. (if the denominator is zero then the result is set also to 0. The vector vectOut could be vect1 or vect2. If vectOut = NULL, a new one is created called newName. This function returns the vector result or NULL in case of error.
Supported types: all.

5.7 FrvFlatten

```
FrVect *FrvFlatten(FrVect *vect1, FrVect *vectOut)
```
This function puts vector extreme equal to zero, subtracting a straight line.
The output is stored in vector vectOut, which should have the same size and type of vect1.
If vectOut==NULL, vect1 is modified.
Supported types: only signed data types.

5.8 FrvIntegrate

```
FrVect *FrvIntegrate(FrVect *inVect, int n, char* newName)
```
This function performs the "n"-order Integration (or Derivation if n<0) of a signal assumed to be in the frequency domain. It takes as input a FrVect vector structure "inVect» and DIVIDES it by omega^n (omega = 2.pi.i.dx[0]). A new vector of type FR_VECT_8R (double) is created with name "newName" or the same name as the input vector if newName == NULL.
Allowed "n" values:

- n=0 trivial case: no operation is performed
- n>0 input spectrum is INTEGRATED n times
- n<0 input spectrum is DERIVED n times (i.e. it is MULTIPLIED by omega^abs(n))

This function returns the resulting vector or NULL in case of error.
Supported types: all non complex.

5.9 FrvModulus

```
FrVect* = FrvModulus(FrVect *vectIn, FrVect* vectOut, char *newName)
```
This function computes the modulus of a complex vector vectIn. The result is stored in vectOut, a real vector (Float or Double according to the complex type of vectIn).
If vectOut = NULL, a new one is created. Its name is newName or 'module(vectIn->name)' if newName = NULL.
It returns the vector result or NULL in case of error.
Supported types: works only for complex.

5.10 FrvMult

```
FrVect* = FrvMult(FrVect* vect1, FrVect* vect2, FrVect* vectOut,
                  char *newName)
```
This function computes vectOut= vect1*vect2 elements by elements. (this is not a matrix multiplication); The vector vectOut could be vect1 or vect2. Both vectors must be of same type and size.
If vectOut = NULL, a new one is created. Its name is then newName. It returns the vector result or NULL in case of error. If vectOut = NULL, the newName argument is ignored.
Supported types: all.

5.11 FrvMultConj

```
FrVect* = FrvMult(FrVect* vect1, FrVect* vect2, FrVect* vectOut,
                  char *newName)
```
This function computes vectOut= vect1*cc(vect2) elements by elements (his is not a matrix multiplication). Both vectors must be of same type and size.
The vector vectOut could be vect1 or vect2. If vectOut = NULL, a new one is created. Its name is then newName. It returns the vector result or NULL in case of error. If vectOut = NULL, the newName argument is ignored.
Supported types: FR_VECT_C8 and FR_VECT_C16.

5.12 FrvPhase

```
FrVect* = FrvPhase(FrVect *vectIn, FrVect* vectOut char *newName)
```
This function computes the phase of a complex vector vectIn. The result is stored in vectOut, a real vector (Float or Double according to the complex type of vectIn).
If vectOut = NULL, a new one is created. Its name is built as "phase(vectIn->name)"

It returns the vector result or NULL in case of error.
Supported types: works only for complex.

5.13 FrvRms

```
double Int = FrvRms(FrVect* vect, doub vectOut could be vect1 or vect2.
```

If vectOut = NULL, a new one is created. Its name is then newName.
It returns the vector result or NULL in case of error.
Supported types: all.

5.14 FrvSub

```
FrVect* = FrvSub(FrVect* vect1, FrVect*  vect2, FrVect* vectOut,
                                 char *newName)
```

This function computes vectOut= vect1- vect2.
The vector vectOut could be vect1 or vect2.
If vectOut = NULL, a new one is created. Its name is then newName.
It returns the vector result or NULL in case of error.
Supported types: all.

5.15 FrvZeroMean

```
FrVect *FrvZeroMean(FrVect *vect1, FrVect *vectOut)
```

This function subtracts the mean to vect1.
The output is stored in vector vectOut, which should have the same size and type of vect1.
If vectOut==NULL vect1 data are modified.
Supported types: only signed data types.

6 The Vector Buffering (FrvBuf)

The purpose of this object is to resize vector. It can provide vector concatenation and support vector overlap, decimation and type changes. The supported vector types are all types except when decimation is requested.

6.1 Using FrvBuf

In this example, the input vectors are extracted from a frame file. New vectors are built. See FrvBufNew for the definition of the FrvBuf parameters.

```
buffer = FrvBufNew(outSize, outSize, -1, 0);  // create a buffer object
while((frame = FrameRead(iFile)) != NULL){    // loop on all frame from one file
   vect = FrameGetV(frame,"adc1");            // find the vector for ADC 'adc1'
   FrvBufFeed(buffer, vect);                  // feed the buffer
   while(FrvBufGetNext(buffer) == 0) {        // is there a vector in the buffer?
      FrVectDump(buffer->output, stdout, 2);} // use the output vector
```

Another example could be found in the file FrvBufTest.c.

6.2 FrvBufFeed

```
int FrvBufFeed(FrvBuf *buffer, FrVect *vect);
```

This function copies the input data to the internal buffer. The size of the inputs vectors could change from one call to the next one but the type need to be the same.
It returns:

- 0 if at least one output is ready
- 1 if more calls to FrvBufFeed are needed to get a complete output vector.
- 2 in case of error
- 3 in case of buffer reset due to timing mismatch

6.3 FrvBufGetNext

```
int FrvBufGetNext(FrvBuf *buffer);
```

This function prepares the next output buffer available in buffer->output.
It returns:

- 1 if a call to FrvBufFeed is needed,
- 0 if a vector is available

6.4 FrvBufNew

```
FrvBuf* FrvBufNew(int outSize, int step, int outType, int decimate,
                          int delay);
```

This function creates an FrvBuf object.
Arguments:

- `outSize` is the number of element of the vector output
- `step` is the element index shift performed at each new vector output. To get a 50% overlap for the output vectors, you need to set step = `outSize`/2. The get no overlap at all, you need to set step = `outSize`.
- `outType` is the output vector type (see the Frame spec.). -1 means use input vector type. Warning: the only type conversion supported are those leading to a FR_VECT_2S (short), FR_VECT_4S (int), FR_VECT_4R (float), FR_VECT_8R (double).
- `decimate` is the output data rate reduction factor. For instance, decimate = 4 means that four elements of the input vector are averaged to produce one element of the output vector. If decimate has a negative value, a pure decimation of -decimate is performed without averaging.
- `delay` give the delay between the input vector and the output vector express in number of output samples.

Example: We give some results for various parameters set. For all the cases, we assuming that we feed the buffer with vector containing: 0, 1, 2, 3, 4, ... :

- for outSize=4 step=2 type=-1 decimate=0 the vectors returned are (0 1 2 3); (2 3 4 5); (4 5 6 7);...
- for outSize=4 step=4 type=-1 decimate=0 the vectors returned are (0 1 2 3); (4 5 6 7); (8 9 10 11);...
- for outSize=4 step=4 type=-1 decimate=3 the vectors returned are (1 4 7 10); (13 16 19 21); (24 27 30 33);...

- for outSize=4 step=2 type=-1 decimate=3 the vectors returned are (1 4 7 10); (7 10 13 16); (13 16 19 21);...

6.5 FrvBufFree

```
void FrvBufFree(FrvBuf* buffer);
```

This function frees the buffer object and all associated vectors, including the output vector.

7 The vector correlation: FrvCorr

This object computes the correlation among two vectors. The type of the input vector could be any non complex type, but the output vectors are of type FR_VECT_8R (double).

7.1 FrvCorrNew

```
FrvCorr* FrvCorrNew( int maxlag , int normalize)
```

Creates a new FrvCorr structure
If normalize ==0 the correlation is unbiased.
Past values are set to zero.

7.2 FrvCorrFree

```
void  FrvCorrFree(FrvCorr* corr)
```

This function frees the space allocated to correlation

7.3 FrvCorrProc

```
int FrvCorrProc(FrvCorr *corr, FrVect *vect1, FrVect *vect2)
```

Calculates the correlation among vectors 1 and 2:

```
c[i]=Sum_k(v1[k]*v2[k+i])
```

the correlation of the two vectors are stored in FrVect * corr->present

the correlation averaged over consecutive frames is in FrVect * corr->average

- pointers to zero lag correlation are available in corr->present0 and corr->average0.
- If corr->normalize==0 the unbiased correlation is calculated.
- For the first call, past values are set to zero: they are recorded for subsequent calls

and can be changed with FrvCorrSetIni

7.4 FrvCorrSetPast

```
void FrvCorrSetPast(FrvCorr* corr, double * past1, double * past2, int flag)
```

Set vectors past times conditions v1_[-n] = past1[n-1]
if flag ==0 past times are set to zero.

8 The second order filter: FrvFilter2

This object filters the data for one vector.
The type of the input vector could be any non complex type, but the output vector (named filter->output) is of type FR_VECT_8R (double).
The length of the output vector is the same as the one of the input vector, which must remain the same during the processing.

8.1 FrvFilter2New

```
FrvFilter2* FrvFilter2New(double a2, double a1, double a0,
                          double b2, double b1, double b0, char option)
```

This function creates an FrvFilter2 Object. It returns NULL in case of problem.
The 6 filter parameters are defined in the following way: The transfer function (Laplace transform s=i*w) is:

```
Yout(s)     a2 * s**2 + a1 * s + a0
------- =  ------------------------
Yinp(s)     b2 * s**2 + b1 * s + b0
```

Example 1: first order low-pass filter at f0 with unity gain at dc
a2=0. a1=0. a0=1. b2=0. b1=1/(2*pi*f0) b0=1.
Example 2: second order low-pass filter at f0 with quality factor Q
and unity gain at dc (e.g. pendulum)
a2=0. a1=0. a0=1. b2=1/(2*pi*f0)**2 b1=1/(2*pi*f0*Q) b0=1
Example 3: integrator with time constant t
a2=0. a1=0. a0=1. b2=0. b1=t b0=0.

8.2 FrvFilter2Proc

```
int FrvFilter2Proc(FrvFilter2 *filter, FrVect *vect)
```

Apply the filter algorithm to the input vector vect.
The input vector is unchanged by this call.
It returns 0 in case of successful completion.

8.3 FrvFilter2Free

```
void FrvFilter2Free(FrvFilter2* filter)
```

Comment: This function free all memory associated to the FrvFilter2 object.

9 The Butterworth filter: FrvFilterBut

This object filters the data for one vector.
The type of the input vector could be any non complex type, but the output vector (named filter->output) is of type FR_VECT_8R (double).
The length of the output vector is the same as the one of the input vector, which must remain the same during the processing.

9.1 FrvFilterButNew

```
FrvFilterB *FrvFilterButNew(int order, double fMin, double fMax, char* name)
```

This function creates a Butterworth filter object: FrvFilterB. It returns NULL in case of problem.

If fMin is different from zero, the filter will be a high pass filter with cutoff frequency (3dB attenuation) fMin.
If fMax is different from zero, the filter will be a low pass filter with cutoff frequency fMax.
If fMin and fMax are both different from zero, it will be a pass band filter between fMin and fMax.

The order of the filter is define by the first parameter.

The last parameter “name” is the name given to the filter by the user. NULL is a valid option.

9.2 FrvFilter2Proc

```
FrVect* FrvFilterButProc(FrvFilterB *filter, FrVect *vect)
```

Apply the filter algorithm to the input vector vect.
The input vector is unchanged by this call.
It returns a pointer to the output filter or NULL in case of successful completion.
The vector output could also be retrieved at any time as filter->output

9.3 FrvFilter2Free

```
void FrvFilterButFree(FrvFilterB* filter)
```

This function frees the memory associated to the FrvFilterB object.

10 The Fast Fourier Transform (FFT) for Real Vector (FrvRFFT)

This Fast Fourier Transform FFT object use FFTW. It works with real vector for the direct FFT, the Fourier Transform in this case is 'halfcomplex' (the negative-frequency amplitudes for real data are
the complex conjugate of the positive-frequency amplitudes) but the result is stored has a standard complex vector. All internal computation are done in double precision. The results are also stored in double precision.
Remark: The FFTW is no more included in the Frv distribution. You need first to install FFTW, unless you use Pm install to do the Frv installation.

10.1 FrvRFFTFor

```
FrvFFT* FrvRFFTFor(FrvFFT *fft, FrVect *vect)
```

This function performs the forward FFT algorithm "fft" to the vector "vect". It returns the result in the FrvFFT structure as output (vector fft->output). The input vector has to be real. The output vector (fft->output) will be a complex. If on input fft=0, builds a standard fft structure from the vector vect with the option = "AHNPS". This function returns NULL in case of error.
The input vector is unchanged by this call.

10.2 FrvRFFTNewT

```
FrvFFT* FrvRFFTNewT(char *option,  double duration, int nAverage)
```

This function creates an FFT Object.
The parameters are:

- `option` string could be set to NULL or could contain one of the following characters:
  - H to apply an Hann window (normalized to not change the amplitude)
  - O to overlap the data by half a vector. This is useful if you work on a continuous stream with an Hann window.
  - P to suppress the pedestal (average value)
  - S to compute the spectrum (amplitude).
  - A to compute the average spectrum (this force the use of the option S).Remark: the averaging is done on the power (amplitude**2).
  - N to normalized the result as if it's noise (i.e. unit are by sqrt(Hz)). The default units are absolute units.

  Example: option = "HS" means Hann window + amplitude spectrum computed
- `duration` is the length of the FFT in second. This is converted to number of points at the first FrvRFFTFor call, using the sampling rate of the input vector.
- `nAverage` is the number of FFT used to compute the averaged values. Once the number of FFT reaches `nAverage`, the averaging switch to exponentially decay of averaging with
  `average = decay*previous + (1-decay)*last` and `decay = 1.-1./nAverage;`

Useful output data:

`fft->output`	the complex vector with the last FFT output
`fft->amplitude`	the spectrum (amplitude) of the last FFT. The option “S” must be used
`fft->amplitude`	the average spectrum. The option “A” must be used
`fft->nFFT`	the number of FFT performed.

Storage: If the input vector as nData elements, the FrvFFT use at 2*nData double (or float if option bit 6 = 1). The function will reserve nData double (or float) for each spectrum computed or if a window is used.

Comment on the FFT normalization: Normalization is computed for single sided spectrums (negative frequencies folded on the positive one). With this convention:

- When we are looking at noise (option “N” used, i.e. units are 1./sqrt(Hz)) the average level for a Gaussian noise with an rms value = A*sqrt(sampleRate) (in the time domain) is A*sqrt(2) (in the frequency domain)
- When we are looking at a signal (absolute unit, i.e. no option “N”) then for a sine wave of amplitude A (in the time domain) the FFT amplitude is A/sqrt(2) (this is the rms value for a sin function).

10.3 FrvRFFTNew

```
FrvFFT* FrvRFFTNew(char *option,  int fftSize, int decimate)
```

This function creates an FFT Object, like for FrvRFFTNewT, but specifying the number of points instead of a time, and with the option of a simple decimation before doing the FFT.

- The option string as the same meaning as in FrvRFFTNewT
- `ffSize` is the number of points (after the optional decimation) used to computed the FFT.
  If fftSize < 1, the number of point used is the input vector size
- `decimate` is the decimation factor apply before the FFT. For example, decimate = 4 means that 4 values of the input vector will be averaged together before entering the FFT algorithm. The is no anti aliasing low pass filter, just an averaging of the values.

10.4 FrvRFFTSetDecay

```
void FrvRFFTSetDecay(FrvRFFT* fft, double decay)
```

This function change the default decay value used to compute the mean values of the amplitude spectrum. This is useful only is the FFT object has been created by FrvRFFTNew and not FrvRFFTNewT which contains the nAverage parameter.

If previous is the previous value and last the last computed value of one bin, decay is defined as:

`mean = decay*previous + (1-decay)*last` with `decay` in the range 0 to 1.

Remark: as long as the number of FFT is less than 1/decay, we performed only a plain averaged on the total number of FFT (i.e. decay = 1./nFFT).
The default value for decay is 0.999999, meaning a plain averaging for 10^6 FFT calls.
This function returns 0 in case of success or 1 in case of problem.

10.5 FrvRFFTFree

```
void FrvRFFTFree(FrvFFT* fft)
```

This function frees the memory associate to the FFT object.

11 The Transfer function computation (FrvTF)

This object computes a transfer function, assuming that the input signal has wide band noise. The useful FrVect members are:

- `tf->output` The complex transfer function
- `tf->modulus` The module of transfer function
- `tf->phase` The phase of the transfer function
- `tf->correlation` The correlation in the frequency domain
- `tf->errorM` The error on the modulus of the transfer function (if option 'E' is used)
- `tf->errorP` The error on the phase of the transfer function (if option 'E' is used)
- `tf->coherence` The coherence (if option 'C' is used)

11.1 FrvTFNewT

```
FrvTF* FrvTFNewT(char* option, double duration, int nAverage);
```

- option could contain one of the following character:
  - C to compute the coherence
  - E to compute the error in the transfer function modulus and phase.
- Duration is the length in seconds on which the FFT are computed.
- nAverage define on how many FFT are averaged.
  - If (nAverage < 0) then just a plain averaged is performed.
  - Otherwise decay = 1.-1./nAverage; If `previous` is the previous value and `last` the last computed value, the result is defined as: mean = decay*previous + (1-decay)*last. Remark: as long as the number of call is less than `1/decay`, we performed only a plain averaged on the total number of fft (ie `decay = 1./nCall`).

11.2 FrvTFProc

```
int FrvTFProc(FrvTF *tf, FrVect *ouput, FrVect *inputNoise)
```

Compute the transfer function define as the ratio between the FFT of the input noise and output signal. The transfer function module (vector tf->modulus) is computed as the ratio between the mean amplitude of each FFT. The transfer function phase is the mean value of the phase extracted from each transfer functions.
The input vectors are unchanged by this call.
It returns 0 in case of successful completion.

11.3 FrvTFFree

```
void FrvTFFree(FrvTF* tf)
```

This function frees the memory associate to the FrvTF object.

12 Library installation

Before installing Frv, you need first to install FFTW version 3.

The latest version of the Frv library could be found in wwwlapp.in2p.fr/virgo/frameL. Assuming that you have downloaded the gzip tar file for the library, you need to:

- unzip the file by using the command gunzip Frv.tar;gz
- untart the file by using the command tar xvf Frv.tar
- go in the mgr directory and change in the script "makesh" the path to
    - the frame Library (FR)(should be at least version v6r00)
    - the FFTW library
    - the rootLibrary (ROOTSYS) (Or comment the root part if root is not installed)
- then run the script "makesh"

The Frv library could also by install using CMT.

For any question, e-mail to mours@lapp.in2p3.fr

13 The ROOT interface

Like for the frame library, a ROOT compatible shared library is available.

To use it, you need to you need to update the PATH and LD_LIBRARY_PATH to include the FrvROOT.so binary directory (named by your system). Then if you start root from the Frv/vXX/root subdirectory, it will execute the FrvLogon.C which load everything you need.

Examples of ROOT macros are available in the root subdirectory. Play with them to get an idea of what you could do.

14 Copyright and Licensing Agreement

This is a reprint of the copyright and licensing agreement of the Frv Library:

15 History

15.1 From v4r20p1 to 4r22 (May 29, 2014)

- FrvFFT.c: Add a protection in the case the number of requested average is zero.
- Minor fix in FrvFDFilter: correct some missing half bin offset when computing the windows.
- Remove the FrvLinFilter from documentation since this part is now obsolete and should not be used.
- FrvFilterBut:
    - Enrich the FrvFilterBut to do high pass and band pass filters.
    - Optimize space: created only one output vector per linked list of filters.
- FrvFilter2: add a protection to reset filter registers and output vector if not normal numbers are present.
- Remove FrvMean: Fr FrVectMean must be used Fr

15.2 From v4r20p1 to 4r21 (Jul 12, 2013)

Thanks to Frédérique Marion for suggestions, finding and reporting problems and bugs.

- Fix the function FrvBufStillToGo which was not working properly.

15.3 From 4r20 to v4r20p1 (May 24, 2013)

- Move to Fr/v8r19p1 and FFTW/v3r3p30

15.4 From 4r19p1 to v4r20 (May 10, 2013)

- Move to Fr/v8r19 and FFTW v3r3p3
- Update the requirement file to follow the cleanup made on the Fr requirements and the logic used to work with/without root.

15.5 From v4r19 to 4r19p1 (September 30, 2012)

Thanks to Emmanuel Pacaud for suggestions, finding and reporting problems and bugs.

- Use Fr/v8r17p1 and FFTW v3r3p21

15.6 From v4r18p3 to 4r19 (June 24, 2012)

Thanks to Loic Rolland and Emmanuel Pacaud for suggestions, finding and reporting problems and bugs.

- Use Fr/v8r16 and FFTW v3r3p20
- Remove the functions FrvCopyTo, FrvCopyToD, … and replace them by FrVectCopyTo, … using #define statements
- In FrvFDFilterFree: add the free of the tfBins objects (this was forgotten and therefore producing a memory leak).
- FrvFilter.c: add the FrvFilterBut object and functions.

15.7 From v4r18p2 to 4r18p3 (Jan 15, 2012)

- Add a check to expand compressed vector in FrvBufFeed. This is used also by FrvFDFilter for instance, and was producing wrong results for compressed frames.
- Use Fr/v8r15p7

15.8 From v4r18p1 to 4r18p2 (December 19, 2011)

- Fix some bug in the logic of reading list of points when defining a frequency domain filter (FrvFDFilter) by its TF

15.9 From v4r18 to 4r18p1 (June 20, 2011)

- Fix the requirement file to work when ROOT is not available

15.10 From v4r17 to 4r18 (May 31, 2011)

Thanks to Loic Rolland and Didier Verkindt for suggestions, finding and reporting problems and bugs.

- Use Fr/v8r15
- Add a protection in FrvFDFilterBuildTFBin

- Fix the type of FrvChFeed in the header file to return now a double, as the .c file is already doing it.
- In FrvRFFTFree, add the free of the vector “temp” and the buffer “buffer”.

15.11 From v4r16 to 4r17 (March 21, 2010)

Thanks to Loic Rolland for suggestions, finding and reporting problems and bugs.

- Change the type of window used in the frequency domain filter (FrvFDFilterOutProc function) to remove glitches present in the h(t) reconstruction.

15.12 From v4r15 to 4r16 (February 20, 2010)

Thanks to Loic Rolland for suggestions, finding and reporting problems and bugs.

- Fix the FrvTF function to support two channels with different frequencies.
- Add the FrvFDFilter functions

15.13 Version v4r13 (April 3, 2005)

Thanks to Alain Masserot for suggestions, finding and reporting problems and bugs.

- Add the FrvCoGap, FrvBandRMS objects
- Add the FrvTFReset, FrvTFNewT functions
- Fix a bug in FrvLinFiltFree if the output vector is the input vector

15.14 Version v4r12 (February 21, 2005)

Thanks to Fabrice Beauville, Frederique Marion and Alain Masserot for suggestions, finding and reporting problems and bugs.

- Add the FrvCh module
- Remove the FrvChi2 code
- Fix the FrCalloc in case of FFTW malloc use
- FrvBuf:
    - Now the code handle frame vector discontinuity
    - Fix a bug when using non zero delay
    - Add the function FrvBufNewID to handle the delay like in the past.

15.15 Version v4r11 (March 03, 2004)

Thanks to Damir Buskulic, Leone Bosi and Frederique Marion for suggestions, finding and reporting problems and bugs.

- Update the requirement file
- FrvCopyTo: Fix C++ style comments and handle FFTW malloc.

15.16 Version v4r10 (January 12, 2004)

Thanks to Leone Bosi and Frederique Marion for suggestions, finding and reporting problems and bugs.

- FrvFFT: Move to FFTW3
- FrvRFFTFor: Now this function returns NULL in case of error.
- FrvCopyTo: This function now handles the complex and hermitian types.
- FrvCopy.h, FrvMath.h: Protect a define agains multiple inclusion
- FrvModulus and FrvPhase: rewrite to work now with hermitian vectors (FR_VECT_8H and FR_VECT_16H)
- FrvMult handle now the hermitian vectors (FR_VECT_8H and FR_VECT_16H)
- Add the functions: FrvInterpolate, FrvInterpolatePhase, FrvFillGaus, FrvGaus, FrvUniform.

15.17 Version v4r02 (February, 2003)

Thanks to Michele Punturo and Frederique Marion for suggestions, finding and reporting problems and bugs.

- FrvBuff: Support now a negative value for decimate to tell to do a true decimate without averaging
- Build a debug version of the library in the standard makescript

15.18 Version v4r01 (December 2, 2002)

Thanks to Damir Buskulic, Isidoro Ferrante, Frederique Marion Jean-Marie Teuler and Gabriele Vedovato for suggestions, finding and reporting problems and bugs.

- FrvBuff

    - Fix a bug in FrvBufIni: when decimation was applied, the output vector time step was not updated.
    - Allow a call of FrvBufStillToGo after a new and before a first feed.
    - Fix the type of the lastGTime variable (int->double).
- FrvFFT
    - Update the normalization in case of decimation according to the change made in FrvBufIni.
    - Increase the default decay time to .999999
- FrvMath: improve FrvStat to work even if the size of the input vector has changed.
- FrvBufTest: fix a compilation error.
- FrvLinFilt.c: update doc for Butterworth filter.
- FrvFilter: Add "next" element to support linked list storage.

15.19 Version v4r00 (August 12, 2002)

Convert to Frame format version 5 and frame library version v5r00 and higher

15.20 Version v3r30 (July 15, 2002)

- FrvMath:
    - add the FrvStat object
    - add the FrvIntegrate function
    - add a protection in FrvCombine (for malloc failed)
- FrvCorr.c fix bug if the type of the second vector is not the same as the type of the first vector and in the normalization.
- FrvLinFilt.c
    - Add the function FrvLinFiltButtLowToHigh
    - Fix a bug in the initialization of filter with na or nb <= 1.
- FrvTFTest.c: fix RAND_MAX definition and the number of arguments for FrvTFNew.
- testFT.cc: add a missing ';'
- FrvBuffFree: add protection when the object was never used.
- FrvMath.c:
    - Add the FrvIntegrate function
    - Add protection for malloc failed in FrvAdd, FrvCombine2, FrvDivide,FrvMult, FrvScale, FrvMulConj

15.21 Version v3r20 (May 2, 2002)

- FrvTF:
    - Rename the "numerator" vector to "correlation"
    - Fix a normalization error for the TF error.
- FrvLinFilt: Add optimized code for Butterworth filter up to order 6.

15.22 Version v3r10 (March 20, 2002)

- FrvFFT:
    - Fix a bug in the normalization when the decimation option was used.
- FrvTF:
    - Fix a bug which was producing a segmentation fault if the coherence option was not used.
    - Add a protection to allow the use of decimate <= 0.
    - Add the function FrvTFFree and FrvTFError.
    - Add the option 'E' to compute the error on the transfer function.
- FrvLinFiltButt: initialize all variables.
- makesh: Update the script to fix some ROOT problem. Add also the file src/FrvLinkDef.h
- root/testTF.cc
- Fix a bug in the FrvTF call
- Change the random generator to be machine independent.
- Rename FrvROOT.so to libFrvROOT.so
- Remove the source code of FFTW
- Add the object FrvSmartDecimate (new file)

15.23 Version v3r00 (November 22, 2001)

- FrvCopy:
  - Propagate time information in FrvClone and FrvCopy
  - Protect FrvClone against null input vector.
  - Remove FrvDecimate (function replaced by FrVectDecimate).
- FrvMath:
  - Remove the function FrvMinMax which is now part of the FrameLib.
  - Add the functions FrvMultConjc, FrvZeroMean, FrvFlatten, FrvBias.
  - Fix a bug in FrvPhase (the sign of the phase was wrong).
- FrvFFT:
  - Fix a bug when computing the average amplitude (the average was done on the amplitude, not the power).
  - Fix a bug on the overall amplitude (it has been reduce by a factor sqrt(2)).
- FrvTF:
  - Fix a bug when computing the average amplitude (the average was done on the amplitude, not the power).
  - Add the option to compute the coherence. WARNING:The API for FrvTFNew has been changed (add the option field).
- Add the modules FrvCorr and FrvLinFilter
- Change the installation script to an sh scrit. (mgr/makesh).
- FrvBuf:
  - Full rewrite.
  - Check the time (GPS) constituency for the input vector if this information is available.
  - Add the possibility to add a delay between the input and output vectors. WARNING: the API for FrvBufNew has been changed (add the delay parameters)

15.24 Version v2r10 (April 22, 2001)

- Fix a bug in FrvTF.c : the code was not working properly if decimation was used.
- Fix a bug in FrvFFT.c the imaginary part of the FFT for on point was stored in the next points. This was introducing a small bias in the FFT and sometime a crash.
- Upgrade FrvBufFeed to be able to change the size of the input vector from one call to the next one.

15.25 Version v2r02 (Jan 16, 2001)

- Fix a bug in FrvFFT.c (the function crash if the vector had not unitY label).
- Fix a bug in FrvTF.c (the output vector was not properly initialized).
- Fix a bug in FrvBuf.c (the step performed was not correct when decimation was apply).
- Remove the function FrameFnR (replace by FrameReadRecycle from Fr).

15.26 Version v2r01 (Jan 11, 2001)

- Add the function FrameFnR. This function will be move in the future to the FrameLib.

15.27 Version v2r0 (Jan 10, 2001)

- Start history