

AnyPlot

A system for reading, displaying, and analyzing
time series data

Wayne C. Crawford

Table of Contents

| | |
|---|----|
| <u>A system for reading, displaying, and analyzing time series data</u> | 1 |
| <u>Wayne C. Crawford</u> | 1 |
| Table of Contents | 1 |
| Overview | 3 |
| Building Blocks | 4 |
| <u>The AnyPlot classes</u> | 4 |
| Getting Started | 6 |
| <u>Requirements</u> | 6 |
| <u>Setting Up</u> | 6 |
| <u>Getting Help</u> | 6 |
| Reading data | 7 |
| <u>ap_readAH</u> | 7 |
| <u>ap_readASCII</u> | 7 |
| <u>ap_readSAC</u> | 7 |
| <u>ap_readPSEGY</u> | 8 |
| <u>ap_readWEBB</u> | 8 |
| Manipulating, analyzing, and plotting data | 9 |
| Writing data | 10 |
| <u>ap_writeASCII</u> | 10 |
| <u>ap_writeSAC</u> | 10 |
| Examples | 11 |
| <u>To concatenate 10 SACfiles</u> | 11 |
| <u>Removing the effect of several channels on another in a multi-record data file</u> | 11 |
| <u>Creating an ap_datafile object</u> | 11 |
| ap_view: an interactive interface to AnyPlot | 13 |
| <u>ap_view(tdata)</u> | 13 |
| <u>ap_view(datafile)</u> | 13 |
| <u>ap_view()</u> | 13 |
| AnyPlot Function Reference | 18 |
| Summary: | 18 |
| ap_set..... | 19 |
| ap_timedata functions | 20 |
| ap_timedata | 20 |
| calcAddress | 21 |
| calcNumSamps | 21 |
| calcTime..... | 21 |
| cat | 23 |
| cut | 23 |
| extract..... | 23 |
| filter | 24 |
| get | 25 |
| merge..... | 26 |
| plot..... | 27 |
| plotfilt..... | 29 |
| resample | 30 |
| set | 31 |
| subtract_xf..... | 32 |
| ap_spectra functions | 33 |
| ap_spectra..... | 33 |
| get..... | 35 |
| plot..... | 36 |

| | |
|--|------------------------------------|
| plotcoher..... | 37 |
| set | 37 |
| ap_xferfun functions..... | 38 |
| ap_xferfun | 38 |
| get | 40 |
| merge..... | 40 |
| plot..... | 41 |
| set | 41 |
| ap_datafile functions..... | 42 |
| ap_datafile | 42 |
| get | 43 |
| set | Erreur ! Signet non défini. |
| ap_filtparm functions..... | 48 |
| ap_filtparm | 48 |
| * | 48 |
| plot..... | 49 |
| response..... | 49 |
| ap_plotinfo functions..... | 50 |
| ap_plotinfo | 50 |
| calcplotpos | 50 |
| get | 51 |
| set | 52 |
| ap_time functions..... | 53 |
| ap_time..... | 53 |
| +..... | 54 |
| -..... | 54 |
| Developer notes..... | 55 |
| <u>Writing ap_routines</u> | <u>55</u> |
| <u>Writing ap_read* routines</u> | <u>55</u> |

Overview

AnyPlot is a collection of Matlab objects and functions to read, display, convert, and analyze time series data. The AnyPlot functions allow you to

- Read data from several different formats, and write routines to read in other formats.
- Perform simple data analysis on and plot the data
- Load the data into Matlab arrays for more advanced data analysis
- Save the data in other formats.

AnyPlot is written in the Matlab language because:

- 1) the Matlab language is fairly simple, so the code is easy to write and to modify
- 2) Matlab is scriptable, so you can automate tasks,
- 3) Matlab has many advanced data analysis functions, and
- 4) Matlab is available on most computing platforms, and the language is identical on these platforms.

Matlab is a high-level language that is easier to modify than standard programming languages, so users can modify the AnyPlot routines to better suit their needs. A large number of commercial and public-domain data analysis routines are already written for Matlab. AnyPlot uses only the functions included in the basic Matlab installation, so anyone who uses Matlab can use AnyPlot. If you have specialized toolboxes, you can read in and prepare your data with AnyPlot and then analyze it using the toolbox functions.

The downside of writing AnyPlot in Matlab is that it is relatively slow, space inefficient, and you have to own Matlab. We could avoid these problems by compiling the Matlab code, but this would reduce the flexibility of the tools. With computers becoming faster, cheaper, and with more memory all the time, we choose to keep flexibility at the expense of efficiency.

Reading and plotting data in AnyPlot is fairly simple. For example, to read and plot an AH data file named 'data.ah', type:

```
tdata = ap_readAH('data.ah');  
plot(tdata);
```

To save the data as a SAC file name 'data.sac', type:

```
ap_writeSAC(tdata, 'data.sac');
```

To calculate and plot a 2048-point power spectra and coherence, type:

```
sdata = ap_spectra(tdata, 'windowsize', 2048);  
figure(2); plot(sdata);  
figure(3); plotcoher(sdata);
```

To calculate and plot the transfer function between data channels 2 and 3, type:

```
xfdata = ap_xferfun(sdata, 2, 3);  
figure(4); plot(xferfun);
```

To look at one section of data, you can add parameters to the plot() command. Writing out the plot() parameters by hand can get tedious, so there's an interactive viewing environment called ap_view() which allows you to zoom in and out on data, pick phase arrivals, and calculate basic spectra, coherences and transfer functions. If you want to perform more complicated data analysis, you can write script that mix Anyplot and standard Matlab functions.

Building Blocks

AnyPlot is a set of object-oriented functions. That means that it is made up of **objects**, which are like variables except that they

- can contain many different variables
- have functions specifically written to act on them, and
- protect the information inside of them.

The type of information contained in the object and the functions that can work on it depend on the object's **class**. For example, the `ap_timedata` and `ap_spectra` classes have a `plot()` function defined for them, so when you type `plot(data)`; you get a time series plot if `data` is an `ap_timedata` object, and a spectra plot if `data` is an `ap_spectra` object.

The AnyPlot classes

AnyPlot consists of three "core" classes: `ap_timedata`, `ap_spectra`, and `ap_xferfun`, and four "helper" classes: `ap_datafile`, `ap_filtparm`, `ap_plotinfo` and `ap_time`. The workhorse is `ap_timedata`, which contains the time series data plus information such as the data sampling rate, start time, clock drift, channel names, etc. Here is a brief overview of the AnyPlot classes:

The `ap_timedata` class allows you to modify, merge, view, slice and dice time series data. `ap_timedata` objects, which are generally created in the `ap_read*` functions, contain the time series data and all the information about it.

The `ap_spectra` class lets you calculate and plot spectra and coherences from `ap_timedata` objects.

The `ap_xferfun` class lets you calculate, merge and plot transfer functions from `ap_spectra` objects. You can also use `ap_xferfun` objects to remove the effect of one channel on another when calculating spectra and coherences.

The `ap_datafile` class allows you to store information about the files containing the time series data. For data storage formats that contain their own info (such as the "AH" seismic data format) there's little or no need for an `ap_datafile` object. `ap_datafile` objects are especially useful for holding information about more basic formats, such as raw binary or ASCII data. An `ap_datafile` object can contain instrument variables, such as the sampling rate and instrument response, and storage variables, such as the length of one record and where time information is located.

The `ap_filtparms` class allows you to specify, manipulate and view the instrument response. `ap_filtparms` objects contain the instrument response in pole-zero format, but you specify the response in other formats such as Butterworth filter parameters. Each channel of an `ap_timedata` or `ap_datafile` object has an `ap_filtparms` object. These values are used to correct for the instrument response when calculating `ap_spectra`.

The `ap_plotinfo` class serves as storage for plot information s (fontsize, linecolor, etc) for each of the core classes. Each "core" classes uses only some of the `ap_plotinfo` object variables, and different core classes may interpret these variables slightly differently.

The `ap_time` class allows you to easily calculate and compare different date/times. It knows the different number of days per month, the conversion from months and days to julian days, and which years are leap years, etc, so you don't have to.

Each of the AnyPlot classes has an "initializer" function and one or more ways to access the internal variables. The initializer function has the same name as the class and is used to create an object of that class. For example:

```
tdata=ap_timedata(sampfreq,data);
```

creates an `ap_timedata` object named `tdata`, with each row in `data` interpreted as one channel and the sampling rate set to `sampfreq`. You may never need to call the `ap_timedata`

function unless you write a new `ap_read*` function or if you want to use the AnyPlot routines to look at some data you've already loaded into Matlab.

The `ap_spectra` initializer calculates spectra and coherence from an `ap_timedata` object:

```
sdata = ap_spectra(tdata);
```

The `ap_xferfun` initializer calculates a transfer function between two channels in an `ap_spectra` object:

```
xdata = ap_xferfun(sdata,drivingchannel,responsechannel);
```

You can access variables within an AnyPlot object using subscripts and the `get()` and `set()` functions. In general, subscripts access and/or modify data arrays and `get()` and `set()` access and modify other internal variables. The exceptions are the `ap_filtparm` and `ap_time` classes, which are simple enough that all the internal variables are accessible using subscripts.

Getting Started

Requirements

You must have Matlab version 5.0 or later to use AnyPlot,. AnyPlot takes advantage of the object-oriented programming environment introduced in Matlab 5.0, so it will not run on older versions.

Setting Up

To use AnyPlot, add the AnyPlot directory path to the standard matlab path. The best way to do this is by putting the AnyPlot folder in your matlab directory, and adding the lines

```
addpath('${matlabdir}/AnyPlot');  
addpath('${matlabdir}/AnyPlot /ap_view');
```

to the file startup.h in your matlab directory

Getting Help

You can get basic help for any of the AnyPlot functions by typing

```
help ap_function
```

in Matlab, where ap_function is the function name.

Reading data

One of the principal goals of AnyPlot is to let you easily read in time series data. I've only written routines to read in a few different kinds of time series data, but you can fairly easily copy and modify one of the following routines to read in other data formats. By convention, the routines to read in the data are called "ap_readXXXX", where "XXXX" is the data format. Any of the routines can be called with an ap_datafile object as the argument, although several (such as ap_readAH, ap_readSAC, and ap_readSEGY) only need the "filename" variable and can also be called with only this filename as a variable. If you call a routine using an ap_datafile object and if the data file contains info conflicting with the object info, the object values are replaced by those read in. The existing routines are:

The general format is (or should be, I'm modifying my routines as of March 2005)

```
ap_readXXX(file,startTime,readLength,channels);
```

Calls

```
[tdata,dfile] = ap_readXXX(file,startAt,readLength);
```

Arguments

| <u>name</u> | <u>type</u> | <u>default</u> | <u>description</u> |
|-------------|-----------------------------|----------------|--|
| file | char -or- ap_datafile | none | the datafile name contains the datafile name, but can also contain information necessary to read the data properly, the start time, or information about the channel gains and filters. If empty, the program will create an appropriate datafile object by prompting the user. |
| startAt | ap_time | 0 | Where to start reading the data. If a scalar, the time from the start of the file in seconds. If ap_time, the time to start reading at. |
| readLength | scalar | 0 | the number of seconds to read in. If 0, read to the end |

Return value

| | | |
|-------|-------------|--------------------------------|
| tdata | ap_timedata | the data read in from the file |
| dfile | ap_datafile | datafile information |

ap_readAH

```
tdata = ap_readAH("data.AH");
```

```
tdata = ap_readAH(datafile); % datafile is an ap_datafile object
```

Read in data stored in the "AH" seismic data format. This format officially requires the UNIX tool "XXXXXX" to allocate variable sizes, but I wrote my function to not depend on UNIX, so I can't guarantee it works on all AH files. It does work on the AH files I tried on a Sun workstation and a Macintosh computer.

ap_readASCII

```
tdata = ap_readASCII("data.ASC");
```

```
tdata = ap_readSCII(datafile); % datafile is an ap_datafile object
```

Read in ASCII data. If you enter just the filename, it will read the entire file, assigning a channel to each column and assuming a sampling rate of 1 Hz.

ap_readSAC

```
tdata = ap_readSAC("data.SAC");
```

```
tdata = ap_readSAC(datafile); % datafile is an ap_datafile object
```

Read in data stored in the "SAC" seismic data format.

ap_readPSEGY

```
tdata = ap_readPSEGY("data.PSEGY");  
tdata = ap_readPSEGY(datafile);      % datafile is an ap_datafile object
```

Read in data stored in the "Pascall SEG Y" seismic data format.

ap_readWEBB

```
tdata = ap_readWEBB(datafile);      % datafile is an ap_datafile object
```

Read in data stored in the WEBB raw binary data format. Information must be entered in datafile so that the function knows how to interpret the data.

There are a lot of parameters to an `ap_datafile` object, so I wrote a graphical tool to help create `ap_datafile` objects. Just type `ap_view` without any arguments. The first window creates an `ap_datafile` object. Hit the "Save As..." button after you've created/modified the object, and all the commands necessary to create your `ap_datafile` object will be saved in a file. After that, you can hit the "Cancel" button to escape the dialog or, if you've entered a valid filename, you can hit "OK" to interactively view the data.

Manipulating, analyzing, and plotting data

Once you've read data into an `ap_timedata` object, you can modify it, merge it, plot it, and/or analyze it. For example, if you want to merge channels from two `ap_timedata` objects with the same sampling frequency and data length, type

```
tdata = merge(tdata1,tdata2);
```

Or you may want to concatenate two consecutive chunks of data:

```
tdata = cat(tdata1,tdata2)
```

or grab a certain range of data

```
tdata = cut(tdata_in,startaddress,endaddress);
```

or only use/view some of the data channels:

```
tdata = extract(tdata_in,channellist);
```

You can merge data sampled at different rates using the `resample` function

```
tdata_b = resample(tdata_b,get(tdata_a,'sampfreq'));  
tdata = merge(tdata_a,tdata_b);
```

and you can calculate what time corresponds to a given address:

```
time = time_calculate(tdata,10000);
```

or what address corresponds to a given time:

```
address = time_locate(tdata,"9/17/96 10:00:00");
```

You can also filter the data:

```
fdata=filter(tdata,'BwHigh',4,10); % 4-pole, 10 Hz Butterworth hipass filter
```

You can also set parameters such as the channel names, instrument response and units:

```
tdata=set(tdata,'cname',2,'Vert');
```

You can plot the data statically or interactively:

```
plot(tdata, ...); % Static display  
ap_view(tdata); % Interactive display
```

The `"..."` means that you can add options, in this case specifying how to plot the data. The plotting parameters can be specified either in `plot()` or `set()`. Specifying them in `plot()` affects only the current plot. Specifying them with `set()` makes them the default plotting option for that object. The `ap_view()` environment is described later on.

You can then calculate and plot spectra, coherences, and transfer functions:

```
sdata = ap_spectra(tdata, ...)  
figure(2); plot(sdata, ...);  
figure(3); plotcoher(sdata, ...);  
xdata = ap_xferfun(sdata,1,2); %driving channel=1, response=2  
figure(4); plot(xdata, ...);
```

You can `set()` only plot variables for `ap_spectra` and `ap_xferfun` objects. To change any of the other parameters you have to rerun `ap_spectra(...)` or `ap_xferfun(...)`.

There is also a `plot()` function for `ap_filtparms` objects so that you can verify instrument responses. To view all of the instrument responses within an `ap_timedata` object `tdata`, type:

```
plotfilt(tdata);
```

To display information about any AnyPlot object, type the object's name. To save this information in a string (named `str` in this example), type:

```
str=char(object);
```

Writing data

Since all of the writes and reads come from or to an `ap_timedata` object, you can use AnyPlot to convert between different data types of data files. In general, it makes sense to only write to "smart" files (ones that have headers to hold all of the instrument and data information)., but you can also write to ASCII files, since they're ubiquitous and easy to read. Even "smart" data formats may not hold all the information in an `ap_timedata` object. For example, the SAC data format doesn't store the instrument response. Despite this disadvantage, I recommend that you always save your data in one of the "official" formats below (or one you create yourself) rather than simply saving an `ap_timedata` object, because the `ap_timedata` object specification is not fixed and will probably change with time.

`ap_writeASCII`

```
ap_writeASCII(tdata, filename)
ap_writeASCII(tdata, filename, 'noheader');
```

Writes the data out in an ASCII file, one column per channel. By default, the data are preceded by a header containing information about the `ap_timedata` object.

`ap_writeSAC`

```
ap_writeSAC(tdata)
ap_writeSAC(tdata, filename);
```

Writes the data out in the SAC seismic data format, with a different file for each channel. If filename isn't specified, the file will be named `YYYYJJJ.HHMM.CH.sac`, where `YYYY`, `JJJ`, `HH`, and `MM` are the year, julian day, hour and minute of the start of the data, and `CH` is the channel number. If filename is specified, then the file will be named `filename.CH.sac`.

Examples

The Examples directory contains a few examples to help you get started using AnyPlot.

To concatenate 10 SACfiles

Here we read in 10 SAC files of one hour each containing pressure data sampled at 1 Hz, concatenate them, and then merge them with data from a SAC file containing 24 hours of seismic data at 0.1 Hz. Finally, we save the resampled and concatenated pressure data for future use.

```
tdataP = [];  
for i = 1:10,  
    fname = sprintf('20031020.%02d00.P.sac',i);  
    tdataP = cat(tdataP,ap_readSAC(fname));  
end  
tdataZ = ap_readSAC('20031020.0100.Z.sac');  
tdata = merge(resample(tdata,0.1),tdataZ);  
plot(tdata);  
ap_writeSAC(tdataP);
```

Removing the effect of several channels on another in a multi-record data file

Assume that the file "data.wbb" contains seafloor seismic data, with channels 1-4 containing horizontalX, horizontalY, vertical, and pressure data, respectively. Also assume that the pressure and horizontal seismic background noise below 0.1 Hz is dominated by non-seismic noise (this is often the case):

Creating an ap_datafile object

You can create an ap_datafile object however you want. However, there are a couple of conventions I use so that you can create these objects once and read them in anytime in the future. Once again, I use the set() function in an ASCII file to construct the ap_datafile object to avoid saving an ap_datafile object directly. The ap_datafile object definition will probably change in the future, but the set() call won't.

```
function dfile = myPARMs(filename);  
  
dfile = ap_datafile('LDE0',filename);  
dfile = set(dfile,'instname','2003 STAG Test Compliance Meter');  
%-----  
% ENTER RAW DATA PARAMETERS FOR LDE0 DEVICES  
dfile = set(dfile,'RawRecLen',1024);  
dfile = set(dfile,'RawHeadBytes',28);  
dfile = set(dfile,'RawTWPos',24);  
dfile = set(dfile,'RawCustom2',23); % location of sampling rate byte in  
header  
  
dfile = set(dfile,'numchans',4);  
  
gz = 1500; % V/(m/s)  
np = 100; % s  
du = 1e6; % counts/volt  
filt = ap_filtparm(du*gz,[],1/np,'fromBwFreqs'); %  
parameters for velocity  
filt = ap_filtparm(filt.gain,[filt.poles 0],filt.zeros); % Convert to  
accel  
  
dfile = set(dfile,'chname',1,'D');  
dfile = set(dfile,'chname',2,'X');  
dfile = set(dfile,'chfilt',2,filt);  
dfile = set(dfile,'chunits',2,'m/s^2');
```

```
dfile = set(dfile,'chname',3,'Y');  
dfile = set(dfile,'chfilt',3,filt);  
dfile = set(dfile,'chunits',3,'m/s^2');  
dfile = set(dfile,'chname',4,'Z');  
dfile = set(dfile,'chfilt',4,filt);  
dfile = set(dfile,'chunits',4,'m/s^2');  
return;
```

Alternatively, you can use `ap_view` (see page 13)

apview: an interactive interface to AnyPlot

apview provides an easy way to look at time series data. You can zoom in on data, filter, calculate spectra and coherences and more using a graphical interface. Even when you become used to using the more powerful command line functions, you may still want to use apview for a first look at your data. There are three ways to call apview():

apview()

The simplest way to call apview is without any arguments. You will then be prompted for what kind of data you want to read, and then for datafile information. Afterwards, it will act like apview(datafile).

apview(datafile)

If you provide an ap_datafile object as the argument to apview, it will read in the data itself. You can use the “Record” button to go backwards and forwards in the file.

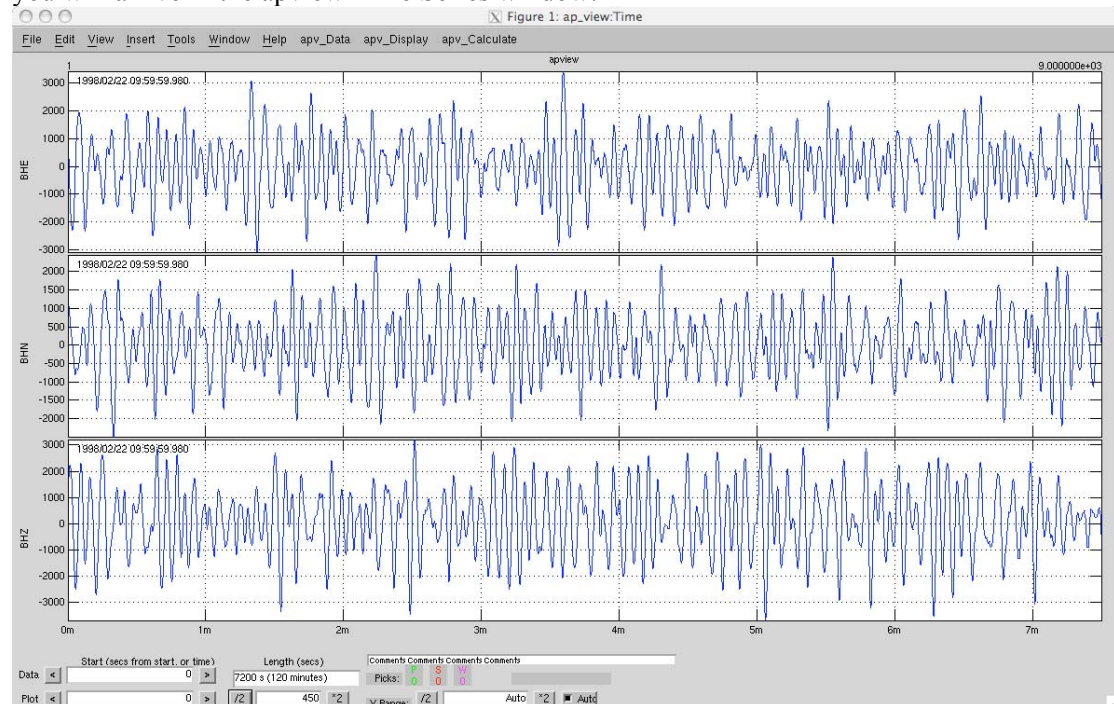
ap_view(tdata)

If you first create an ap_timedata object (tdata for this example), then type:

➤ `apview(tdata)`

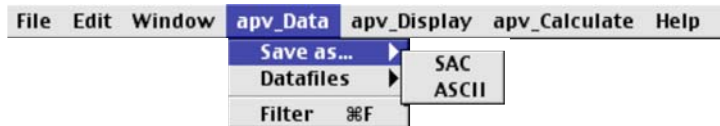
you will be able to look around in the existing data, but not to read other data.

For the latter two calls, or for the first call after you have specified the datafile information, you will arrive in the apview Time Series window:

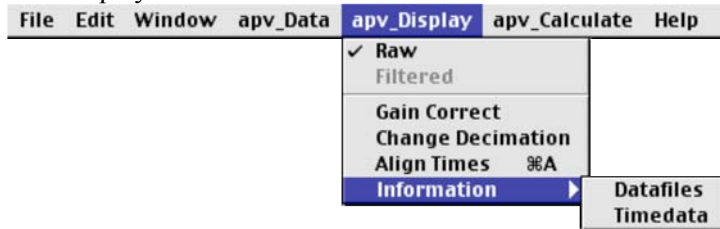


You can click and drag on one of the traces to zoom in, or use the address and segment length commands to select a part of the data and a display length. The segment length edit field understands “d” (for days), “h” (for hours), “m” (for minutes) and “s” for seconds, so you can type “10m” and ap_view will calculate the segment length to show.

There are also three menu items available:
The Data Menu:



The Display Menu:



and the Calculate Menu:



If you want to calculate spectra from your data, select the apv_Calculate->Spectra menu and fill in the form that pops up:

Start address (>=1)

1

End address (<=83000)

83000

Start record (>=1)

1

End record

1

Window size (power of 2)

4096

Window type ('prol1pi','prol4pi')

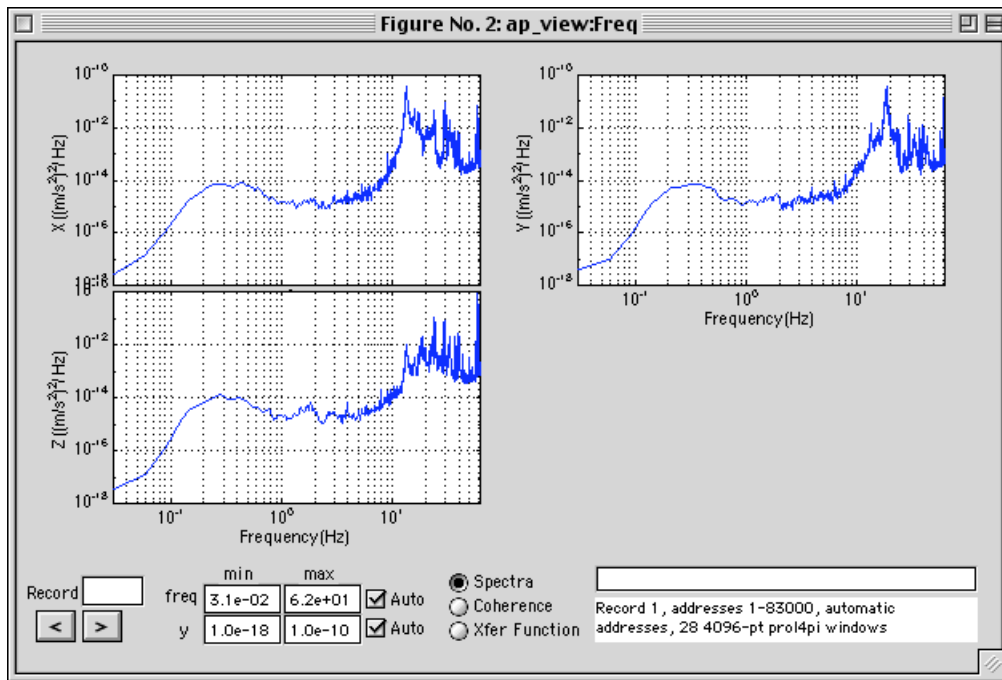
prol4pi

Allow further control over rec/address

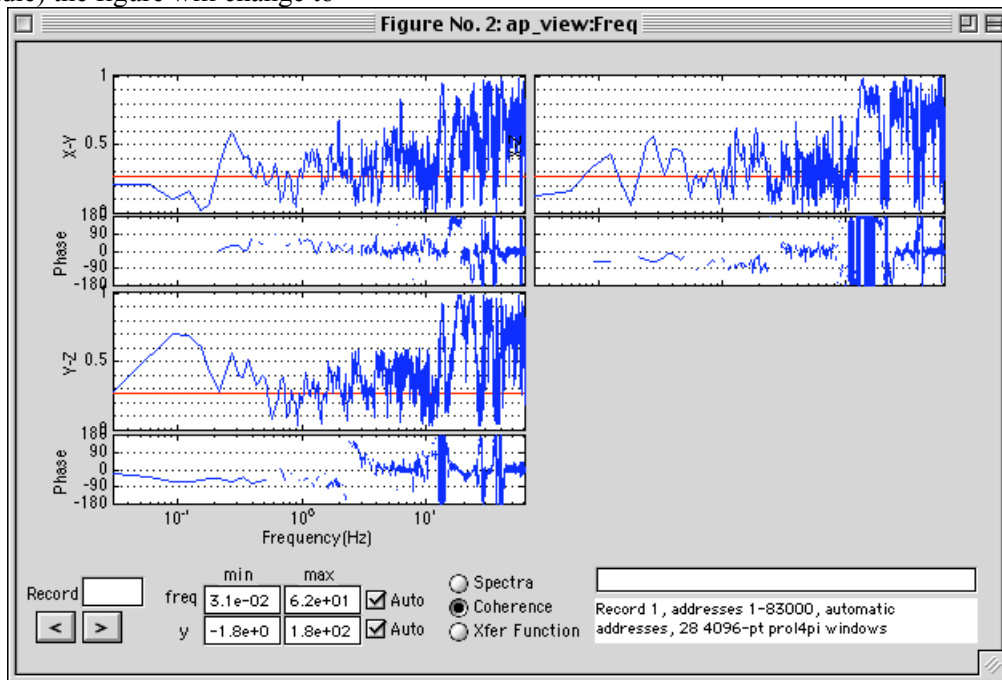
No

Cancel OK

The spectra will be calculated and a new display window will open up:



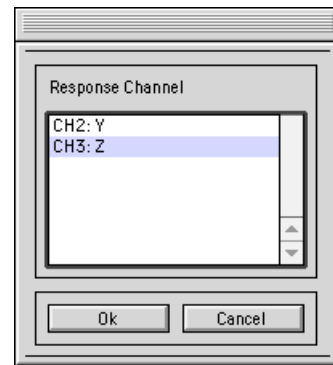
The coherence is already calculated, so if you then click on the “Coherence” buttonj (bottom middle) the figure will change to



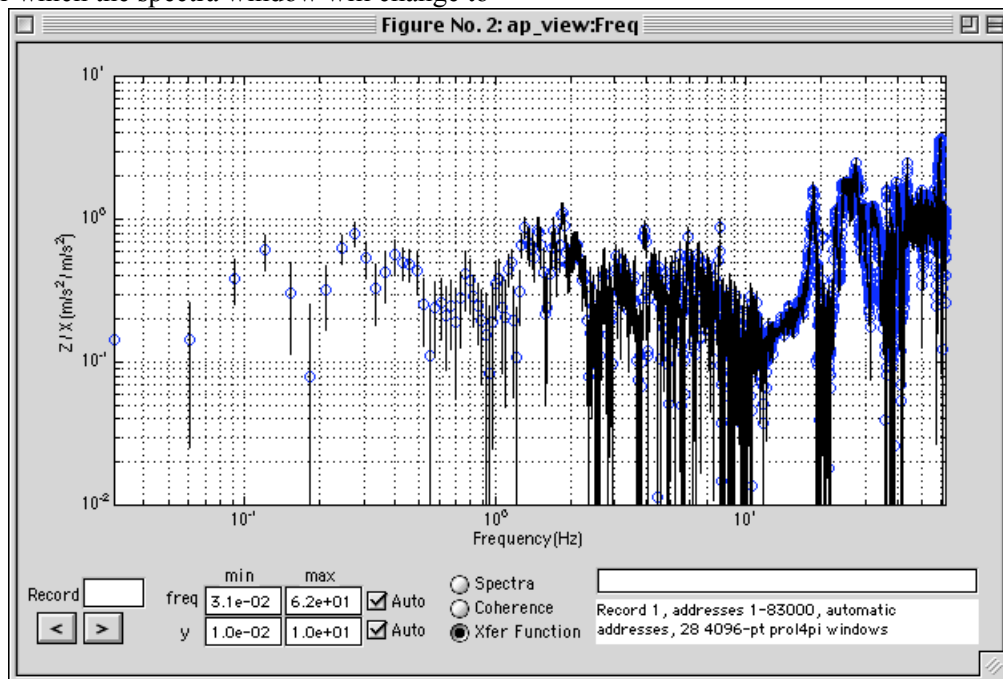
(not a very good coherence in this example!). If you click on the “Xfer Function” button, you will have to fill in two dialogs to specify the driving and response channels:



then



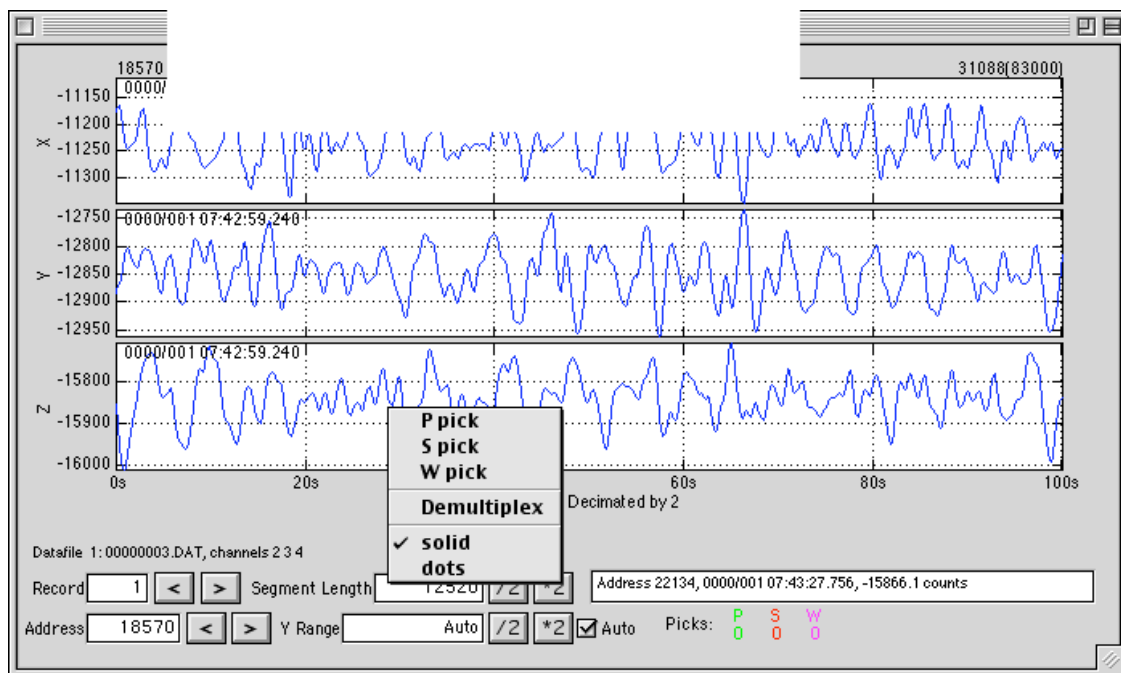
after which the spectra window will change to



This isn't a great example, because the coherences are pretty bad. Note that the transfer function assumes that all non-correlated noise is on the response channel (if you use `ap_xferfun()` directly you can specify what assumptions to make).

Back to the time data, if you want to filter the time data, select the `apv_Data->Filter` menu. You will fill in the following form

, the data will be filtered and the time display will show the filtered data:



You can shift back and forth between viewing the unfiltered and filtered data using the `apv_Display` menu. Note that there is also a menu in the middle of the screen. This is a contextual menu that pops up if you hold down the key while clicking on one of the traces. Using this, you can display the data as unconnected dots (useful if the channel has several multiplexed data, demultiplex the data (showing each subchannel by an individual line) or pick P,S and W arrivals (this last is still buggy).

AnyPlot Function Reference

Summary:

| Class | function | description |
|-------------|--|---|
| none | ap_set | set global AnyPlot behaviours |
| ap_timedata | ap_timedata calcTime cat cut extract filter findTime get merge plot plotfilt resample set subtract_xf | create ap_timedata object calculate the time for a given address concatenate ap_timedata objects extract a section from the data extract selected channels filter data calculate the address for a given time get variable value merge ap_timedata objects plot time series data plot the filter responses of the recording devices change the data sampling rate set variable value subtract the effect of one channel from another |
| ap_spectra | ap_spectra get plot plotcoher set | create ap_spectra object get variable value plot spectra plot coherences set variable value |
| ap_xferfun | ap_xferfun get merge plot set | create ap_xferfun object get variable value merge ap_xferfun objects plot the transfer function(s) set variable value |
| ap_datafile | ap_datafile get set | create ap_datafile object get variable value set variable value |
| ap_filtparm | ap_filtparm * plot response | create ap_filtparm object multiply filter parameters plot the filter response output the filter response |
| ap_plotinfo | ap_plotinfo calcplotpos get set | create ap_plotinfo object calculate the subplot position get variable data set variable data |
| ap_time | ap_time - + | create ap_time object subtract one time from another, or seconds from a time add seconds to a time |

ap_set

Description

Set Anyplot global behaviours. Should there be an ap_get, too?

Calls

```
ap_set(field,value);
```

Arguments

| | |
|-------|---------------------|
| field | the field to change |
| value | the new value |

fields:

'verbose'

'blah'

values:

'dialog'

'command'

'all'

'none'

Blah

Print info in dialog box(es) only

Print info in command window only

Print in command window and put up dialog boxes

Do not print any info

Return value

None

Notes

For now, just an idea. Not implemented.

ap_timedata functions

ap_timedata

Description

Creates an ap_timedata object.

Calls

```
dfile = ap_timedata(sampfreq,data);  
dfile = ap_timedata(datafile,data);  
dfile = ap_timedata(datafile,data,readchans);
```

Arguments

| | |
|-----------|--|
| data | the data matrix. Each row is treated as a separate channel |
| sampfreq | the data sampling frequency |
| datafile | an ap_datafile object. Must have the same number of channels as the data matrix has rows. Channel 1 corresponds to the first row, etc. |
| readchans | array of which rows of the data matrix to read and in what order. The channel information in datafile is identically sorted/shuffled. |

Return value

The object

Notes

To get the data, type:

```
var = tdata.data;  
var = tdata.data(CHANNELS,ADDRESSES);
```

To set the data, type:

```
tdata.data = var;  
tdata.data(CHANNELSS,ADDRESSES) =var;
```

where var is a matlab data array with one row for each data channel, and CHANNELS and ADDRESSES are scalars, vectors (e.g., 1:1000, 1:end, [1 2 5]) or :, meaning all channels or addresses. The word end specifies the last channel or address.

Should there be a function to read ap_datafile info into an ap_timedata channel?

calcAddress

Description

Calculates the address corresponding to the given time. Corrects for the clock drift

Calls

```
address = calcAddress(tdata,ich,time);
```

Arguments

| | | |
|-------|-------------|--|
| tdata | ap_timedata | the data |
| ich | scalar | a channel # (each channel can have a different start time and clock drift) |
| time | ap_time | the time to search for |

Return value

| | | |
|---------|--------|---|
| address | scalar | the closest address to the requested time |
|---------|--------|---|

Notes

calcNumSamps

Description

Calculates the number of samples corresponding to a time string

Calls

```
numSamps = calcNumSamps(tdata,elapsedTimeStr);
```

Arguments

| | | |
|---------|-------------|---|
| tdata | ap_timedata | the data |
| timeStr | char | a number terminated by 's' (seconds), 'm' (minutes), 'h' (hours) or 'd' (days). If there is no termination, the value is assumed to already be in samples |

Return value

| | |
|----------|---|
| numSamps | the number of samples corresponding to the specified elapsed time |
|----------|---|

Notes

calcTime

Description

calculates the time corresponding to an address for the given channel number. Corrects for the clock drift.

Calls

```
time = calcTime(tdata,channel#,address);
```

Arguments

| | |
|----------|-----------------------|
| tdata | an ap_timedata object |
| channel# | the channel number |
| address | the address |

Return value

| | |
|------|---------------------------------------|
| time | an ap_time object containing the time |
|------|---------------------------------------|

Notes

`time_calculate(tdata,channel#,1)` is equivalent to
`get(ap_timedata,'chstarttime',channel#)`

cat

Description

Concatenates the data in two or more ap_timedata objects

Calls

```
tdata = cat(tdata1,tdata2,...);
```

Arguments

| | |
|--------|----------------------------|
| tdata1 | an ap_timedata object |
| tdata2 | another ap_timedata object |
| ... | more ap_timedata objects |

Return value

A new ap_timedata object with the parameters (starttime, filters, etc) from tdata1, and with the concatenated data from all the ap_timedata objects.

Notes

Aborts if the objects don't have the same number of channels and the same sampling rate.
Prints a warning if the data are not consecutive in time.

cut

Description

creates a new ap_timedata object containing a subset of the original data

Calls

```
tdata = function(tdata_in,startaddress,endaddress);
```

Arguments

| | |
|--------------|------------------------|
| tdata_in | ap_timedata object |
| startaddress | start address (scalar) |
| endaddress | end address (scalar) |

Return value

A new ap_timedata object

extract

Description

Create a new ap_timedata object with selected channels from the original object

Calls

```
tdata = function(tdata_in,channels);
```

Arguments

| | |
|----------|-----------------------------|
| tdata_in | an ap_timedata object |
| channels | channels to extract (array) |

Return value

A new ap_timedata object

filter

Description

Filter data in an ap_timedata object

Calls

```
tdata = filter(tdata_in, filtttype, filtparms);
```

Arguments

| | |
|-----------|--|
| tdata_in | an ap_timedata object |
| filtttype | The type of filter to use: 'BwHigh' Highpass Butterworth filter 'BwLow' Lowpass Butterworth filter 'BwPass' Bandpass Butterworth filter 'BwStop' Bandstop Butterworth filter |
| filtparms | The filter parameters. If filtttype is 'BwHigh' or 'BwLow', filtparms = {npoles, cfreq} npoles number of poles cfreq corner frequency If filtttype is 'BwPass' or 'BwStop', filtparms = {npoles, lfreq, hfreq} npoles number of poles lfreq corner frequency hfreq corner frequency |

Return value

A new ap_timedata object

Examples

```
tdata = filter(tdata_in, 'BfPass', {4, .001, .1}); filters the data using a 4-pole  
Butterworth filter with a passband between 0.001 and 0.1 Hz.
```

Notes

Adds information about the new filter poles and zeros to the new object. I think there's a bug in this, because the spectra calculated before and after filtering are significantly different.

get

Description

Get a field from the `ap_timedata` object. If the field name starts with 'ch', gets the field from the specified channel#. If no field is specified, print out all the fields.

Calls

```
outp = get(tdata,field)
outp = get(tdata,field,channel#)
get(tdata)
```

Arguments

| | |
|-----------------------|--|
| <code>tdata</code> | an <code>ap_timedata</code> object |
| <code>channel#</code> | the channel # (only valid if field starts with 'ch') |
| <code>field</code> | the field to get |

Return value

Depends on field:

| <u>if field is</u> | <u>the return value is</u> |
|--------------------|---|
| 'chclockdrift' | Clock drift for the specified channel (dimensionless, positive if the instrument clock is fast) |
| 'chfilt' | Instrument response of the specified channel (<code>ap_filtparm</code> object) |
| 'chinstname' | Name of the instrument that this channel came from |
| 'chinstnum' | Number of the instrument that this channel comes from |
| 'chinstloc' | Location of the instrument that this channel comes from: {lat, long, elev, wdepth} lat Instrument latitude in decimal degrees N long Instrument longitude in decimal degrees E elev Instrument elevation in meters wdepth Water depth in meters at the instrument |
| 'chname' | The name of the specified channel |
| 'chorigchnum' | Channel number of the specified channel in the original datafile |
| 'chstarttime' | Time at address 1 of the specified channel: <code>ap_time</code> object |
| 'chunits' | Data units of the specified channel when corrected for filter response |
| 'datalength' | The number of samples in each channel |
| 'eventtime' | Time of some event within the data (<code>ap_time</code> object) |
| 'eventloc' | Event location: {lat, long, depth, wdepth} lat Event latitude in decimal degrees N long Event longitude in decimal degrees E depth Event depth in meters wdepth Water depth in meters |
| 'numchans' | Number of data channels |
| 'plotinfo' | information about how to plot the data (<code>ap_plotinfo</code> object) |
| 'sampfreq' | Sampling frequency (Hz) |
| 'recnum' | The record number (only meaningful for data from files containing consecutive readable records) |

Notes

merge

Description

Creates a new `ap_timedata` object with as many channels as the combined number of channels in the original object

Calls

```
tdata = merge(tdata1,tdata2,...);
```

Arguments

| | |
|---------------------|---|
| <code>tdata1</code> | an <code>ap_timedata</code> object |
| <code>tdata2</code> | another <code>ap_timedata</code> object |
| <code>...</code> | more <code>ap_timedata</code> objects |

Return value

A new `ap_timedata` object with the event info from `tdata1`, but the start times, filter parameters for each channel from the original objects.

Notes

Aborts if the sampling frequency or the number of samples is different in any of the `ap_timedata` objects.

plot

Description

Plots the data in an ap_timedata object. By default, plots all of the data

Calls

```
[ha,h1] = plot(tdata,...);
```

Arguments

| | |
|-----------------|--|
| tdata | an ap_timedata object |
| | fields and values affecting the plot. If field starts with 'ch', then the next two arguments are channel#, value. If not, the next argument is value. If channel#=0, change all the channels to the given value. You can append as many fields and values as you want. |
| field | value |
| 'startaddr' | Address to start plotting at (default 1). The same as "chstartaddr", with the channel set to 0, but it is called so often this way that AnyPlot allows this shortcut. |
| 'seglen' | Segment length to plot (default=0: the rest of the data). If a scalar, plot that many data points. If a string, calculate how many data points to plot according to the letter at the end of the string: 's'=seconds, 'm'=minutes, 'h'=hours, 'd'=days. |
| 'aligntraces' | Boolean: Align time plots so they all start at the same time as the channel 1 trace (useful for channels with different start times or different clock drifts) |
| 'overlay' | Plot all the plots on one axis, using the channel 1 plot parameters? (default=0). Not yet implemented. |
| 'decimtype' | The type of decimation used when plotting time data (default 'smart'): 'none' no decimation 'regular' plot every plotregdecim points 'smart' decimate so that no more than plotsmartdecim points are plotted |
| 'smartdecim' | Maximum number of points to plot when using smart decimation (default 5000) |
| 'regdecim' | Number of points to decimate by when using regular decimation (default 16) |
| 'chdemultiplex' | Boolean: If 0, data are plotted normally, if not, then data are treated like N different multiplexed channels and plotted as separate lines on the same figure. (default 0) |
| 'chfontsize' | Font size used (default 10) |
| 'chgrid' | Boolean: place grid behind data (default 1) |
| 'chlinetype' | Line style and color used to plot the data (default '-') |
| 'chmarker' | Marker used to plot the data (default none) |
| 'chremovegain' | Boolean: Divide data by the instrument gain in the passband before plotting. Doesn't correct for poles and zeros, just for the bandpass gain value. (default 0) |
| 'chstartaddr' | Same as startaddr, but for an individual channel |
| 'chymin' | Minimum y-axis value. If NaN, set ymin as the mean data value minus chplotyrange/2 |
| 'chyrange' | Y-axis range. If NaN, automatically calculate the y range based on the data. |
| 'highlight' | Allows you to highlight sections of the data: {hilite1, hilite2, ...}, where hiliten = {chnum,saddr,seglen,text,stroke,fill }: |

| | |
|--------|---|
| chnum | Channel # to place the highlight (can be one channel, an array of channels, or ':' meaning all channels) |
| saddr | Starting address of the highlight |
| seglen | Length of the highlight. Can be in data samples, seconds, minutes, hours, or days. For all but the first, write the length as 'XXXXt', where XXXX is the length and t=='s' means seconds, t=='m' means minutes, t=='h' means hours, and t=='d' means days |
| text | Text to place at the upper right corner of the highlighted region [default=""] |
| stroke | Type of line to mark the boundaries of the highlight [default = '-'] |
| fill | Fill color of the highlight [default = no fill] |

Examples

To plot 5 minutes of data in tdata, starting at address 1000, type

```
plot(tdata, 'startaddress', 1000, 'seglen', '5m');
```

To plot 1 hour of data starting at 1997/132 04:45:00, type:

```
plot(tdata, ...
      'startaddress', time_locate(tdata, 1, '1997/132 4:45:00'), ...
      'plotaligntraces', 1, ...
      'seglen', '1h');
```

Return value

| | |
|----|--|
| ha | array of handles to all the subplot axes |
| hl | array of handles to all the lines |

plotfilt

Description

Plots the instrument frequency response for each channel in an ap_timedata object

Calls

```
plotfilt(tdata);  
plotfilt(tdata, freqs);
```

Arguments

| | |
|-------|---|
| tdata | an ap_tdata object |
| freqs | the frequencies of interest. If not specified, plots from one-thousandth the Nyquist frequency to the Nyquist frequency |

Return value

None

resample

Description

Resamples `ap_timedata` data at a different frequency

Calls

```
tdata = function(tdata_in,average);  
tdata = function(tdata_in,newsampfreq);  
tdata = function(tdata_in,newsampfreq,interpmethod);
```

Arguments

| | |
|---------------------------|---|
| <code>tdata_in</code> | an <code>ap_timedata</code> object |
| <code>average</code> | resample by averaging over a number of samples. The form is <code>'N'</code> , where <code>N</code> is the number of samples to average. Throws away the residual points and prints a warning message if the original number of samples isn't an integer multiple of <code>N</code> . |
| <code>newsampfreq</code> | the new sampling frequency. Data are interpolated at the new frequency. If the new sampling frequency is lower than the old one, the data are 2-pole causal Butterworth lowpass filtered at <code>newsampfreq/2</code> before interpolating. |
| <code>interpmethod</code> | the interpolation method [default <code>'linear'</code>] <code>'linear'</code> linear interpolation <code>'nearest'</code> nearest neighbor interpolation <code>'cubic'</code> cubic interpolation <code>'spline'</code> cubic spline interpolation |

Return value

A new `ap_timedata` object, sampled at the requested frequency

Examples

To resample at 4 Hz, linear interpolation:

```
outdata = resample(tdata,4);
```

To resample at 4 Hz, cubic spline interpolation:

```
outdata = resample(tdata,4,'spline');
```

To resample at 1/4 the original frequency by sampling:

```
outdata = resample(tdata,'/4');
```

Notes

set

Description

Set a field in an `ap_datafile` object. If the field name starts with 'ch', sets the field in the specified channel#. If no field is specified, print out all possible set values.

Calls

```
tdata = set(tdata_in,field,value)
tdata = set(tdata_in,field,channel#,value)
get(tdata_in)
```

Arguments

| | |
|-----------------------|--|
| <code>tdata</code> | an <code>ap_timedata</code> object |
| <code>field</code> | the field to set |
| <code>channel#</code> | the channel # (only valid if field starts with 'ch'). If =0, change all the channels to the given value. |
| <code>value</code> | the new field value |

fields:

'chfilt'

values:

Instrument response of the specified channel: `ap_filtparm` object
OR {`gain`,`poles`,`zeros`}

| | |
|--------------------|------------------------|
| <code>gain</code> | Pole-zero gain |
| <code>poles</code> | frequency-domain Poles |
| <code>zeros</code> | frequency-domain Zeros |

'chfiltRC'

Optional method for specifying instrument response using RC circuit values: {`gain`,`lpRC`, `hpRC`}

| | |
|-------------------|---|
| <code>gain</code> | RC circuit gain (NOT the same as for poles & zeros) |
| <code>lpRC</code> | lowpass RC values |
| <code>hpRC</code> | highpass RC values (equivalent to e-folding time) |

'chfiltF'

Optional method for specifying instrument response using Butterworth filter corner frequencies: {`gain`,`lpF`, `hpF`}

| | |
|-------------------|--|
| <code>gain</code> | passband gain (same as RC circuit gain) |
| <code>lpF</code> | lowpass corner frequencies ($2\pi \cdot \text{lpRC}$) |
| <code>hpF</code> | highpass corner frequencies ($2\pi \cdot \text{hpRC}$) |

'chfiltT'

Optional method for specifying instrument response using Butterworth filter corner periods: {`gain`,`lpT`, `hpT`}

| | |
|-------------------|--|
| <code>gain</code> | passband gain (same as RC circuit gain) |
| <code>lpT</code> | lowpass corner periods ($1/(2\pi \cdot \text{lpRC})$) |
| <code>hpT</code> | highpass corner periods ($1/(2\pi \cdot \text{hpRC})$) |

'chname'

The name of the specified channel

'chstarttime'

Time at address 1 of the specified channel: `ap_time` object OR {`yr`, `mon`, `day`, `hr`, `min`, `sec`} OR {`yr`, `jday`, `hr`, `min`, `sec`}

| | |
|-------------------|--------------|
| <code>yr</code> | Year |
| <code>mon</code> | Month |
| <code>day</code> | Day of Month |
| <code>jday</code> | Julian Day |
| <code>hr</code> | Hour |
| <code>min</code> | Minute |
| <code>sec</code> | Seconds |

'chunits'

Data units of the specified channel when corrected for filter response

'eventtime'

Time of some event within the data. `ap_time` object or {`yr`, `mon`, `day`, `hr`, `min`, `sec`} or {`yr`, `jday`, `hr`, `min`, `sec`}

| | | |
|---------------|---|--------------------------------------|
| | yr | Year |
| | mon | Month |
| | day | Day of Month |
| | jday | Julian Day |
| | hr | Hour |
| | min | Minute |
| | sec | Seconds |
| 'eventloc' | Event location: {lat, long, depth, wdepth} | |
| | lat | Event latitude in decimal degrees N |
| | long | Event longitude in decimal degrees E |
| | depth | Event depth in meters |
| | wdepth | Water depth in meters |
| 'sampfreq' | Sampling frequency (Hz) | |
| 'recnum' | The record number (only meaningful for data from files containing consecutive readable records) | |
| 'plotinfo' | Plot information variables (ap_plotinfo object) | |
| plot() fields | All of the field, value pairs described for plot() can also be modified here. The difference is that fields set in plot() only affect that plot, while fields set here affect all subsequent plots. | |

Return value

The modified ap_datafile object

Notes

To change an ap_timedata object, specify the same input and output object

A lot of instrument info is lacking here. How do I read it into ap_timedata in the first place"?

subtract_xf

Description

Subtracts the effect of “driving” channels from “response” channels calculated using ap_xferfun

Calls

```
tdata = subtract_xf(tdata_in,xferfunc);
```

Arguments

| | |
|----------|---|
| tdata_in | an ap_timedata object |
| xferfun | an ap_xferfun object calculated from tdata_in |

Return value

| | |
|-------|-----------------------|
| tdata | an ap_timedata object |
|-------|-----------------------|

Notes

ap_spectra functions

ap_spectra

Description

Creates an ap_spectra object by calculating the spectra of one or more ap_timedata objects.

Calls

```
spectra = ap_spectra(tdata, ...);  
[spectra,raw] = ap_spectra(tdata, ...);
```

Arguments

| | |
|----------------|---|
| tdata | an ap_timedata object |
| ... | field, value pairs: |
| <u>fields:</u> | <u>values:</u> |
| 'windowsize' | Number of samples in each fft window (default = 2048). Must be a power of 2. |
| 'taper' | Taper to apply to each data window (default = 'prol4pi'): 'prol4pi' 4-pi prolate spheroidal taper. A tight taper that allows windows to be overlapped by 30% of the windowsize. Excellent (120 dB) broad-band rejection, but low (4-bin) frequency resolution. 'prol1pi' 1-pi prolate spheroidal taper. High frequency resolution (approximately 1 bin), but weak (55 dB) broad-band rejection. |
| 'startaddress' | First address in ap_timedata to calculate spectra from (default=1) |
| 'seglen' | Number of data samples to use for calculating the spectra (default = to the end of the data) |
| 'addrlist' | An explicit list of addresses at which to calculate ffts. Overrides the 'startaddress' and 'seglen' values. |
| 'badaddrs' | A list of addresses to avoid when calculating spectra. Write as a text string in the Matlab address format (e.g. '10:50 3000:4000 40000:end'). |
| 'badbuff' | A scalar value to extend the addresses to avoid in 'badaddrs'. Useful if you want to use the same badaddrs after you filter the data. |
| 'subtractxf' | An ap_xferfun object. Used to subtract the effect of one channel on another. |
| 'rawspect' | A raw spectral structure to be added to the spectra currently calculated. In this manner, you can calculate spectra for data stored in multiple files. |

Return value

| | |
|---------|---|
| spectra | an ap_spectra object |
| raw | a "raw" (unaveraged) version of the spectra and coherences, which can be used to calculate spectra and coherences across multiple ap_timedata objects |

Examples

To calculate a 4096-point spectra from data in tdata1, tdata2, and tdata3, avoiding a data spike at addresses 4000 to 4010 in tdata1:

```
[sdata,raw] = ap_spectra(tdata1,'windowsize',4096,'badaddrs','4000:4010');  
[sdata,raw] = ap_spectra(tdata2,'windowsize',4096,'rawspect',raw);  
sdata = ap_spectra(tdata3,'windowsize',4096,'rawspect',raw);
```

Notes

To get the spectra, coherences, and frequencies:

```
var= sdata.spect;
```

```
var= sdata.spect(CH,ADDRS);  
var= sdata.coh;  
var= sdata.coh(CH1,CH2,ADDRS);  
var= sdata.freq;  
var= sdata.freq(ADDRS);
```

where CHANNELS and ADDRESSES are scalars, vectors (e.g., 1:1000, 1:end, [1 2 5]) or :, meaning all channels or addresses. The word end specifies the last channel or address. You can only "set" the data by running ap_spectra() again.

get

Description

Get a field from the `ap_spectra` object. If the field name starts with 'ch', gets the field from the specified channel#. If no field is specified, print out all the fields.

Calls

```
outp = get(sdata,field)
outp = get(sdata,field,channel#)
get(sdata)
```

Arguments

| | |
|-----------------------|--|
| <code>sdata</code> | an <code>ap_spectra</code> object |
| <code>channel#</code> | the channel # (only valid if field starts with 'ch') |
| <code>field</code> | the field to get |

Return value

Depends on field:

| <u>if field is</u> | <u>the return value is</u> |
|--------------------|--|
| 'winsize' | The data window size used to calculate the spectra |
| 'numfreqs' | The number of frequencies in the spectra (should be winsize/2) |
| 'addrlist' | A list of the startaddresses of each window used to calculate an FFT |
| 'numwinds' | The number of data windows used to calculate the spectra |
| 'cohsigniflevel' | The 95% significance level for the coherence amplitude |
| 'numchans' | The number of spectra channels |
| 'chname' | The name of the specified channel (same as in <code>ap_timedata</code>) |
| 'chunits' | The power spectral units for the specified channel |
| 'chorigunits' | The original data units of the specified channel |
| 'plotinfo' | the <code>ap_plotinfo</code> object |
| plotinfo fields | the plotting field value (see <code>plot()</code>) |

Notes

plot

Description

Plots the spectra in an ap_spectra object.

Calls

```
[ha,hl] = plot(sdata,...);
```

Arguments

| | |
|----------------|--|
| sdata | an ap_spectra object |
| | fields and values affecting the plot. If field starts with 'ch', then the next two arguments are channel#, value. If not, the next argument is value. If channel#=0, change all the channels to the given value. You can append as many fields and values as you want. |
| field | value |
| 'bottomoffset' | How many pixels up from the bottom of the figure to place the bottommost plot (default=0) |
| 'fontsize' | Font size used (default 10) |
| 'overlay' | Plot all the plots on one axis, using the channel 1 plot parameters? (default=0). Not yet implemented. |
| 'yscale' | y axis scale: 'default' logarithmic scale. 'log' logarithmic scale. 'linear' linear scale. 'dB' decibal scale. |
| 'chxmax' | Maximum frequency to plot. If NaN use maximum frequency. |
| 'chymin' | Minimum y-axis value to plot. If NaN, set ymin as the power of ten beneath the minumum data value |
| 'chymax' | Minimum y-axis value to plot. If NaN, set ymin as the power of ten above the minumum data value |
| 'chgrid' | Boolean: plot a grid beneath the data? |
| 'chlinetype' | Line style and color to plot the data (default '-') |
| 'chmarker' | Marker used to plot the data (default none) |

Return value

| | |
|----|--|
| ha | array of handles to all the subplot axes |
| hl | array of handles to all the lines |

plotcoher

Description

Plots the coherences in an ap_spectra object.

Calls

```
[ha,hl] = plot(sdata,...);
```

Arguments

| | |
|----------------|--|
| tdata | an ap_timedata object |
| | fields and values affecting the plot. If field starts with 'ch', then the next two arguments are channel#, value. If not, the next argument is value. If channel#=0, change all the channels to the given value. You can append as many fields and values as you want. |
| field | value |
| 'chxmin' | Minimum frequency to plot. If NaN, use minimum frequency |
| 'chxmax' | Maximum frequency to plot. If NaN use maximum frequency. |
| 'bottomoffset' | How many pixels up from the bottom of the figure to place the bottommost plot (default=0) |
| 'overlay' | Plot all the plots on one axis, using the channel 1 plot parameters? (default=0). Not yet implemented. |
| 'fontsize' | Font size used (default 10) |

Return value

| | |
|----|--|
| ha | array of handles to all the subplot axes |
| hl | array of handles to all the lines |

set

Description

Set a plotting field in an ap_spectra object.

Calls

```
sdata = set(sdata_in,field, value);  
sdata = set(sdata_in,field, channel#,value);
```

Arguments

| | |
|----------|---|
| sdata | an ap_spectra object |
| field | the field to set. All of the field, value pairs described for plot() can also be modified here. The difference is that fields set in plot() only affect that plot, while fields set here affect all subsequent plots. |
| channel# | the channel # (only valid if field starts with 'ch') |
| value | the new field value |

Return value

The modified ap_spectra object

Notes

You can't change the non-plotting fields in an ap_spectra object. You have to rerun ap_spectra().

ap_xferfun functions

ap_xferfun

Description

Creates an ap_xferfun object by calculating the transfer function between two ap_spectra channels.

Calls

```
xdata = ap_xferfun(xdata,drivechan,respchan, ...);
```

Arguments

| | |
|------------------|--|
| sdata | an ap_spectra object |
| drivechan | the driving channel number |
| respchan | the response channel number |
| ... | field, value pairs: |
| <u>fields:</u> | <u>values:</u> |
| 'noisechan' | 'response' [default] Assume that all noise is on the response channel |
| | 'driving' Assume that all noise is on the driving channel |
| | 'equal' Assume the same signal/noise on the driving and response channels |
| | 'unknown' Make no assumption about noise |
| 'setnoiselevels' | A cell array containing noise power spectral density (PSD) levels for the driving and response channels as follows: {[drivefreqs drivelevels],[respfreqs resplevels]} The noise PSD units are the same as the data PSDs Values are linearly interpolated in the log(PSD) domain |

Return value

| | |
|---------|---|
| spectra | an ap_spectra object |
| raw | a "raw" (unaveraged) version of the spectra and coherences, which can be used to calculate spectra and coherences across multiple ap_timedata objects |

Examples

Notes

To get the transfer function, standard error, "good" data boolean, and frequencies, type

```
var= xdata.data;  
var= xdata.data(CH,ADDRS);  
var= xdata.err;  
var= xdata.err(CH,ADDRS);  
var= xdata.gooddata;  
var= xdata.err(CH,ADDRS);  
var= xdata.freq;  
var= xdata.freq(ADDRS);
```

where CHANNELS and ADDRESSES are scalars, vectors (e.g., 1:1000, 1:end, [1 2 5]) or :, meaning all channels or addresses. The word end specifies the last channel or address.

Initially, gooddata=1 at all indices where the coherence was greater than the 95% significant level, and zero at the rest. ap_spectra(tdata,'subtractxf',xdata,...) only subtracts the transfer functions at indices where gooddata is not 0. You can reset which values are "good" by typing:

```
xdata.gooddata = var;
```

```
xdata.gooddata(CHANNELS,ADDRESSES) = var;
```

var must be the same size as the data in xdata, or as the specified CHANNELS and ADDRESSES
You can only "set" the other data by running ap_xferfun() again.
ap_xferfun() only calculates one "channel" of transfer function data, but you can merge()
different ap_xferfun objects together to create multiple channels.

Confirm that it is the the standard error that I calculate

get

Description

Get a field from the `ap_xferfun` object. If the field name starts with 'ch', gets the field from the specified channel#. If no field is specified, print out all the fields.

Calls

```
outp = get(xdata,field)
outp = get(xdata,field,channel#)
get(xdata)
```

Arguments

| | |
|-----------------------|--|
| <code>xdata</code> | an <code>ap_xferfun</code> object |
| <code>channel#</code> | the channel # (only valid if field starts with 'ch') |
| <code>field</code> | the field to get |

Return value

Depends on field:

| <u>if field is</u> | <u>the return value is</u> |
|------------------------------|---|
| 'numchans' | The number of spectra channels |
| 'numfreqs' | The number of frequencies in the spectra (should be <code>windsize/2</code>) |
| 'chdrivechan' | The number of the driving channel |
| 'chrespchan' | The number of the response channel |
| 'chname' | The name of the specified channel |
| 'chunits' | The units of the transfer function for the specified channel |
| 'plotinfo' | <code>ap_plotinfo</code> object for the transfer functions |
| <code>plotinfo</code> fields | plotting values (see <code>plot()</code>) |

Notes

merge

Description

Creates a new `ap_xferfun` object with as many channels as the combined number of channels in the original objects

Calls

```
xdata = merge(xdata1,xdata2,...);
```

Arguments

| | |
|---------------------|---|
| <code>tdata1</code> | an <code>ap_timedata</code> object |
| <code>tdata2</code> | another <code>ap_timedata</code> object |
| <code>...</code> | more <code>ap_timedata</code> objects |

Return value

A new `ap_xferfun` object containing all the transfer function channels from the input objects.

Notes

Doesn't care if the transfer functions have different lengths or frequency arrays. It's probably not a good idea to mix them, though.

plot

Description

Plot the transfer function(s) in an ap_xferfun object.

Calls

```
[ha,hl] = plot(xdata,...);
```

Arguments

| | |
|--------------------|---|
| xdata | an ap_timedata object |
| | field, value pairs affecting the plot. |
| field | value |
| 'chplotxmin' | Minimum frequency to plot. If NaN, use minimum frequency |
| 'chplotxmax' | Maximum frequency to plot. If NaN use maximum frequency. |
| 'chplotymin' | Minimum y-axis value to plot. If NaN, set ymin as the power of ten beneath the minumum data value |
| 'chplotymax' | Minimum y-axis value to plot. If NaN, set ymin as the power of ten above the minumum data value |
| 'chplotgrid' | Boolean: plot a grid beneath the data? |
| 'plotbottomoffset' | How many pixels up from the bottom of the figure to place the bottommost plot (default=0) |
| 'plotoverlay' | Overlay the plots on one axis? (default=0). Not yet implemented. |
| 'chplotfontsize' | Font size used (default 10) |

Return value

| | |
|----|--|
| ha | array of handles to all the subplot axes |
| hl | array of handles to all the lines |

set

Description

Set a plotting field in an ap_xferfun object.

Calls

```
xdata = set(xdata_in,field, value);  
xdata = set(xdata_in,field, channel#,value);
```

Arguments

| | |
|----------|---|
| xdata_in | an ap_xferfun object |
| field | the field to set. All of the field, value pairs described for plot() can also be modified here. The difference is that fields set in plot() only affect that plot, while fields set here affect all subsequent plots. |
| channel# | the channel # (only valid if field starts with 'ch') |
| value | the new field value |

Return value

The modified ap_xferfun object

Notes

The only way to change the non-plotting fields in an ap_xferfun is to rerun ap_xferfun(...).

ap_datafile functions

ap_datafile

Description

Creates an ap_datafile object.

Calls

```
dfile = ap_datafile filetype,filename);
```

Arguments

| | |
|----------|------------------------------------|
| filetype | the file format ('WEBB','SAC',...) |
| filename | the filename |

Return value

The object

get

Description

Get a field from the `ap_datafile` object. If the field name starts with 'ch', get the field from the specified channel#. If no field is specified, print out all the fields.

Calls

```
outp = get(datafile,field)
outp = get(datafile,field,channel#)
get(datafile)
```

Arguments

| | |
|-----------------------|--|
| <code>datafile</code> | an <code>ap_datafile</code> object |
| <code>channel#</code> | the channel # (only valid if field starts with 'ch') |
| <code>field</code> | the field to get |

Return value

Depends on field:

| <u>if field is:</u> | <u>the return value is</u> |
|--|---|
| 'chname' | Name of the specified channel |
| 'chfilt' | Instrument response of the specified channel (<code>ap_filtparm</code> object) |
| 'chunits' | Data units of the specified channel when corrected for filter response |
| 'clockdrift' | The instrument clock drift (dimensionless, positive if the instrument clock is fast) |
| 'eventtime', 'eventloc', | Time of some event within the data (<code>ap_time</code> object) Event location: {lat, long, depth, wdepth} lat Event latitude in decimal degrees N long Event longitude in decimal degrees E depth Event depth in meters wdepth Water depth in meters |
| 'filename', 'filetype', 'instname', 'instnum', 'location', | the name of the datafile the datafile format (e.g., 'SAC','AH','ASCII') the instrument name The instrument number Instrument location: {lat, lon, elev, wdepth} lat Instrument latitude in decimal degrees N long Instrument longitude in decimal degrees E elev Instrument elevation in meters wdepth Water depth in meters at the instrument |
| 'numchans', 'pathname', 'sampfreq', 'starttime', | the number of channels in the data file the path to the datafile the instrument sampling frequency time at the beginning of the datafile (if there's an onboard counter, time at count 0) |
| 'RawParms' | Raw data file parameters: {rl,dl,bs,tba,tbs,twl,cr} rl record length: The length in bytes of one data record dl datalength: The number of samples per data record bs byteswapped: 1 means data are byteswapped, 0 means they aren't tba time byte addresses: Array of addresses of non-data blocks containing time bytes tbs time byte size: Size of these blocks |

twl time word length: The time word is this many bytes
 into these blocks
cr clockrate: for files containing time bytes, the
 number of counts/second
I should add a more complete description of the time blocks

save

Description

Saves ap_datafile parameters to a readable and executable M-file. Will prompt for the file name

Calls

save(dfile)

Arguments

dfile an ap_datafile object

set

Description

Set a field in an ap_datafile object. If the field name starts with 'ch', sets the field in the specified channel#. If no field is specified, print out all possible set values.

Calls

```
dfile = set(dfile_in,field,value)
dfile = set(dfile_in,field,channel#,value)
set(dfile_in)
```

Arguments

dfile_in an ap_datafile object
field the field to set
channel# the channel # (only valid if field starts with 'ch')
value the new field value

valid fields:

| | |
|-------------|---|
| 'chname' | channel name |
| 'chunits', | channel units, when corrected for filter response |
| 'chfilt' | channel filter response (ap_filtparm object) OR channel filter response in a cell array: gain poles zeros |
| 'chfiltRC', | channel filter response using RC circuit values in a cell array (note that the gain is NOT the same as for poles and zeros): gain gain lpRC array of lowpass filter RC values hpRC array of highpass filter RC values (same as e-folding time) |
| 'chfiltF', | channel filter response using Butterworth filter corner frequencies in a cell array (the gain is the same as for the RC circuit): gain gain lpfreqs array of lowpass filter corner frequencies (2pi*lpRC) hpfreqs array of highpass filter corner frequencies (2pi*hpRC) |
| 'chfiltT' | channel filter response using Butterworth filter corner periods in a cell array: gain gain lpT lowpass filter corner periods (1/(2pi*lpRC)) |

| | | |
|--------------|--|--|
| | hpT | highpass filter corner periods ($1/(2\pi \cdot \text{hpRC})$) |
| 'clockdrift' | The instrument clock drift (dimensionless, positive if the instrument clock is fast) | |
| 'eventtime', | the time of some event within the data: ap_time object OR {yr, mon, day, hr, min, sec} OR {yr, jday, hr, min, sec} | |
| | yr | Year |
| | mon | Month |
| | day | Day of Month |
| | jday | Julian Day |
| | hr | Hour |
| | min | Minute |
| | sec | Seconds |
| 'eventloc', | Event location: {lat, long, depth, wdepth} | |
| | lat | Event latitude in decimal degrees N |
| | long | Event longitude in decimal degrees E |
| | depth | Event depth in meters |
| | wdepth | Water depth in meters |
| 'filename', | the name of the datafile | |
| 'filetype', | the datafile format (e.g., 'SAC','AH','ASCII') | |
| 'instname' | the instrument name | |
| 'instnum', | The instrument number | |
| 'location', | Cell array containing the instrument location: | |
| | latitude | decimal degrees N |
| | longitude | decimal degrees E |
| | elevation | meters |
| | water depth | meters |
| 'numchans', | the number of channels in the data file | |
| 'pathname', | the path to the datafile | |
| 'sampfreq', | the instrument sampling frequency | |
| 'starttime', | time at the beginning of the datafile (if there's an onboard counter, time at count 0): ap_time object OR {yr, mon, day, hr, min, sec} OR {yr, jday, hr, min, sec} | |
| | yr | Year |
| | mon | Month |
| | day | Day of Month |
| | jday | Julian Day |
| | hr | Hour |
| | min | Minute |
| | sec | Seconds |
| 'RawParms' | Raw data file parameters: {r1,d1,bs,tba,tbs,twl,cr} | |
| | r1 | record length: The length in bytes of one data record |
| | d1 | datalength: The number of samples per data record |
| | bs | byteswapped: 1 means data are byteswapped, 0 means they aren't |
| | tba | time byte addresses: Array of addresses of non-data blocks containing time bytes |
| | tbs | time byte size: Size of these blocks |
| | twl | time word length: The time word is this many bytes into these blocks |
| | cr | clockrate: for files containing time bytes, the number of counts/second |
| | I should add a more complete description of the time blocks | |

Return value

The modified ap_datafile object

Notes

To change an ap_datafile object, specify the same input and output object

ap_filtparm functions

ap_filtparm

Description

Creates an ap_filtdata object, which contains filter information.

Calls

```
filtp = ap_filtdata();  
filtp = ap_filtdata(pzgain,poles,zeros);  
filtp = ap_filtdata(RCgain,lpRC,hpRC,'fromRC');  
filtp = ap_filtdata(RCgain,lpF,hpF,'fromBwFreqs');
```

Arguments

| | |
|--------|--|
| pzgain | pole-zero instrument gain |
| poles | array of S-plane poles evaluated in the frequency ($i\omega$) domain |
| zeros | array of S-plane zeros evaluated in the frequency ($i\omega$) domain |
| RCgain | instrument passband gain for an RC circuit or Butterworth filter |
| lpRC | array of lowpass RC values |
| hpRC | array of highpass RC values |
| lpF | array of lowpass Butterworth filter corner frequencies |
| hpRC | array of highpass Butterworth filter corner frequencies |

Return value

A new ap_filtdata object

Notes

ap_filtdata() without any arguments creates an object with gain = 1 and no poles or zeros

To access the filter gain, poles, and zeros, type:

```
var = filtparm.gain;  
var = filtparm.poles;  
var = filtparm.zeros;
```

Description

Combine (cascade) filter parameters. Multiplies gains and concatenates poles and zeros

Calls

```
filtp = filtp1 * filtp2;
```

plot

Description

plot filter parameters

Calls

```
plot(filtp);  
plot(filtp,freqs);  
plot(filtp,freqs,position);
```

Arguments

| | |
|----------|---|
| filtp | ap_filtparm object |
| freqs | frequencies to plot. If unspecified, will plot a range that shows all the changes in slope of the filter. |
| position | position on the figure to plot the filtparms. If unspecified, plots a full-sized axis. |

Return value

None

Notes

This is a very basic plotting function, and there are no ap_plotinfo parameters to modify.

response

Description

Calculate the complex filter response in the frequency domain

Calls

```
var = response(filtp,freqs);
```

Arguments

| | |
|-------|--|
| filtp | an ap_filtparm object |
| freqs | an array of frequencies at which to calculate the response |

Return value

The response in a complex array the same size as freqs

Notes

ap_plotinfo functions

ap_plotinfo

Description

Creates an `ap_plotinfo` object, which does little by itself but acts as plot variable storage for `ap_timedata`, `ap_spectra`, and `ap_xferfun` objects

Calls

```
plotp = ap_plotinfo(numchans,...);
```

Arguments

| | |
|----------|---|
| numchans | the number of subplots to create |
| ... | field, value pairs (see <code>ap_plotinfo:set()</code> for options) |

Return value

A new ap `plotinfoobject`

Notes

calcplopos

Description

Calculate where to place a subplot on a figure

Calls

```
[x,y,w,h] = calcplotpos(pinfo,numplots,plotnumber);
[x,y,w,h] = calcplotpos(pinfo,numplots,plotnumber,option);
```

Arguments

| | |
|-------------------------|--|
| <code>pinfo</code> | an <code>ap_plotinfo</code> object |
| <code>numplots</code> | the number of subplots to make |
| <code>plotnumber</code> | the current subplot number |
| <code>option</code> | string indicating a special way to organize the subplots |
| | <code>'onecolumn'</code> Stack the plots on one column, on top of each other |
| | <code>'onerow'</code> Stack the plots on one row next each other |

Return value

| | |
|-----|---|
| x,y | lower left corner of the subplot (pixels) |
| w | width of the subplot (pixels) |
| h | height of the subplot (pixels) |

Notes

You have to explicitly declare the number of plots just in case the number is set wrong in the `ap_plotinfo` object.

get

Description

Get a field from the `ap_plotinfo` object. If the field name starts with 'ch', gets the field from the specified channel. If no field is specified, print out all the fields.

Calls

```
outp = get(pinfo,field)
outp = get(pinfo,field,channel#)
get(pinfo)
```

Arguments

| | |
|-----------------------|--|
| <code>pinfo</code> | an <code>ap_plotinfo</code> object |
| <code>channel#</code> | the channel # (only valid if field starts with 'ch') |
| <code>field</code> | the field to get |

Return value

Depends on field, and the interpretation depends on what class is using the `ap_plotinfo` object. See `ap_plotinfo set()` for all the names and the `plot()` descriptions for `ap_timedata`, `ap_spectra`, and `ap_xferfun` for the interpretation:

Notes

set

Description

Set a field of an ap_plotinfo object. If the field name starts with 'ch', sets the field from the specified channel. If no field is specified, print out all the fields and their possible set values.

Calls

```
pinfo = set(pinfo_in,field,value)
pinfo = set(pinfo_in,field,channel#,value)
set(pinfo)
```

Arguments

| | |
|----------|--|
| pinfo | an ap_plotinfo object |
| field | the field to set |
| channel# | the channel # (only valid if field starts with 'ch') |
| value | the new field value |

| <u>fields:</u> | <u>used by</u> |
|---------------------|--|
| 'plotbottomoffset' | ap_timedata, ap_spectra, ap_xferfun (actually used by calcplotpos(), which is called in all of the above's plot() functions) |
| 'chplotfontsize' | ap_timedata, ap_spectra, ap_xferfun |
| 'chplotgrid' | ap_timedata, ap_spectra, ap_xferfun |
| 'chplotymin' | ap_timedata, ap_spectra, ap_xferfun |
| 'plotoverlay' | Should work for all. Not yet implemented. |
| 'plotaligntraces' | ap_timedata |
| 'plotdecimtype' | ap_timedata |
| 'plotsmartdecim' | ap_timedata |
| 'plotregdecim' | ap_timedata |
| 'chplotlinetype' | ap_timedata |
| 'chplotmarker' | ap_timedata |
| 'chplotstartaddr' | ap_timedata |
| 'chplotseglen' | ap_timedata |
| 'chplotyrange' | ap_timedata |
| 'chplotremovegain' | ap_timedata |
| 'chplotdemultiplex' | ap_timedata |
| 'chplotxmin' | , ap_spectra, ap_xferfun |
| 'chplotxmax' | , ap_spectra, ap_xferfun |
| 'chplotymax' | , ap_spectra, ap_xferfun |
| 'numsubplots' | Is this used for anything? |
| 'chsubplotinfo' | Do I use this? |

Return value

The modified ap_plotinfo object

Notes

This is a bit weak, how sometimes it has to be preceded by "plot" and sometimes not. And I'm not consistent in explaining why some variables are preceded sometimes by a 'ch' and sometimes not. Is this only the case for plotinfo parameters?

ap_time functions

ap_time

Description

Creates an ap_time object, a convenient way to calculate dates and times

Calls

```
time = ap_time();  
time = ap_time(YR,MO,DY,HR,MN,SS);  
time = ap_time(YR,JD,HR,MN,SS);  
time = ap_time('YR/MO/DY HR:MN:SS');  
time = ap_time('YR/JDY HR:MN:SS ');  
time = ap_time(time_in);
```

Arguments

| | |
|---------|---|
| YR | The year (99 means the year 99, so write out 1999!) |
| MO | The month (1-12) |
| DY | The day of the month (1-31) |
| JDY | The Julian day (1-366) |
| HR | The hour (0-23) |
| MN | The minute (0-59) |
| SS | The second (0.0-59.999...) |
| time_in | an ap_time object |

Return value

A new ap_time object

Notes

If ap_time is called without arguments, the time is set to 0001/01/01 00:00:00.0

You can access the time in an ap_timedata object using:

```
var=time.year; var=time.yr, var=time.y  
var=time.month; var=time.mo  
var=time.day; var=time.dy; var=time.d  
var=time.jday; var=time.jd; var=time.j  
var=time.hour; var=time.hr; var=time.h  
var=time.minute; var=time.mn  
var=time.second; var=time.ss; var=time.s
```

+

Description

Add seconds to an ap_time object

Calls

```
time = time_in+seconds;
```

Arguments

| | |
|---------|-------------------|
| time_in | an ap_time object |
| seconds | a scalar |

Return value

The modified ap_time object

Notes

-

Description

Subtract seconds from an ap_time object, or calculate the difference in seconds between two times

Calls

```
time = time_ina-seconds;  
difference = time_ina-time_inb;
```

Arguments

| | |
|----------|-------------------|
| time_ina | an ap_time object |
| time_inb | an ap_time object |
| seconds | a scalar |

Return value

Depends on what you're subtracting. If you subtract seconds from an ap_time object, the value returned is a new ap_time object. If you subtract one ap_time object from another, the value returned the number of seconds difference between the two objects (positive if time_ina is later than time_inb, negative if time_ina is earlier than time_inb)

Notes

Developer notes

Writing ap_ routines

- All routines in the AnyPlot directory, the AnyPlot/ap_view directory or the object creation routines should first call `ap_set()`, which sets up the global variable `ANYPLOT_GLOBALS`.
- Call `ap_get('verbose')` to determine how verbose your routine should be:
 - 0: No text messages.
 - 1: Use `disp(...)` to write messages to the command window
 - 2: Only output messages to dialog boxes (if you wish to)
 - 3: Output messages to command window AND dialogsError and warning messages should be copious and sent to the command window.
- If `ap_get('debug') = 1`, your code should print out debugging info. Each line of debugging info should start with the name of the routine, for example:
"ap_timedata/merge: Starting up"
- Ap_timedata routines should get their plotinfo parameters from `ap_get('plotinfo');`

Writing ap_read* routines

All `ap_read*.m` routines should be able to be called as:

`tdata = ap_read*(datafile,recordNumber,readChannels),`

where `tdata` is an `ap_timedata` object and `datafile` is an `ap_datafile` object. Your routine may ignore `recordNumber` and/or `readChannels` if they make no sense for your data.

As long as the "filetype" parameter in the `datafile` object is the same as your "*" in `ap_read*`, `ap_view(datafile)` should be able to read in new records.

