The Frame Vector Library (Frv)

Version v3r30
July 15, 2002
By D. Buskulic, I. Fiori, I. Ferrante, F. Marion, B. Mours

Summary

- Introduction
- The FrVect Structure
- Usefull FrameLib vector functions (Vector constructors, dump, destructor)
- Vector Copy and type Conversion and Decimation
- How to change the vector length: vector buffuring (FrvBuf)
- Vector arithmetic
- The vector correlation: FrvCorr
- The linear filter: FrvLinFilt
- The second order filter: FrvFilter2
- The Fast Fourier Transform for Real vector (FrvRFFT)
- Transfer Function
- Library installation (Basic Install, ROOT usage)
- The ROOT interface
- History

Introduction

This document describes a utility library dedicated to vector manipulation. The vectors used in this packages 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 frome the FrameL.h. This library is intend to simplify the access of date for the various vector. It provide 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.

The FrVect Structure

The FrVect srtucture 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 managemend have been omitted

```
struct FrVect {

  char  *name;            /* vector name                              */
  unsigned short compress;/* 0 = no compression; 1 = gzip, ...        */
  unsigned short type;    /* vector type  (see bellow)                */
  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 field s exist but are used only for management purpose

---

Usefull FrameLib vector functions

Back to summary

- **FrVect *FrFileIGetV(FrFile *file, char *name, double tStart. double duration);** Return the vector for a given ADC, PROC or SIM data time series starting at time tStart and lasting 'duration' seconds. The vector boundaries are adjusted to the frame boundary.
- **FrVect *FrXXXDataReadT(FrFile *file, char *name, double tStart);** With XXX = Adc, Proc or Sim. These function performed a random access in a file and return the vector for a given (name) Adc, Proc or Sim data time serie starting at time tStart. The vector boundaries are adjusted to the frame boundary. If tStart = 0, it returns the data for the first frame in the file.
- **FrVect *FrameGetV(FrameH *frame, char *name);** Return the vector for a given ADC, PROC or SIM data time series.

- **FrVect *FrXXXDataGetV(FrameH *frame, char *name);** With XXX = Adc, Proc or Sim. These functions return the vector for a given (name) Adc, Proc or Sim data time serie.

- **FrVect* FrVectCopy(FrVect *vect);** Returns a copy of the original vector.
- **void FrVectDump(FrVect *vect, FILE *fp, long debugLvl);** Dump on fp (like stdout) a vector
- **void FrVectFillC(FrVect *vect, char value);** This function add one data element to an existing vector. The vector size is adjusted. Usually the vector is created with a 0 length, and then elements are added.
- **void FrVectFillI(FrVect *vect, int value);** Same as FrVectFillC but for integer.
- **void FrVectFillS(FrVect *vect, short value);** Same as FrVectFillC but for short.
- **int FrVectFindQ(FrVect *vect, char *name);** This function returns for a vectro of string the index corresponding to the given name.
- **void FrVectFree(FrVect *vect);** Free all the vector space
- **double = FrVectGetV(FrVect *vect, int index);** Return the vector value for a given index. They do the type conversion if needed. They returns 0 if the vector do not exist or if the index is out of range. This function is not as efficient as a direct acces but it provides some protection and automatic type conversion. Supported types: all non complex numbers
- **FrVect *FrVectNew(int type, int ndim, ...)** To create a vector of any dimension. (see the Fr doc for more details).
- **FrVect *FrVectNewTS(char *name, double sampleRate, long nData,long nBits);** Create a time serie.
- **FrVect *FrVectNew1D(char *name, long type, long nData, double dx, char *unitx, char *unity);** Create a one dimension vector

---

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

Go to: **FrvClone,** **FrvCopy,FrvCopyToX, FrvCopyTo,Back to summary**

FrvClone(vect, newName)

- Syntax: **FrVect *=FrvClone(FrVect, char *newName)**
- This function create a new vector (FrVect structure and data area); It copy 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.

FrVectCopy(vect)

- Syntax: **FrVect *=FrVectCopy(FrVect, char *newName)**
- This is a full vector copy from the FrameLib (FrVectHeader + data part). It does not change the vector type. This function returns null in case of problems like malloc failed.
- Supported types: all types.

FrvCopyToF, FrvCopyToD, FrvCopyToI, FrvCopyToS

- Syntax:
  FrVect * = FrvCopyToF(FrVect *vect, double scale, char* newName);
  FrVect * = FrvCopyToD(FrVect *vect, double scale, char* newName);
  FrVect * = FrvCopyToI(FrVect *vect, double scale, char* newName);
  FrVect * = FrvCopyToS(FrVect *vect, double scale, char* newName);
- These functions create a new vector of type float (FrvCopyToF), double (FrvCopyToD), int (FrvCopyToI) or short (FrvCopyToS). The data are copy using the scale factor 'scale' and casted to the proper type. The new vector will have the same name as the original one except if a newName is provided (value non NULL).
- These functions return NULL in case of error (malloc failed, no input vector).
- Supported input types: all types except complex.

FrvCopyTo

- Syntax:
  FrVect * = FrvCopyTo(FrVect *vect, double scale, FrVect *copy);
- This function copy the data from vector vect to the vector copy using the scale factor 'scale'and casted to the vector copy type.
- This function returns NULL in case of error (malloc failed, no input vectors).
- Supported types:
  - all types for vect except complex.
  - float, double, int and short for copy.

FrvDecimateF, FrvDecimateD, FrvDecimateI, FrvDecimateS

- Syntax:
  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).
- Supported input types: all types except complex.

FrvDecimate

- It has been replace by FrVectDecimate: Syntax:
  FrVect * = FrVectDecimate(FrVect *vect, int nGroup, FrVect *vOut);
- This function decimates the data from the vector vect by averaging nGroup values togehter. 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.

Vector arithmetic (file FrvMath.c)

Go to: FrvAdd, FrvCombine2, FrvCombine, FrvDelta, FrvDiv,FrvMean,FrvMult,FrvModulus,FrvPhase,FrvRms,FrvScale,FrvSub,Back to summary

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 a the output vector could be past as argument.

FrvBias

- Syntax: **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.

FrvAdd

- Syntax: **FrVect* = FrvAdd(FrVect* vect1, FrVect* vect2, FrVect* vectOut, char *newName)**
- This function computed vectOut= vect1+vect2. 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

FrvCombine2

- Syntax: **FrVect* = FrvCombine(double s1, vect1, double s2, vect2, vectOut, char *newName)**
- This function computed vectOut = s1*vect1+ s2*vect2 (elements by elements). 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

FrvCombine

- Syntax: **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). The vector vectOut could be any of the input vector. 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

FrvDelta

- Syntax: **int = FrvDelta(FrVect* vect, double *delta, double *previouse)**
- 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.

FrvDivide

- Syntax: **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.

FrvFlatten

- Syntax: **FrVect *FrvFlatten(FrVect *vect1, FrVect *vectOut)**
- This function puts vector extrema equal to zero, subtracting a straigth 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.

FrvIntegrate

- Syntax: **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.

FrvMean

- Syntax: double **Int = FrvMean(FrVect* vect, double *mean)**
- This function computes the mean value of the input vector vect. The result is put in mean. It returns 1 in case of failure (vector not found), or 0 in case of success.
- Supported types: all non complex types

FrvModulus

- Function prototype: **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.

FrvMult

- Syntax: **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. 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.

FrvMultConj

- Syntax: **FrVect* = FrvMult(FrVect* vect1, FrVect* vect2, FrVect* vectOut, char *newName)**
- This function computes vectOut= vect1*cc(vect2) elements by elements. (this is not a matrix multiplication); 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.

FrvPhase

- Function prototype: **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 build as "phase(vectIn->name)"
- It returns, the vector result or NULL in case of error.
- Supported types: works only for complex.

FrvRms

- Syntax: double **Int = FrvRms(FrVect* vect, double *mean, double *rms)**
- This function computes the rms and mean value of the input vector vect. The result is put in mean and rms. It returns 1 in case of failure (vector not found), or 0 in case of success.
- Supported types: all non complex types

FrvScale

- Syntax: **FrVect* = FrvScale(double scale, FrVect* vect1, FrVect* vectOut, newName)**
- This function computes vectOut= scale*vect1 elements by elements. (this is not a matrix multiplication);
- 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.

FrvStat

- This object compute an sliding mean and rms values.
- Syntax: **FrvStat* = FrvStatNew()**
- or **FrvStat* FrvStatProc(FrvStat *stat, FrVect *vect)**

FrvSub

- Syntax: **FrVect* = FrvSub(FrVect* vect1, FrVect* vect2, FrVect* vectOut)**
- 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.

FrvZeroMean

- Syntax: **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.

The Vector Buffering (FrvBuf)

The purpose of this object is to resize vector actracted from frame. It can provides vector concatenation and support vector overlap, decimation and type changes. The suported vector type are all types excpet when decimation is requested.

Go to: Using FrvBuf,FrvBufFree,FrvBufFeed,FrvBufGetNext, FrvBufInit,FrvBufNew,Back to Summary

Using FrvBuf

In this example, the input vectors are extract from a frame file. New vectors are build . 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
```

An other example could be found in the file FrvBufTest.c.

FrvBufFree

- syntax: **void FrvBufFree (FrvBuf* buffer);**
- This function free the buffer object and all associated vectors, including the output vector.

FrvBufFeed

- syntax: **int FrvBufFeed (FrvBuf *buffer, FrVect *vect);**
- This function copy the input data to the internal buffer. The size of the iinputs 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 call to FrvBufFeed are needed to get a complete output vector.
  - 2 in case of error
  - 3 in case of error due to timing missmatch

FrvBufGetNext

- Syntax: **int FrvBufGetNext(FrvBuf *buffer);**
- This function prepare the next output buffer available in buffer->output.
- It returns:
  - 1 if a call to FrvBufFeed is needed,
  - 0 if a vector is available

FrvBufInit

- Syntax: **int FrvBufInit (FrvBuf *buffer, FrVect *signal, FrVect *ref);**
- This function initiliaze the output buffer according to vect type. The vector provided should be identical (for the size and type point of view) to the one provide to FrvBufFeed. A call to this function is not mandatory and is automatically performed at the first FrvBufFeed call.
- This function returns 1 in case of problem or 0 if everything is OK.

FrvBufNew

- Syntax :**FrvBuf* FrvBufNew(int outSize, int step, int outType, int decimate, int delay);**
- This function create an FrvBuf object.
- Arguments:
  - outSize is the number of element of the vector output

  - step is the element index shift performd at each new vector output. To get a 50% overlapp for the output vectors, you need to set step = outSize/2. The get no overlapp at all, you need to set step = outSize.
  - outType is the output vector type (see the Frame spec.). -1 means use input vetor 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.
  - delay give the delay between the input vector and the output vector express in number of output samples.
- Example: We give here some result 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);...

The vector correlation: FrvCorr

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

FrvCorrNew

- Syntax: **FrvCorr* FrvCorrNew(int maxlag , int normalize)**
- Creates a new FrvCorr structure
- If normalize ==0 the correlation is unbiased.
- Past values are set to zero.

FrvCorrFree

- Syntax: **void FrvCorrFree(FrvCorr* corr)**
- This function frees the space allocated to correlation

FrvCorrInit

- Syntax: **int FrvCorrInit(FrvCorr* corr, FrVect *vect1, FrVect *vect2)**
- Initilizes the corr object: creates all output structures
  It is automatically called by FrvCorrProc
- Returns: 0 in case of success, >0 otherwise

FrvCorrProc

- Syntax: **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

FrvCorrSetPast

- Syntax: **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.

The linear filter: FrvLinFilt

This object filters the data for one vector. The type of of the input vector could be any non complex type, but the output vector (named filter->output) is of type FR_VECT_8R (double).

FrvLinFiltButt

- Syntax: **FrvLinFilt *FrvLintFiltButt(double fp, double fs, double tp, double ts, int order, double fc)**
- Comment: This function creates a FrvlinFilt object consinsting of a low pass butterworth filter.
  You can specify the filter using the two set of parameters order, fc or fp,fs,tp,ts.

  if order>0, a filter of order "order" with a -3dB frequency of fc is created, other inputs are ignored.

  if order=0 the filter order is calculated according to the parameters fp,fs,tp,ts, where:

  fp upper edge of pass band in units of sampling frequency.
  fs lower edge of stop band in units of sampling frequency
  tp value of transfer function modulus at fp.
  ts value of trensfer function modulus at fs.

  if order<0 and all the other parameters are different from zero, then the minimum order between the two given possibilities is chosen. This option could be used to give an upper limit to filter length.

FrvLinFiltButtLowToHigh

- Syntax: **void FrvLintFiltButtLowToHigh(FrvLinFilt* filter)**
- Comment: This function transforms a low pass butterworth filter in an high pass one, of the same order and with the same cut-off.

FrvLinFiltFree

- Syntax: **void FrvLinFiltFree(FrvLinFilt* filter)**
- Comment:This function frees all the memory associate to the FrvLinFilt object.

FrvLinFiltInit

- Syntax: **int FrvLinFiltInit(FrvLinFilt* filter, FrVect *vect)**
- Initializes the filter object: creates all output structures.
- Returns: 0 in case of success, 1 otherwise.

FrvLinFiltSetIni

- Syntax: **void FrvLinFiltSetIni(FrvLinFilt* filter, double * ym, double * xm)**
- Sets filter initial conditions x_n, n= -1.....-(na-1) y_m, m = -1....-(mb-1)
  These are stored in the order ym[0]=y_-1, ym[1]=y_-2....ym[na-2]=y_na-1
  xm[0]=x_-1, xm[1]=x_-2....xm[na-2]=x_mb-1

FrvLinFiltSetGain

- Syntax: **void FrvLinFiltSetGain(FrvLinFilt* filter, double freq, double gain)**
- Sets filter gain at frequency freq .
  freq is in sampling frequency units:

FrvLinFiltNew

- Syntax: **FrvLinFilt* FrvLinFiltNew(double * a, double * b, int na, int mb)**
- Creates a new filter with coefficients a[0...na-1] and b[0...mb-1].
  See FrvLinFiltProc to understand how the output is calculated.
  Initial conditions are set to zero: can be changed by FrvLinFiltSetIni

FrvLinFiltMult

- Syntax: **FrvLinFilt* FrvLinFiltMult(FrvLinFilt* filter2, FrvLinFilt* filter1)**
- Multiplies filter1 and filter2 by convolution, and creates a new filter.
  If necessary, old filters can be freed by the user.

FrvLinFiltCopy

- Syntax: **FrvLinFilt * FrvLinFiltCopy(FrvLinFilt * filter0)**
- Copies a filter to a new one. Filter output is not initialized.

FrvLinFiltAddZP

- Syntax: **FrvLinFilt * FrvLinFiltAddZP(FrvLinFilt * filter0, double fase, double modulo, int type)**
- Creates a new filter adding a single real zero or pole or a couple of complex conjugates ones in the Z plane
- fase : zero-pole phase, in units of 2*pi, or frequency in units of sampling frequency
  If it is zero or 0.5, only one is zero or pole is added.
- modulo: zero-pole module. If it is a pole, and if modulo>1, the filter is unstable
- type: 0 adds a zero
  type: 1 adds a pole only if it is stable
  type: 2 adds a pole stable or not.

FrvLinFiltProc

- Syntax: **int FrvLinFiltProc(FrvLinFilt *filter, FrVect *vect)**
- Applies the filter "filter" to the vector "vect". The output is stored in FrVect* filter->output .
  It is calculated according to:
  a_0 * y_n = -Sum_(k=1...na) a_k *y_(n-k)+ Sum_(k=0...mb) b_k *x_(n-k)
  For the first call, initial condition are set to zero: they are recorded
  for subsequent calls and can be changed with FrvLinFiltSetIni.

- The input vector is unchanged by this call.
- It returns 0 in case of successfull completion.

FrvSmartDecimate

This object perform a multistep decimation.

Example: The initialization of the filter bank::

```
FrvSmartD *SDep01=FrvSmartDInit(10,50.,2.,0.99,0.001);
```

This ask to decimate of a factor ten starting from 50 Hz.
The frequencies up to 2 Hz are attenuated a maximum of 0.99 and the frequencies which are aliased in the frequency band up to 2 Hz are attenuated at least by a factor 0.001 .

Processing:

FrvSmartDProc(SDep01,vector,NULL);

The second order filter: FrvFilter2

This object filter the data for one vector. The type of of the input vector could be any non complex type, but the output vector (named filter->output) is of type FR_VECT_8R (double).

FrvFilter2Free

- Syntax: **void FrvFilter2Free(FrvFilter2* filter)**
- Comment:This function free all the memory associate to the FrvFilter2 object.

FrvFilter2Init

- Syntax: **int FrvFilter2Init (FrvFilter2 *filter, FrVect *vect);**
- This function initiliaze the filter (especially the filter->output vector) according to vect type. The vector provided should be identical (for the size and type point of view) to the one provide to FrvFilter2Proc. A call to this function is not mandatory and is automatically performed at the first FrvFilter2Proc call.
- This function returns 0 in case of success or 1 in case of problem.

FrvFilter2New

- Syntax: **FrvFilter2* FrvFilter2New(double a2,double a1,double a0, double b2,double b1,double b0har option)**
- This fonction create 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 : first order high-pass filter at f0 with unity gain at dc
a2=0. a1=1/(2*pi*f0) a0=1. b2=0. b1=0. b0=1.

Example 3 : 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 4 : integrator with time constant t
a2=0. a1=0. a0=1. b2=0. b1=t b0=0.

FrvFilter2Proc

- Syntax: **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 successfull completion.

The Fast Fourrier Transform (FFT) for Real Vector (FrvRFFT)

This Fast Fourrier Transform FFT object use FFTW. It works with real vector for the direct FFT, the Fourrier Tranform 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 compuation 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.

FrvRFFTFree

- Syntax: **void FrvRFFT (FrvFFT* fft)**
- Comment:This function free all the memory associate to the FFT object.

FrvRFFTFor

- Syntax: **FrvFFT* FrvRFFTFor(FrvFFT *fft, FrVect *vect)**
- Performed the forward FFT algorithm "fft" to the vector "vect". 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".
- The input vector is unchanged by this call.

FrvRFFTInit

- Syntax: **void FrvRFFTInit(FrvRFFT* fft, FrVect *vect)**
- This function initiliaze the FFT (especially the fft->output vector) according to vect type. The vector provided should be identical (for the size and type point of view) to the one provide to FrvRFFTFor. A call to this function is not mandatory and is automatically performed at the first FrvFFFTFor call.
- This function returns 0 in case of success or 1 in case of problem.

FrvRFFTNew

- Syntax: **FrvFFT* FrvRFFTNew(char *option, int fftSize, int decimate)**
- This fonction create an FFT Object.
  - The option string could be set to NULL or could contain one of the following character:
    H to apply an hanning window (normalized to not change the amplitude)
    O to overlapp the data by half a vector. This is usefull if you work on a continous stream with an Hanning window.
    P to supress the pedestal (average value)
    S to compute the spectrum (amplitude)
    A to compute the average spectrum (this force the use of the option S).
    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 hanning window+amplitude spectrum computed
  - 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 togheter before entering the fft algorithm.
- 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.

FrvRFFTSetDecay

- Syntax: **void FrvRFFTSetDecay(FrvRFFT* fft, double decay)**
- This function change the default decay value used to computed the mean values of the amplitude spectrum. If previous is the previous value and last the last computed value, decay is defined as:
  mean = decay*previous + (1-decay)*last
  So decay should be in the range 0 to 1.
- Remark: as long as the number of FFT is less than 1/decay, we perfomed only a plain averaged on the total number of fft (ie decay = 1./nFFT).
- The default value for decay is 0.99
- This function returns 0 in case of success or 1 in case of problem.

The Transfer function computation (FrvTF)

This object compute a transfer function, assuming the 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)

FrvTFError

- Syntax: **void FrvTFError(FrvTF* tf)**
- Comment:This function compute the transer function errors. It is automatically called by FrvTFProc is the option 'E' has been used when creating the TF object.

FrvTFFree

- Syntax: **void FrvTFFree(FrvTF* tf)**
- Comment:This function free all the memory associate to the FrvTF object.

FrvTFInit

- Syntax: **int FrvTFInit (FrvTF *tf, FrVect *signal, FrVect* reference);**
- This function initiliaze the transfer function (especially the tf->output vector) according to vect type. The vector provided should be identical (for the size and type point of view) to the one provide to FrvTFProc. A call to this function is not mandatory and is automatically performed at the first FrvTFProc call.
- This function returns 0 in case of success or 1 in case of problem.

FrvTFNew

- Syntax: **FrvTF* FrvTFNew(char *option, int tfSize, int decimate)**
- This fonction create an Transfer Function object.
  - The option string could be set to NULL or could contain one of the following character:
    C to compute the coherence
    E to compute the error in the transfer function modulus and phase.
    Example: option = "C"means the coherence will be computed

    - tfSize is the number of points (after the optional decimation) used to computed the TF. If tfSize < 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 togheter before entering the fft algorithms. Decimate=0 means no decimation.
- 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.

FrvTFProc

- Syntax: **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 transfert function module (vector tf->modulus) is computed as the ratio between the mean amplitude of each FFT. The transfert function phase is the mean vaule of the phase extracted from each transfer functions.
- The input vector is unchanged by this call.
- It returns 0 in case of successfull completion.

FrvTFSetDecay

- Syntax: **void FrvTFSetDecay(FrvTF* tf, double decay)**
- This function change the default decay value used to computed the mean modulus and the phase.If previous is the previous value and last the last computed value, decay is defined as:
  mean = decay*previous + (1-decay)*last
  So decay should be in the range 0 to 1.
- Remark: as long as the number of call is less than 1/decay, we perfomed only a plain averaged on the total number of fft (ie decay = 1./nCall).
- The default value for decay is 0.999
- This function returns 0 in case of success or 1 in case of problem.

Library installation

Before installing Frv, you need first to install FFTW.

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)
    - the FFTW library
    - the rootLibrary (ROOTSYS) (Or comment the root part if root is not installed)
- then run the script "makesh"

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

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.

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

History

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

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.

Version v3r10 (March 20, 2002)

- FrvFFT:
  - Fix a bug in the normalisation 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 independant.
  - rename FrvROOT.so to libFrvROOT.so
- remove the source code of FFTW
- Add the object FrvSmartDecimate (new file)

Version v3r00 (November 22, 2001)

- FrvCopy:
  - Propgate 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) constitance for the input vector if this infomration 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 paramters)

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.

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 initilazed).
- 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).

Version v2r01 (Jan 11, 2001)

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

Version v2r0 (Jan 10, 2001)

- Start history