Data Monitoring Tool Status

John G. Zweizig
LIGO, Caltech

LSC collaboration meeting
Stanford, July 19, 1999

Data Monitor Tool Overview

DMT is meant to address following needs:

- Development/Innovation platform
- Production data quality monitoring
- Detector status depiction.

Running modes:

- Foreground: Interactive data manipulation & display.
- Background: Continuous data checking & feature tagging.

Progress since March LSC meeting

- Data Monitoring Environment.
- Distribution Kits.
- Installation at LHO
- Classes
- Examples

DMT Distribution Kits

- DMT background environment (C language):
  http://www.ligo.caltech.edu/~jzweizig/cdist.tar.gz.

- DMT foreground/ROOT environment (C^{++}):
  http://www.ligo.caltech.edu/~jzweizig/rdist-0.1.tar.gz.

- Create CDist (RDist) directory with:
  gunzip cdist.tar.gz
  tar xvf cdist.tar

- Instructions in README file.

- Distribution includes:
  - README with instructions
  - All source files
  - Makefile (includes installation test)
  - Test data (cdist only).
  - Sun/Solaris binaries (rdist only).

Installation at LHO

Sun E-450 has been delivered to LHO

- 4 × 350MHz CPUs, 1 GByte common memory.
- ATM OC-12 & Gigabit Ethernet connections for networking with CDS, GC and LDAS.
- No GBit Ethernet hub: Connect directly to CDS.
- Running & useable by early August 1999.

Access Plans

- Offsite access available for:
  - Operational monitors.
  - Applications needing current data (*e.g.* for debugging running equipment).
  - Applications needing GDS data (*i.e.* test points).
  - System specific applications (*i.e.* system management).
- Access granted on individual basis.
- "Honor system" resource control (for now).

http://www.ligo.caltech.edu/~jzweizig/Sand/

Classes: Dacc

Purpose:

- Data access class.
- Control reading data from file(s)/online
- Unpack selected Channels from frame

Public Interface

- `Dacc(const char* file)`: create data access object and attach it to a file or online partition.
- `addChannel(const char* name, int decim, TSeries** t)`: Add entry to list of channels to be unpacked.
- `addFile(const char* file)`: Add a file to the list of files to be processed.
- `fillData(Interval& time)`: read frames and fill `TSeries` for all channels in list.
- `list(ostream& str)`: write channel list to output stream.
- `refData(const char* name)`: get pointer to a `TSeries` with data from channel name.
- `rmChannel(const char* name)`: remove name from channel list.

Classes: TSeries

Purpose:

- Time Series container.
- Variable length, arbitrary type.
- Keeps start time, sample rate & accounting info.

Public Interface:

- `TSeries(Time& t0, Interval& Rate, int N, data_t d[])`: Construct a `TSeries`, fill it with data.
- `Append(TSeries& d)`: Append a `TSeries`.
- `extract(Time& t0, Interval& dT)`: Extract a subset of the data as a `TSeries`.
- `getData(int N, data_t d[])`: Get the first N data words.
- `operator+=(float bias)`: Add constant bias to all elements.
- `operator*=(float scale)`: Multiply all elements by scale.
- `operator*(TSeries& rhs)`: Inner product of two `TSeries`.

Classes: FSeries

Purpose:

- Frequency Series container.
- Variable length, Real or Complex type.
- Keeps frequency range, time epoch & accounting info.

Public Interface:

- `FSeries(TSeries& ts)`: Construct an `FSeries` from a `TSeries`.
- `Power(float Flow, float Fhi)`: Return power in frequency interval.
- `setData(TSeries& ts)`: Set `FSeries` from a Fourier transformed `TSeries`.
- `operator+=(FSeries& fs)`: Add an `FSeries`.
- `operator*(FSeries& rhs)`: Inner product of two `FSeries`.
- `operator()(float F)`: Return complex amplitude at frequency F.

Classes: Window

Purpose:

- Window base class performs windowing functions.
- Hamming, Hanning and Blackman window classes.

Public Interface

- `Hamming()`: Hamming window constructor.
- `Hanning()`: Hanning window constructor.
- `Blackman()`: Blackman window constructor.
- `setWindow(int N)`: Calculate window series.
- `operator()(TSeries& ts)`: Return a windowed `TSeries`.

Classes: Filter

Purpose:

- General filter function.
- Design linear FIR filters.
- Maintain input data history

Public Interface

- `dRemez()`: McClellan-Parks linear FIR filter design.
- `dFirW()`: FIR design based on windows.
- `operator()(TSeries& ts)`: Return a filtered `TSeries`.
- `Xfer()`: Calculate filter transfer function.

Classes: Templates

Purpose:

- Template function base class
- Astrophysical template: Newtonian inspiral.
- Other functions: Sine, (gaussian noise).

Public Interface

- `Inspiral(float M1, float M2, Time& t0, Interval dT)`: Newtonian inspiral of objects of masses M1, M2.
- `Sine(float F, Time t0, Interval dT)`: Sine wave of frequency F.
- `TSeries(Time& ts, Interval& Rate, int N, Chirp& func)`: Construct a `TSeries` from the template

Display Macros/Objects

Purpose:

- Display Time and Frequency series
- Maintain a Stripchart display.

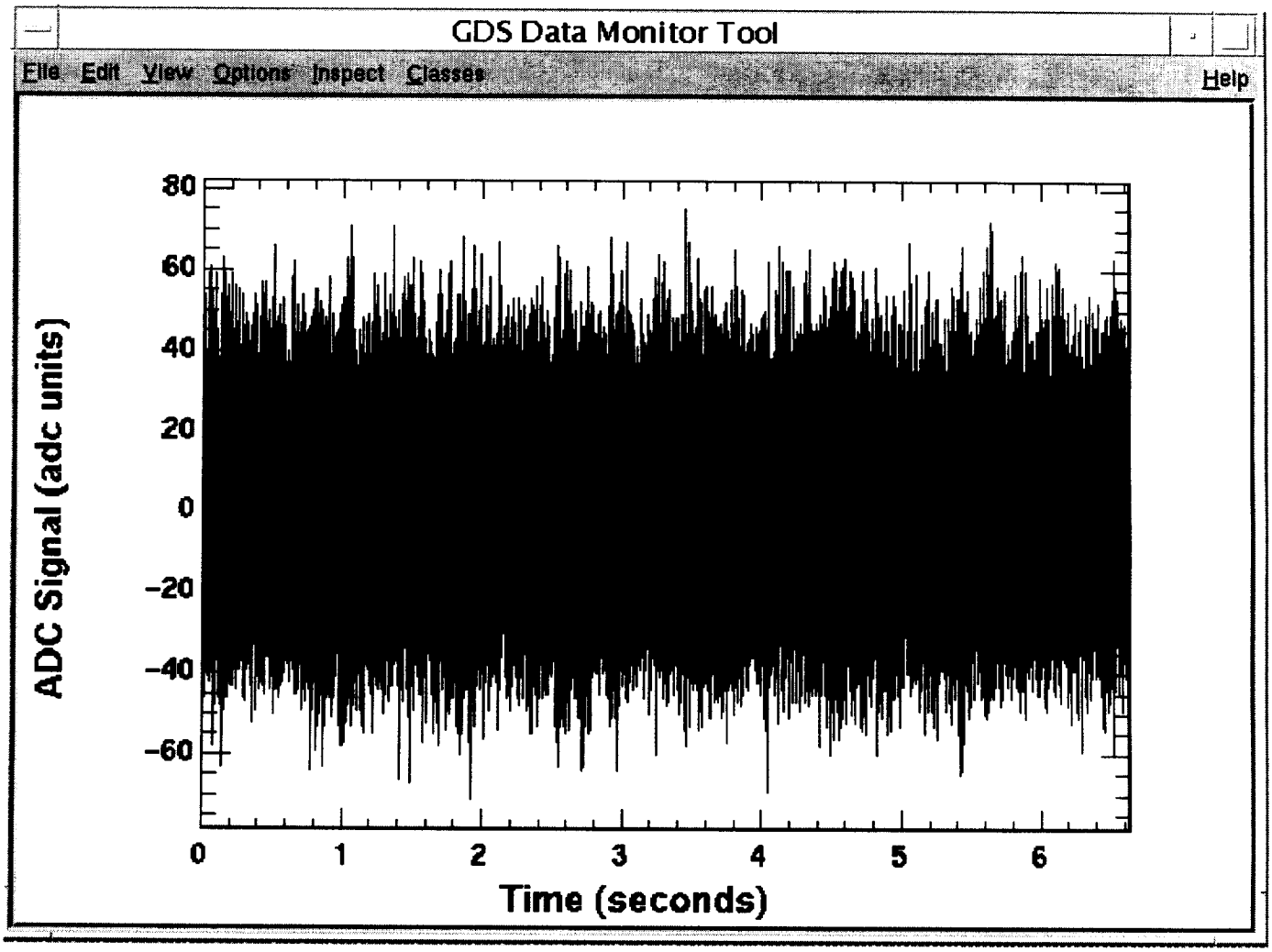
Public Interface

- `Bode(FSeries& ts)`: Bode plot of an `FSeries`.
- `Spectrum(FSeries& ts)`: Plot Fseries power spectrum.
- `THist(TSeries& ts)`: Histogram a specified `TSeries` entries.
- `TPlot(TSeries& ts)`: Plot a specified `TSeries` versus time.
- `TPlot(const char* Chan)`: Plot data from a selected channel.
- `StripChart(int nPen, const char* Title)`: Strip chart constructor.
- `StripChart::setStripe(int Pen, float x, float y)`: Set pen transform.
- `StripChart::plot(float x, float y[])`: Plot a Stripchart time point.

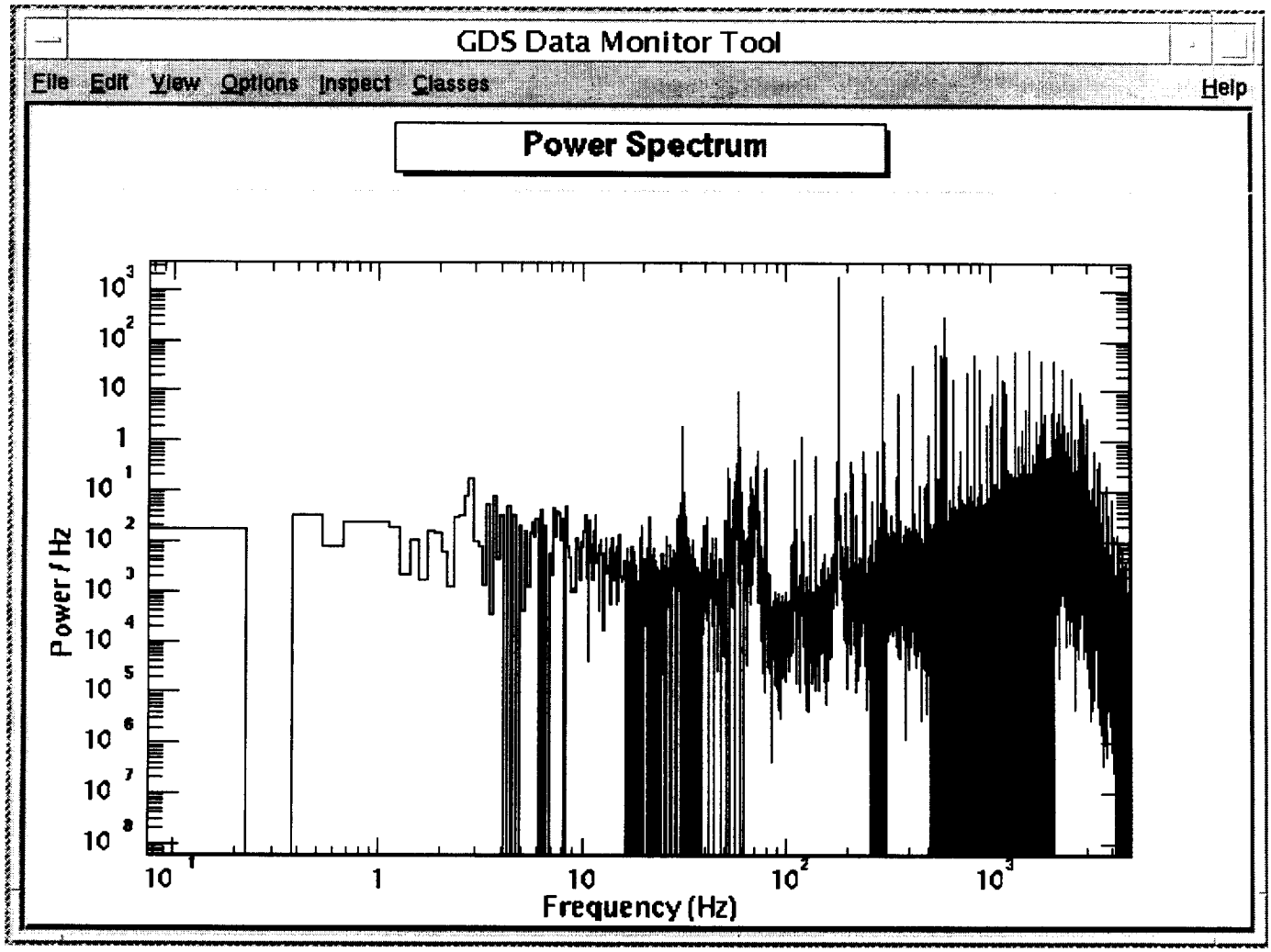
Example 1

```
root[0] In.close();
root[1] In.addFile("/home/jzweizig/temp/frames/C1-*.F");
root[2] In.addChannel("IFO_DMRO");
root[3] In.fillData(6.64098);  // 65536 samples
root[4] TPlot("IFO_DMRO");
root[5] TSeries* ts = In.refData("IFO_DMRO");
root[6] Spectrum(*ts); // Note implicit TSeries->FSeries
```

Example 1a


GDS Data Monitor Tool
File Edit View Options Inspect Classes Help
ADC Signal (adc units)
80
60
40
20
0
-20
-40
-60
0
1
2
3
4
5
6
Time (seconds)

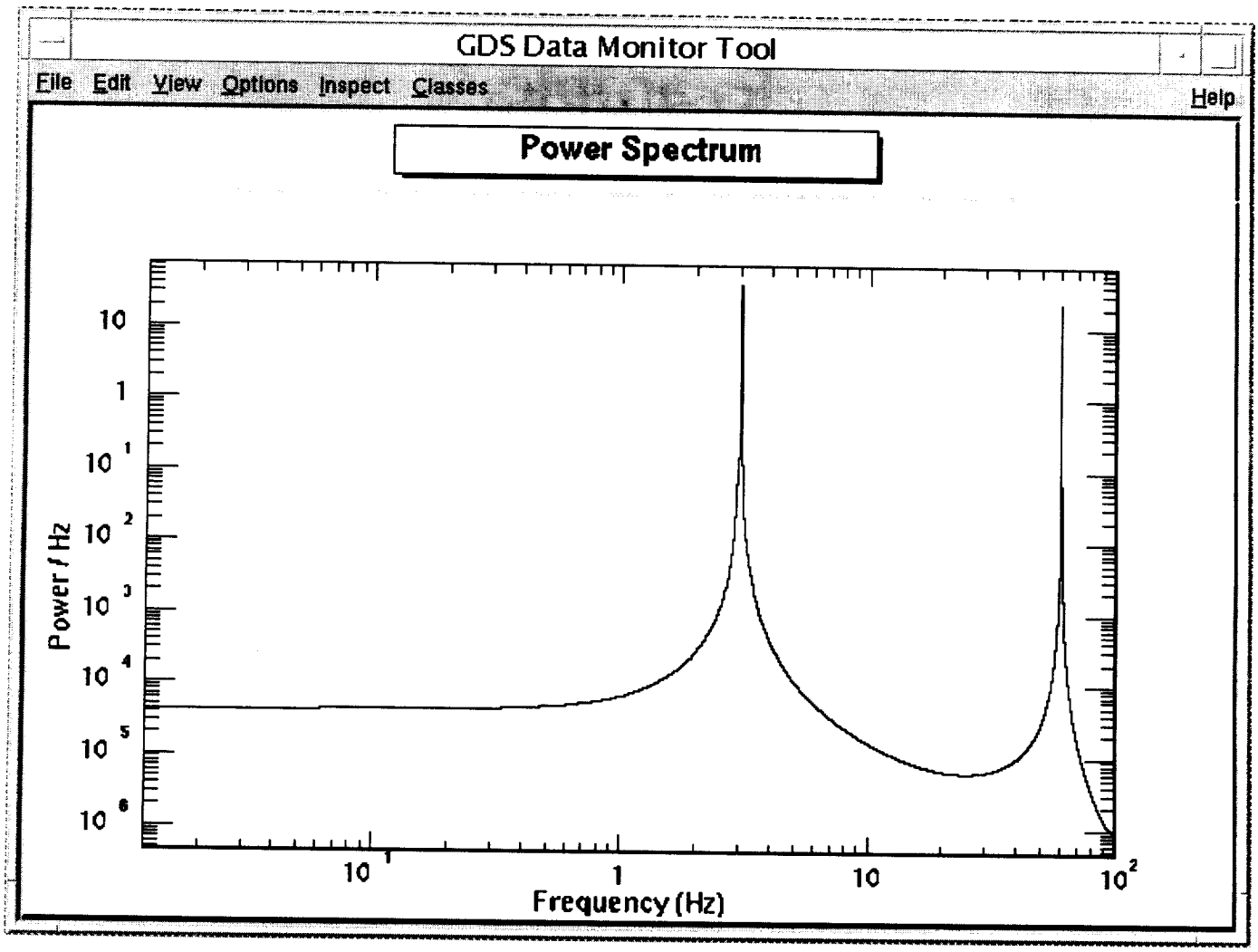
Example 1b


GDS Data Monitor Tool
File Edit View Options Inspect Classes Help
Power Spectrum
Power / Hz
Frequency (Hz)

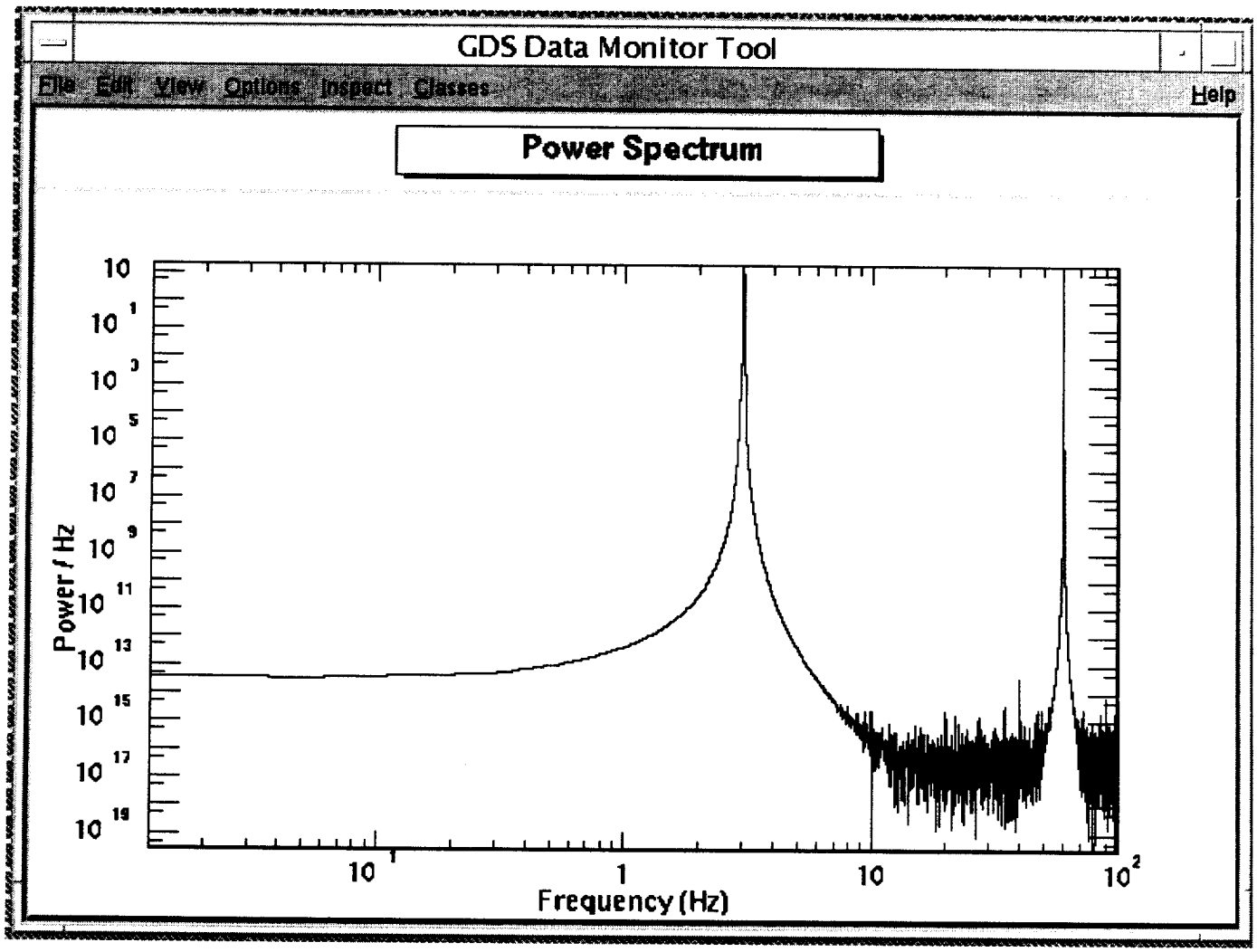
Example 2

```
root[0] Sine sin60(60);
root[1] Sine sin3(3);
root[2] TSeries ts60(Time(0), Interval(0.005), 8192, sin60);
root[3] TSeries ts3x(Time(0), Interval(0.005), 8192, sin3);
root[4] ts3x += ts60;
root[5] Spectrum(ts3x); // Note implicit TSeries->FSeries
root[6] Hanning Wn;
root[7] Spectrum(Wn(ts3x));
root[8] Filter f(25, 200.0);
root[9] f.dFirW(25, "Hamming", "LowPass", 10.0);
root[10] FSeries* fx=f.Xfer();
root[12] Spectrum(*fx);
root[13] TSeries* tf = f(ts3x);
root[14] Spectrum(*tf);
```

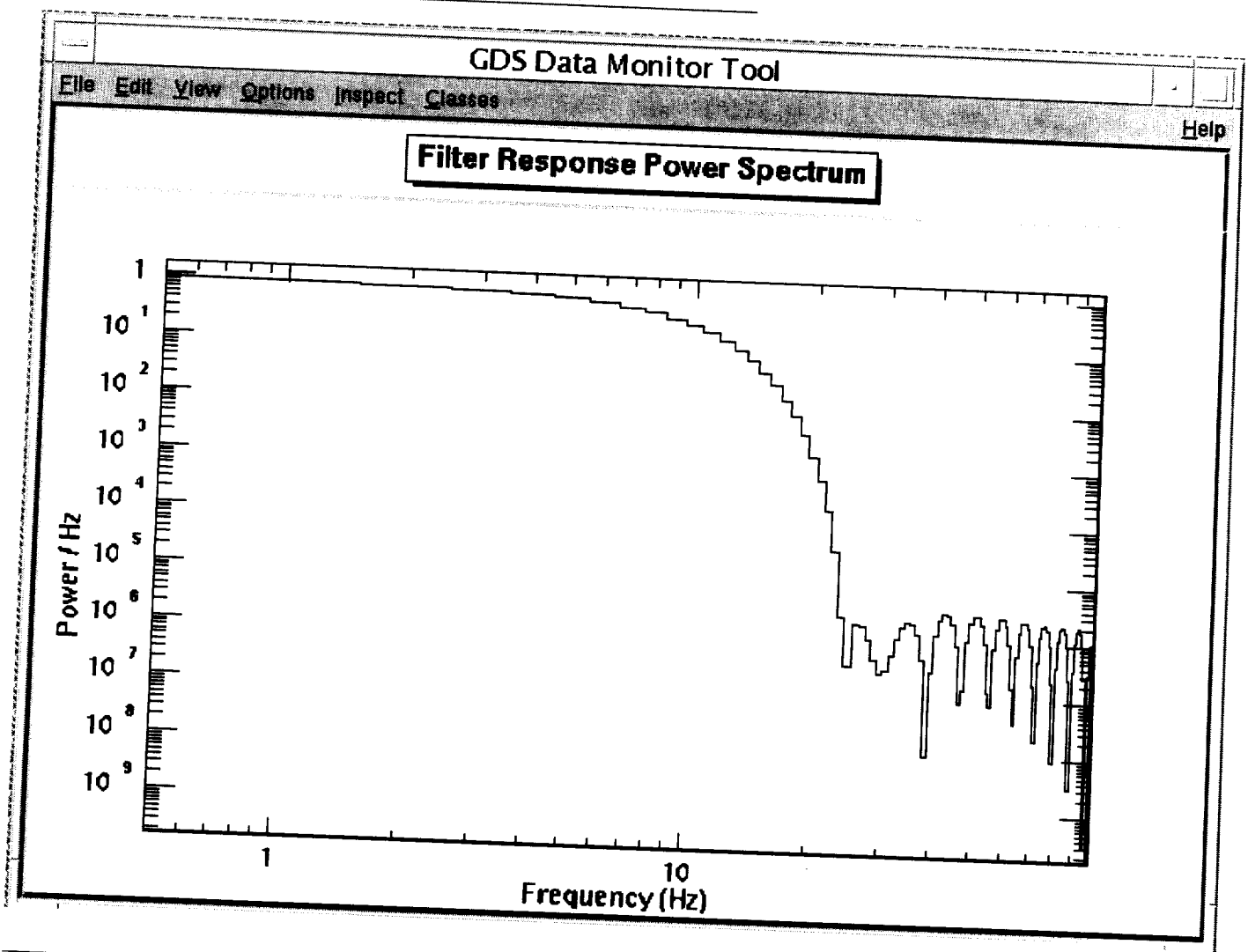
Example 2a


GDS Data Monitor Tool
File Edit View Options Inspect Classes Help
Power Spectrum
Power / Hz
10
1
10 1
10 2
10 3
10 4
10 5
10 6
10 -1
1
10
10 2
Frequency (Hz)

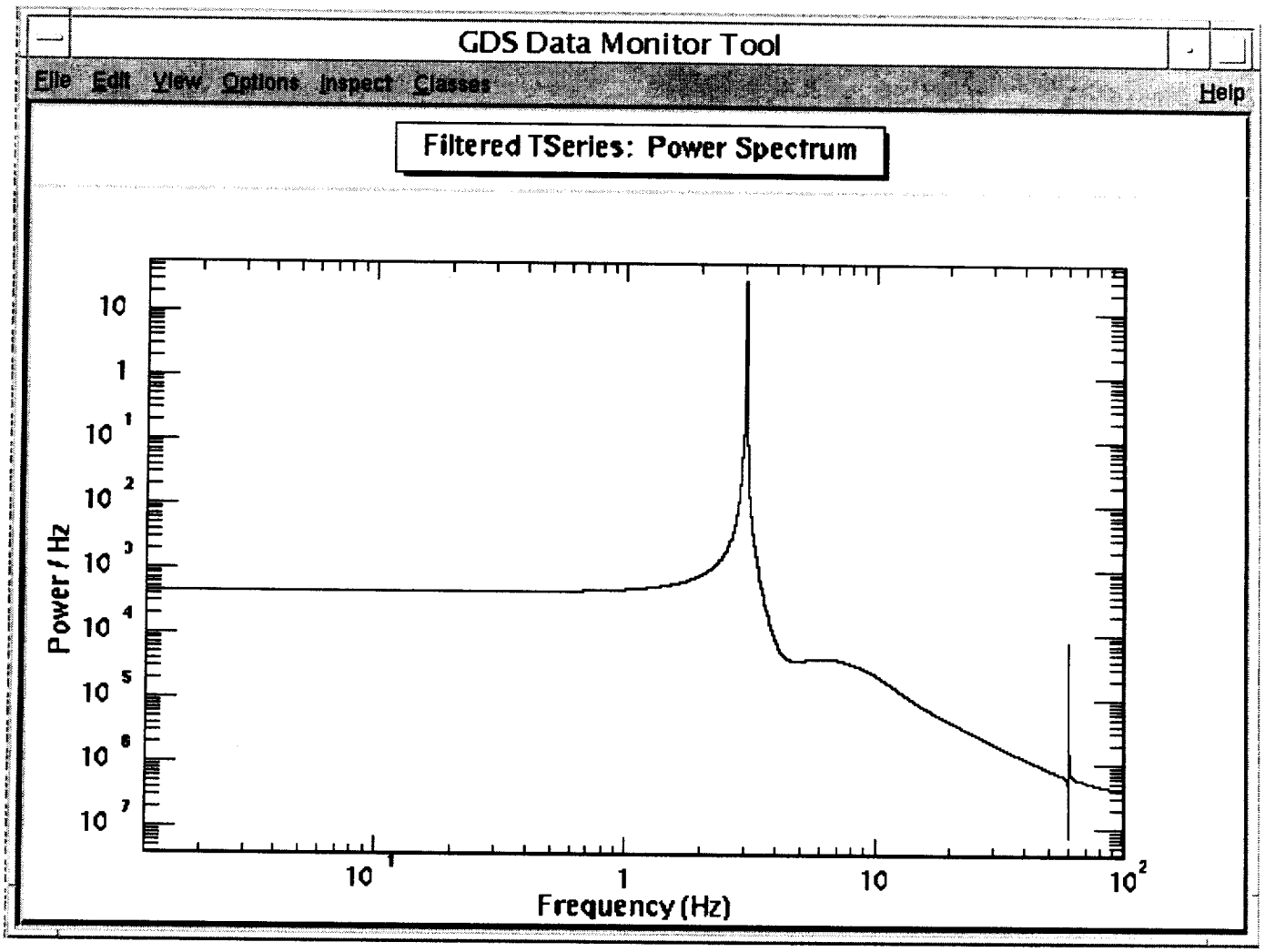
Example 2b


GDS Data Monitor Tool
File Edit View Options Inspect Classes Help
Power Spectrum
Power / Hz
Frequency (Hz)

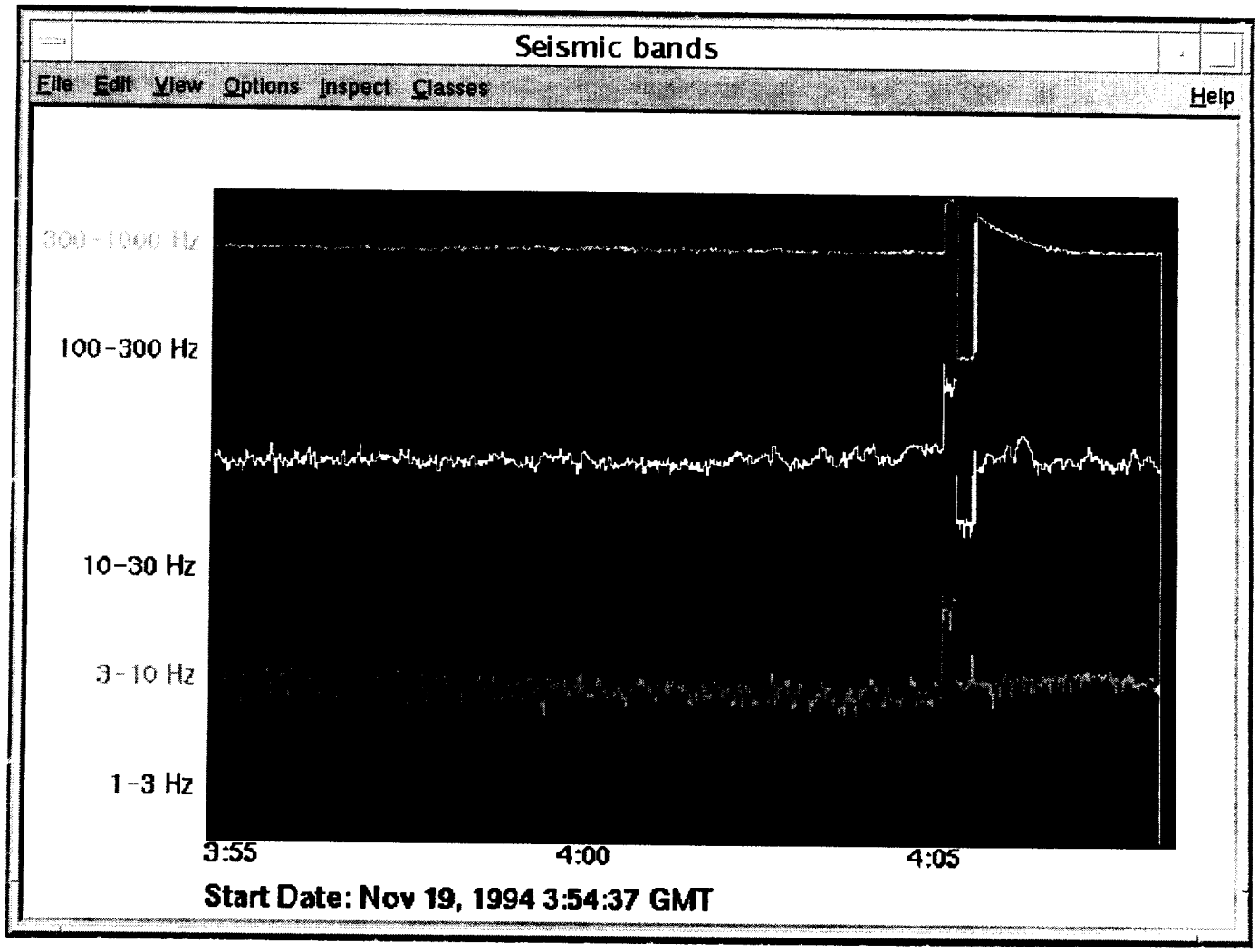
Example 2c


GDS Data Monitor Tool
File Edit View Options Inspect Classes
Help
Filter Response Power Spectrum
Power / Hz
1
10 1
10 2
10 3
10 4
10 5
10 6
10 7
10 8
10 9
1
10
Frequency (Hz)

Example 2d


GDS Data Monitor Tool
File Edit View Options Inspect Classes Help
Filtered TSeries: Power Spectrum
Power / Hz
10
1
10 -1
10 -2
10 -3
10 -4
10 -5
10 -6
10 -7
10 -1
1
10
10 2
Frequency (Hz)

Example 3


Seismic bands
File Edit View Options Inspect Classes Help
300-1000 Hz
100-300 Hz
10-30 Hz
3-10 Hz
1-3 Hz
3:55
4:00
4:05
Start Date: Nov 19, 1994 3:54:37 GMT

Note 1, Linda Turner, 08/17/99 08:56:18 PM
LIGO-G990079-35-M