USGS
science for a changing world

Documentation for a Web Site to Serve ULF-EM (Ultra-Low Frequency Electromagnetic) Data to the Public

How ulfem-data.stanford.edu and the servers that support it retrieve, maintain, and serve data

Open File Report 2010-1321

U.S. Department of the Interior
U.S. Geological Survey

Cover: Professor Simon Klemperer inspecting field equipment at the Marin Headlands site.

Documentation for a Web Site to Serve ULF-EM (Ultra-Low Frequency Electromagnetic) Data to the Public

How ulfem-data.stanford.edu and the servers that support it retrieve, maintain, and serve data

By Danny A. Neumann, Selwyn McPherson, Simon L. Klemperer, Jonathan M.G. Glen, Darcy K. McPhee, and Karl Kappler

Open File Report 2010-1321

U.S. Department of the Interior
U.S. Geological Survey

U.S. Department of the Interior
KEN SALAZAR, Secretary

U.S. Geological Survey
Marcia K. McNutt, Director

U.S. Geological Survey, Reston, Virginia 2010

For product and ordering information:
World Wide Web: *http://www.usgs.gov/pubprod*
Telephone: 1-888-ASK-USGS

For more information on the USGS—the Federal source for science about the Earth, its natural and living resources, natural hazards, and the environment:
World Wide Web: *http://www.usgs.gov*
Telephone: 1-888-ASK-USGS

Suggested citation:

Neumannn, D.A., McPherson, S., Klemperer, S.L., Glen, J.M.G, McPhee, D.K, and Kappler, K., 2010, Documentation for a web site to serve ULF-EM (Ultra-Low Frequency Electromagnetic) data to the public: U.S. Geological Survey Open-File Report 2010-1321, 44 p. [*http://pubs.usgs.gov/of/2010/1321/*].

Contents

Figures

Documentation for a Web Site to Serve ULF-EM (Ultra-Low Frequency Electromagnetic) Data to the Public

How ulfem-data.stanford.edu and the servers that support it retrieve, maintain, and serve data

By Danny A. Neumann, Selwyn McPherson, Simon L. Klemperer, Jonathan M.G. Glen, Darcy K. McPhee, and Karl Kappler

Introduction

About this Manual

This manual documents ulfem-data.stanford.edu, a Web site that serves ultra-low frequency electromagnetic data to the internet.

Project Overview

The Stanford Ultra-Low Frequency Electromagnetic (ULF-EM) Monitoring Project is recording naturally varying electromagnetic signals adjacent to active earthquake faults, in an attempt to establish whether there is any variation in these signals associated with earthquakes.

Our project is collaborative between Stanford University, the U.S. Geological Survey (USGS), and UC Berkeley. Lead scientists are Simon Klemperer (Stanford University), Jonathan Glen (USGS) and Darcy Karakelian McPhee (USGS).

Our initial sites are in the San Francisco Bay Area, monitoring different strands of the San Andreas fault system, at Stanford University's Jasper Ridge Biological Preserve (JRSC), Marin Headlands of the Golden Gate National Recreation Area (MHDL), and the UC Berkeley's Russell Reservation Field Station adjacent to Briones Regional Park (BRIB). In addition, we maintain in conjunction with the Berkeley Seismological Laboratory (BSL) two remote reference stations at the Bear Valley Ranch in Parkfield, Calif., (PKD) and the San Andreas Geophysical Observatory at Hollister, Calif., (SAO). Metadata about our site can be found at http://ulfem-data.stanford.edu/info.html. Site descriptions can be found at the BSL at http://seismo.berkeley.edu/, and seismic data can be obtained from the Northern California Earthquake Data Center at http://www.ncedc.org/.

The site http://ulfem-data.stanford.edu/ allows access to data from the Stanford-USGS sites JRSC, MHDL and BRIB, as well as UC Berkeley sites PKD and SAO.

Important Notes

- This manual was intended to be read linearly, as there are important notes in each section. To get the most out of it, consider at least skimming the entire document.

- This document describes the general function of each software component of this project. For more technical details, please read the actual documentation within the code, which is included in the appendices.
- When this document mentions "days," it is referring to *UTC* days. For instance, when a server is said to download "a day's worth of data," it downloads 24 hours of data starting at 0:00 UTC.
- The username@domain notation is used in this document; for instance, an account at quake called jane can be expressed as jane@quake.
- The terms "ulfm Web site" and "ulfem-data" refer to http://ulfem-data.stanford.edu/.
- There is an ulfem user on the ulfem server at Stanford. Everything mentioned in this document that happens on the ulfem server is done by the ulfem user.
- The notation "~/" indicates the home directory for whatever user is being discussed. For instance, when the document is describing a process on the ulfem server, ~/scripts refers to the ulfem user's scripts directory, or /home/ulfem/scripts.
- Different type-faces are used to help distinguish computer commands, computer directories and physical computers, all shown in Andale Mono, from references to separate (sub-)sections of this document shown in *italic Times New Roman*, from the main text of the document in Times New Roman.

Servers

Overview

Server Names and Locations

The following servers are mentioned in this document:

ulfem.stanford.edu:	Data storage and processing server hosted by the Stanford School of Earth Sciences
quake.geo.berkeley.edu:	Data storage server at Berkeley
pangea.stanford.edu:	Web server hosted by the Stanford School of Earth Sciences
ulfem-data.stanford.edu:	Virtual Web site hosted on pangea
sesfs.stanford.edu:	File server hosted by the Stanford School of Earth Sciences.

The servers will be referred to as ulfem, quake, pangea, ulfem-data, and sesfs. Their various roles are discussed in the next section.

How Servers Interact to Generate Plots and Manage Data

This section describes the chain of events that occurs when a user requests a plot from the ulfem-data Web site. More specific details about each link in the chain appear later in this document.

Pangea hosts the ulfem-data Web site; when a user visits the site, she interacts with the pangea server. After she requests a plot on the front page, pangea invokes process.php, passing it the plot parameters that the user has specified. The PHP script connects from pangea to ulfem@ulfem via SSH using public/private key encryption, where it remotely calls the script ~/run_imageplot.sh. That shell script is responsible for calling the MATLAB executable file imageplot. Imageplot accesses the necessary data and generates an image of the plot that the user requested. Then, imageplot puts the image into the ~/data/usrImg directory on ulfem[1]. Finally, if there are no errors, process.php generates a webpage that displays the image.

[1]The ~/data directory is a symbolic link, pointing to a folder on the "ulfem" volume on the sesfs file server which is accessible to both the ulfem and pangea servers through a network file mount (NFS).

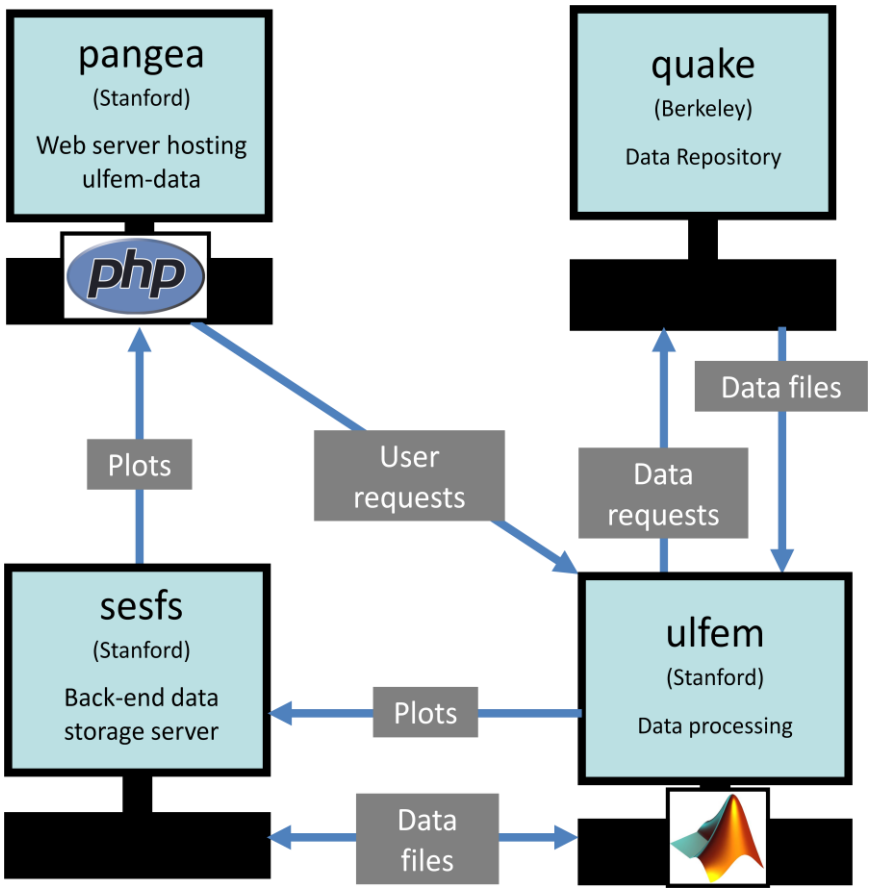

Figure 1. How the various servers interact.

In order to generate the plots, imageplot uses data available in the ~/data directory on ulfem. Ulfem automatically downloads that data from quake at UC Berkeley every 30 minutes through SSH by using private/public key authentication. See section 2.3: *Data Acquisition and Storage* to learn more.

Scripting Languages Used

- MATLAB: A MATLAB executable was implemented because it integrates relatively smoothly with our Web application and because MATLAB is widely familiar.
- PHP: This server-side scripting language is very common and supported by the pangea Web server.
- Shell scripting: Shell scripting was implemented to handle Unix commands.
- Python: The python scripts that were implemented are extensions of programs written by Karl Kappler at UC Berkeley.

Server Directory Structure

Note: each bullet point in this section represents a directory; indented bullet points are subdirectories. See section 2.4 *Logging in remotely: SSH and SFTP access* to access these directories manually.

Ulfem

The user ulfem on the server ulfem.stanford.edu has a home directory with the following directory structure:

- **backups**: TAR archive backups of the ulfem home directory. Not frequently used.
- **data:** Where ulfem stores data files and user-generated images. This "directory" is actually a symbolic link to a directory on the Stanford School of Earth Sciences Web server (sesfs:/exavol/ulfem). Our project is allocated 100 GB of storage space in that directory.[2]
    - **2009_1hz**: stores 1 Hz data for 2009. The "1hz" suffix is a vestige of the old file system.
        - **123**: stores 1 Hz data for all channels on the 123rd of 2009
    - …And so on.
    - **40hz**: A directory to store 40 Hz data temporarily. *Note*: 40 Hz data handling was still experimental and not supported when this document was last updated. However, there is code in imageplot to acquire and store data in this directory, which can be finished at some point in the future. Imageplot was designed to get only the data it needs and put it into this directory. To conserve space, a cron job automatically deletes data that has not been accessed in 15 minutes in this directory (see section 2.3: *Data acquisition - Automated data acquisition and maintenance through cron jobs* for more information).
- **imageplot**: Stores code and MATLAB executable for the imageplot program (see section 3.1: *Plot generation scripts - Imageplot* for more details).
- **imageplotTst:** Houses an experimental version of imageplot, which can be freely modified and tested without disturbing the main Web site.
- **misc:** Directory into which administrators can put miscellaneous files.
- **scripts**: Scripts that the ulfem server uses to carry out its tasks, as well as several scripts for manual use.
    - **old:** A place to put old files that are no longer active, but may be useful for reference.

---

[2]A year's worth of 1 Hz data is about 4.7 GB, so 100 GB is plenty of capacity for the ulfem Web site. Scheduled scripts automatically delete 1 Hz data older than a decade (see the section 2.3: *Data acquisition and storage: Automated data acquisition and maintenance through cron jobs* for more information). Data storage space on sesfs can be expanded if necessary.

- **site_info**: Notes about data acquisition sites. Documents under this directory end in a suffix similar to "_072409_DN.doc." That particular file ending indicates it was last modified on July 24, 2009, by someone with the initials "DN."
    - **generic_equip**: Notes about the equipment used at data acquisition sites.
    - **BRIB:** Notes about the Briones site.
    - **JRSC:** Notes about the Jasper Ridge site.
    - **MHDL:** Notes about the Marin Headlands site.
    - **PKD:** Notes about the Parkfield site.
    - **SAO:** Notes about the Hollister site.
- **user_space**: directory in which individuals can store their own files. Its purpose is similar to that of the misc directory, except the files are less suited for general use.

Quake

Quake, or quake.geo.berkeley.edu, is the central data repository from which ulfem downloads data files.

- **ulfem_scripts:** stores scripts that ulfem remotely runs on quake to obtain data. These files also are backed up on the scripts directory on ulfem. See section 3.0: *Scripts and programs* for more information.
- **scratch:** Houses the ulfem directory (see below).
    - **ulfem:** Directory used by scripts on quake to store data temporarily before sending it back to ulfem.

Pangea

Pangea is the Web server that hosts the ulfem-data Web site. All directories mentioned in this section are located under /WWW/research/groups/crustal/ulfem,[3] and everything within this directory and its subdirectories are visible to the world at http://ulfem-data.stanford.edu/.[4]

- **css:** CSS[5] files for the Web site.

---

[3]The entire /WWW directory tree on pangea is a nework file is actually a network file mount of the WWW volume on the sesfs file server.

[4]At the time that this document was written, pangea was running Apache version 2.0.63 and PHP version 5.2.10.

[5]CSS: "Cascading Style Sheets." CSS files are used to set the look and feel of the ulfem Web site.

- **data:** Points to the same location that the ~/data directory on ulfem does (which is located on the server sesfs.stanford.edu). See section 2.3: *Data acquisition and storage* for more information.
- **dirLIST_files:** Directory for dirLIST,[6] which is a third-party application used by previous.php.
- **img**: Static images that are used by the Web site, as opposed to user-generated ones.
- **js:** Javascript files used by the Web site.[7]
- **old:** Files that are no longer active but may be useful for reference.

Data Acquisition and Storage

Data Storage Directory

Data is stored in the ~/data directory on ulfem, which is a network file mount of the "ulfem" volume on the sesfs file server.

Data Files: Naming Conventions and Storage

The ~/data directory has the following naming convention: ~/data/YYYY_1hz/ts/DDD/channel_designator.YYDDD000000.d, where YYYY is the four-digit year (YY is, naturally, the two-digit year), DDD is the Julian day of the year, and "channel_designator" is a string determining the channel and station from which the data comes. A typical filename would be ~/data/2008_1hz/081/JRSC.LQ5.08081000000.d, which would indicate:

- Time: the 81st day of 2008.
- Station: JRSC, or the station located at the Jasper Ridge Biological Preserve.
- Frequency: L, which means 1 Hz (all files in the 2008_1hz directory have the designator L).
- Channel: Q5, or electric channel number 5. Note that channels are not numbered sequentially.
- Start time: 000000, or 00:00:00 UTC time. All data files have trailing zeros, but they are left in place because corrupted files do not end in that suffix and are therefore easy to find. Additionally, this format is convenient because it is in accordance with the naming convention implemented by Berkeley scripts.

Frequency designators:

- B – 40 Hz (rarely used).

---

[6]See http://sourceforge.net/projects/dir-list/ for more information.

[7]The ulfem Web site uses Javascript, a client-side scripting language (or at least client-side in this context), to implement its data-range selection tools.

- L – 1 Hz.
- V - 0.1 Hz (never used, but can be requested using quan2asc—see section 2.4: *Data management scripts* for more information).

Channel designators:

- Q – Electric data.
- T – Magnetic data.

Channel numbers:

- The channel numbers for each station are listed at http://www.ncedc.org/ftp/pub/doc/BK.info/BK.channel.summary.day. These frequency and channel designators are also used when invoking quan2asc, which is discussed in more detail in section 3.2: *Data management scripts*.

Automated Data Acquisition and Maintenance through cron Jobs

Ulfem relies on cron jobs to acquire its data automatically. A cron job is a command that an administrator can program a Unix-based computer to run at specified times. For instance, a cron job could tell a computer to back up the contents of a directory every week, or run a script (for example, getYesterdaysData.sh) every hour. The cron jobs in place belong to the user ulfem. Ulfem's cron jobs do the following:

- Download 1 Hz data every 30 minutes using remote_getAscii.sh.
- Download the previous day's data, to fill in any potential gaps, by calling getYesterdaysData.sh.
- Delete 40 Hz data files that are more than 15 minutes old in the ~/data/40hz directory.
- Delete week-old imageplot files in the ~/data/userImg directory.
- Delete data directories that are more than 10 years old by using decadizer.sh.

The ulfem user can edit the existing cron jobs by typing crontab –e into the terminal window. See A*ppendix A: Ulfem scripts* to view the cron job code.

Logging In Remotely: SSH and SFTP Access

Administrators can log into ulfem and quake remotely through SSH and sftp clients to modify scripts and other files on those servers. One must access sesfs in order to modify files used by pangea. Consult section 2.2: *Server directory structure* for information about file locations. To get passwords to log into ulfem and quake, please contact Simon Klemperer at sklemp@stanford.edu.

Ulfem

- Username: ulfem.
- Server: ulfem.stanford.edu.

Quake

- Username: darcy.
- Server: quake.geo.berkeley.edu.

Sesfs

- Accounts are only available to faculty, staff, and students at the Stanford School of Earth Sciences and their fully-sponsored guests. Please visit this Web site for access information: http://earthsci.stanford.edu/computing/sesfs/ .

Scripts and Programs

Plot Generation Scripts

Imageplot

Imageplot is responsible for generating the plots displayed on the ulfem-data Web site. To invoke imageplot, pangea connects to ulfem and passes necessary parameters to the script ~/imageplot/run_imageplot.sh, which then calls the MATLAB executable ~/imageplot/imageplot.

Imageplot, using data files in ulfem's ~/data directory, generates a plot and puts it into ~/data/userGenImg, which is accessible by pangea.

Manual Use

One can SSH into ulfem and use imageplot via command line. To do that, one needs to call ~/imageplot/run_imageplot.sh with parameters that are described below.

Compiling Imageplot

In order to use the imageplot.m file with the ulfem Web site, one must compile it first. To do that, log into ulfem through SSH, change directories to the ~/imageplot directory and run the script "compile_ip.sh" to update the executable. Please be careful, as breaking imageplot breaks the ulfem-data Web site. It is advisable to test any new versions of imageplot.m in the imageplotTst directory before modifying the primary imageplot MATLAB executable.

Running Imageplot from MATLAB: Usage

imageplot(stationI, channelI, starttimeI, endtimeI, UTCoffsetI,outputI,typeI)

Arguments

- **stationI**: the station you want to plot, such as 'BRIB'. If you want to plot multiple stations on one graph, enter them in the format {'BRIB','JRSC'}, or any combination you like. The program scales the image automatically and stacks the subplots so that they are parallel to one another.
- **channelI**: similar format as stationI. You can also input an array of channels. Any permutations of stationI and channelI have to be in the form n-by-one, one-by-n, or n-by-n. For instance, you can enter one station with three different channels, or three stations with one channel, or three stations and three channels, and the program "fills in the blanks" by repeating the individual channel or station three times.
- **starttimeI** is in the format: YYYY.MM.DD.hh:mm:ss. Like the previous two arguments, you can input an array of time.
- **endtimeI**: Similar to starttimeI, except using end times.
- **UTCoffsetI**: the UTC offset. For instance, the offset for PDT time is -7.
- **outputI**: the name of the image file. If outputI is set to foo, then the filename is foo.jpg.
- **typeI**: pwr (power density), spec (spectrogram), or plot (plot). Note that power density graphs and spectrograms are still experimental and rely on 40 Hz data.

Examples

- imageplot('BRIB',{'LT1','LT2','LT3'}, '2008.8.1.3:00:00','2008.8.2.3:00:00',-7,'foo','plot').
    - Produces three plots of 1 Hz data at BRIB—one for each of the magnetic channels—from 3:00 AM on 8-1-08 to 3:00 AM 8-2-08 local time.
- imageplot({'MHDL','BRIB','SAO'},{'LT1','LT2','LT3'}, '2008.8.1.3:00:00','2008.8.2.3:00:00',-7,'foo','plot').
    - Generates plots for MHDL's channel LT1, BRIB's LT2, and SAO's LT3 over the same time interval.
- imageplot({'MHDL','BRIB'},'LT1', '2008.8.1.3:00:00','2008.8.2.3:00:00',-7,'foo','plot').
    - Generates plots for MHDL's and BRIB's LT for the same time interval.

Data-Management Scripts

Important Notes

- All of these scripts are located in the ~/scripts directory on ulfem.
- The "ulfem_" prefix means the script in question is designed to reside on the Quake server at Berkeley, in the directory ~/ulfem_scripts on the darcy account. A copy of each of the ulfem_ scripts is kept on ulfem for reference and backup. In contrast, remote_ scripts are intended to be run from ulfem. It is perhaps counterintuitive, but it is the way things were initially established.
- All 40 Hz data processing is still experimental and may not work properly.
- As always, data on ulfem are stored in ~/data directory structure. See section 2.2: *Server directory structure* for more information.

The Scripts

- **decadizer.sh**: Called by the computer to delete decade-old data automatically. It is not intended for human use.
- **getAYear.sh**: Usage: sh getAYear.sh YYYY DDD, where YYYY is the year and DDD is the Julian day of year. It gets a full year's worth of 1 Hz data prior to the specified date.
  Example: "sh getAYear.sh 2009 123" would get all days between the 123rd day of 2008 and the 123rd day of 2009, inclusive, by repeatedly calling remote_getAscii.sh.
- **getRange.sh**: Usage: sh getRange.sh YYYY DDD DDD, where YYYY is the year and DDD DDD is the start and end day of year, respectively. It gets all the 1 hz data between the first and second DDD parameter (day of year).
  Example: calling sh getRange.sh 2009 123 234 B would get all the 40 Hz data between days 123 and 234. It works by repeatedly calling remote_getAscii.sh.
- **getYesterdaysData.sh**: Usage: sh getYesterdaysData.sh. It gets yesterday's 1 Hz data (that is, data from the day before an administrator runs the script). It works by calling remote_getAscii.sh.

- **remote_get40hzRange.sh**: *experimental.* It calls ulfem_get40hzRange.py on quake, passing it the same parameters it received. See that file for more information.

- **remote_getAscii.sh**: Usage: sh remoteGetAscii.sh YYYY DDD. It calls ulfem_getAscii.py, sending it the same parameters it received.

- ***ulfem*_get40hzRange.py**: *experimental.* Usage: python ulfem_get40hzRange.py site channel_specifier YYYY.ddd,hh:mm YYYY.ddd,hh:mm. It is a script that is meant to be used on quake that gets a single file of 40 Hz data between the start and end time.
  Example: python ulfem_get40hzRange.py BRIB BQ2 2009.020,02:00 2009.020,03:00.
  Administrators logged into ulfem should use remote_get40hzRange.sh instead of this script.

- ***ulfem*_getAscii.py**: Usage: python ulfem_getAscii.py YYYY DDD F, where F is frequency L or B. It gets 1 or 40 Hz data on YYYY DDD. Ulfem administrators should use remote_getAscii.sh instead of this script. Note that remote_getAscii.sh does not have the F parameter; remote_getAscii.sh automatically passes this script ~~–L~~."

The ulfem-Data Web Interface

Where ulfem-Data is Hosted

http://ulfem-data.stanford.edu/ is a –cosmetic" URL that redirects to http://pangea.stanford.edu/research/groups/crustal/ulfem/ where the Web site is hosted. See section 2.4: *Logging in remotely: SSH and SFTP access* to find out how to access files in that Web directory.

Web Interface: HTML, PHP, and Other Scripts

When users visit http://ulfem-data.stanford.edu/, they can use a Javascript form to enter the parameters they want for each plot. Hitting –submit" calls process.php, which, from pangea, tries to connect to ulfem using SSH. If it is successful, it passes the user's parameters along to run_imageplot.sh, which generates the plots and outputs them to ~/data/userImg/filename.png. Because the ~/data/ directory is directly accessible by pangea and shared on the internet, process.php simply outputs a tag with the image's URL. If it fails at any point, the PHP script shows users an error message featuring an adorable lolcat.

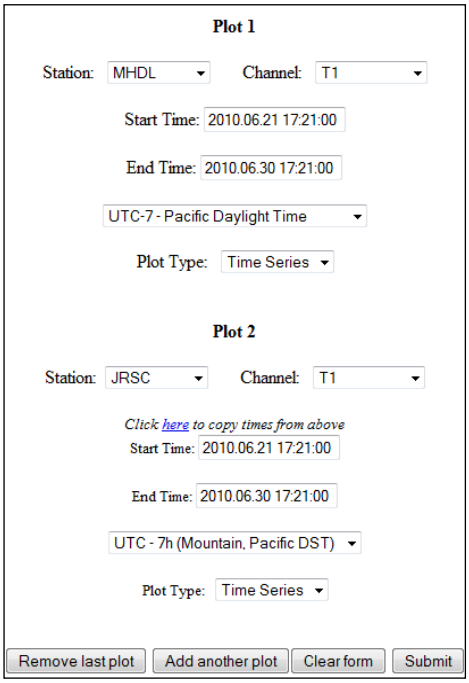

Figure 2. A snapshot of the Web site interface filled in with sample data.

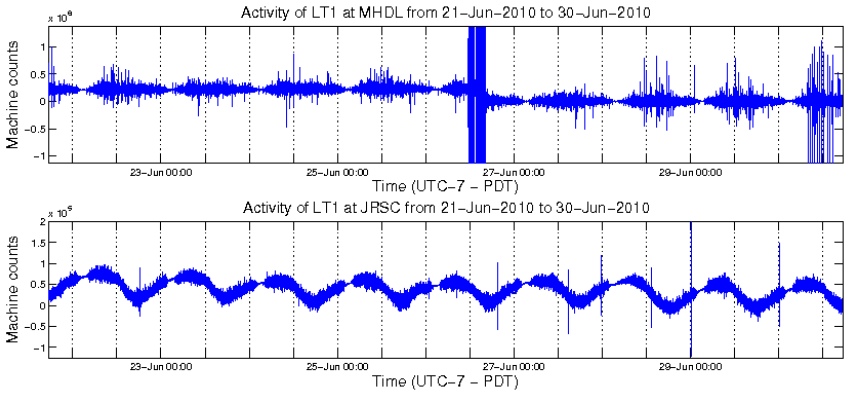

Figure 3. A plot generated by submitting the form shown in figure 2.

Acknowledgments

Danny Neumann and Selwyn-Lloyd McPherson were responsible for programming ulfem-data, under the guidance of Simon Klemperer, Jonathan Glen, Darcy McPhee, and Karl Kappler. The Stanford School of Earth Sciences provides computing resources, and Phil Farrell and Kai Lanz of the Stanford School of Earth Sciences IT department were particularly helpful throughout the development of our project.

Funding for the programming was received from the Vice-Provost or Undergraduate Education at Stanford University. Primary funding for establishment and maintenance of the Stanford-USGS ULFEM network comes from National Science Foundation Earthscope Science award EAR-0346236, with additional in-kind support from USGS and UC Berkeley.

Appendixes

Appendix A: Ulfem Scripts

Decadizer.sh

```
# At the beginning of each year, a cron job calls this script, which
# subsequently deletes 1hz data folder from ten years ago. For instance,
# on January 1st, 2009 UTC, while people in Greenwich, UK are celebrating
# the new year, ulfem quietly deletes the /home/ulfem/data/1998_1hz
# folder. Out with the old, in with the new.

year=`date +%y`
# this line strips leading zeros from year
year="$(echo $year | sed 's/0*//')"
# warning: this line only works in the current millenium!
fullYear=$[$year+2000]
tenYearsAgo=$[$fullYear-10];
rm -rf /home/ulfem/data/$[tenYearsAgo]_1hz
```

getAYear.sh

```
#!/bin/sh

##Arguments: YYYY DDD L/B
##Gets L data for 1 year before YYYY DDD

startYear=$1
startDay=$2
lastYearsLength=`date -d$[$startYear-1]-12-31 +%j`
date=''
echo $lastYearsLength
sleep 1

for((i=1; i<=$lastYearsLength; i++))
  do
  sleep .1
  if [ $[i+startDay] -gt $lastYearsLength ]
  then
    date="$startYear $[$i+startDay-$lastYearsLength]"
  else
    date="$[startYear-1] $[$i+startDay]"
  fi
  echo GETTING $date
  sh remote_getAscii.sh $date
done
```

getRange.sh

```
#!/bin/sh

##Arguments: YYYY DDD DDD
##Gets L data from the first day (DDD) to the second DDD, within year YYYY

year=$1
startDay=$2
endDay=$3

thisDay=$startDay

while ([ $thisDay -le $endDay ])
do
  sh /home/ulfem/scripts/remote_getAscii.sh $year $thisDay
  thisDay=$[$thisDay+1]
done
```

getYesterdaysData.sh

```
#!/bin/sh

### This gets yesterday's ULFEM data (yesterday in terms of UTC time) through
### remote_getAscii.sh. See that file's documentation for more information.

year=`date +%Y`
today=`date +%j -u`
## the line below strips leading zeroes
today="$(echo $today | sed 's/0*//')"
yesterday=$[today-1]
# This if statement makes the program work on January 1st.
# January 1st, 2008 would go to the 365th day of 2007.
if [ $yesterday == 0 ]
 then
  yesterday=`date -d$[$year-1]-12-31 +%j`
  year=$[$year-1]
fi

## The call that actually gets the data
sh /home/ulfem/scripts/remote_getAscii.sh $year $yesterday
remote_get40hzRange.sh
```

remote_get40hzRange.sh

```
#!/bin/sh

# This shell script calls ulfem_get40zAscii.py, giving it the same
# parameters that you give this script. For more information, see
# ~/scripts/remote/ulfem_get40hzAscii.py.

for i in $*
do
        params=$params$i" "
done

echo Calling remote_get40hzRange.sh $params
echo "<br>"
# Makes a temporary file, cmds.tmp, to store the commands
# that are to be executed on the remote server
cat > /home/ulfem/scripts/cmds.tmp << EOF
cd ~/ulfem_scripts/
echo "calling ulfem_get40hzRange.py"
python ulfem_get40hzRange.py $params
exit
EOF
# Runs the commands
echo /home/ulfem/scripts/cmds.tmp
echo "<br>"
/usr/bin/ssh darcy@quake.geo.berkeley.edu < /home/ulfem/scripts/cmds.tmp
>cmds.tmp
exit 0
```

remote_getAscii.sh

```
#!/bin/sh

# This shell script calls ulfem_getAscii.py, giving it the same
# parameters that you give this script. For more information, see
# ~/scripts/remote/ulfem_getAscii.py.
# If there's no arguments, then it just gives it today's YYYY DDD,
# in utc time.

# The following concatenates all the parameters into string "params"
for i in $*
do
        params=$params$i" "
done

if [ "$1" == "" ]
then
 params=$params`/bin/date +%Y -u`" "
fi

if [ "$2" == "" ]
then
  params=$params`/bin/date +%j -u`" "
fi

#if [ "$3" == "" ]
#then
#  # if it doesn't specify a data channel, the script also downloads
#  # 40hz data by calling itself:
#  sh /home/ulfem/scripts/remote_getAscii.sh $params L
#  params="$params B"
#fi

echo params

echo Calling remote_getAscii.sh $params

# Makes a temporary file, cmds.tmp, to store the commands
# that are to be executed on the remote server

cat > /home/ulfem/scripts/cmds.tmp << EOF
cd ~/ulfem_scripts/
echo "calling ulfem_getAscii"
python ulfem_getAscii.py $params L
exit
EOF
echo "The time is now `date`"
echo ""
echo ""
# Runs the commands
/usr/bin/ssh darcy@quake.geo.berkeley.edu < /home/ulfem/scripts/cmds.tmp
> /home/ulfem/scripts/cmds.tmp
exit 0
```

ulfem_get40hzRange.sh

```
# This script is like ulfem_getAscii.py, except it gets
# ASCII data from just channel, and sends it back to
# ulfem.stanford.edu. Also, this code is different because
# you can specify a time range down to the minute.
#
# Usage:
# python ulfem_get40hzAscii.py Site Channel_specifier YYYY.ddd,hh:mm YYYY.ddd,hh:mm
# The first time is when you want the data to start, and the second one is where you
# want it to end. This program deals with UTC time. Example:
# python ulfem_get40hzAscii.py BRIB BQ2 2009.020,02:00 2009.020,03:00.
# Notice: The "B" in "BQ2" is redundant because this script only deals with
# 40 hz data, but I left it there for readability/standardization purposes.

#gets ULFEM data from NCEDC, changes to ascii .d files


def zeroPadStr(x, fin_length):
	zro='0'
	xstr=str(int(x))
	if len(xstr)==fin_length:
		zps=xstr
	else:
		for i in range(0,fin_length-len(xstr)):
			xstr=zro+xstr

	return xstr

import re
import os
import sys
import time

print "Acquiring ASCII data from quake.geo.berkeley.edu:"

#determine start time, duration, and sampling rate of download TS
#ulfem_get40hzAscii.py site channel_specifier YYYY.ddd,hh:mm YYYY.ddd,hh:mm
#Notice: this script is designed to deal with 40 hz data, so if you
# chose "lt1," it would assume you meant "bt2."

site=sys.argv[1]
channel=sys.argv[2]
startTime=sys.argv[3]
endTime=sys.argv[4];
# this code breaks up YYYY.ddd,hh:mm into its components
startYear=startTime[0:4]
startDay=startTime[5:8]
startHour=startTime[9:11]
startMinute=startTime[12:14]
# Now for end times. Sounds ominous, doesn't it?
endYear=startTime[0:4]
endDay=startTime[5:8]
endHour=startTime[9:11]
```

```
endMinute=startTime[12:14]

remotePath='/home/ulfem/data/'+startYear+'/'+zeroPadStr(startDay,3)+'/'
scratchWorkPath='/home/dc1/darcy/scratch/ulfem/'
remoteServer='ulfem@ulfem.stanford.edu'
verbose_mode=True

#if verbose mode disabled, supress qdata and quan2asc chatter
suppressOut=''
if (not verbose_mode):
        suppressOut=' >/dev/null'

#runs remote script that checks to see if there's a directory on 'remoteServer'
#for 'startYear'. if not, it makes the directory.
os.system('ssh '+remoteServer+' sh /home/ulfem/data/manageDirs.sh '+startYear+' '+zeroPadStr(startDay,3))

#cleans scratch directory
os.system('/bin/rm -f '+scratchWorkPath+'*.tmp '+scratchWorkPath+'*.d' + suppressOut)

# desired format: qdata -f 2008.345,11:00 -t 2008.345,11:20 JRSC.BK.BQ2.10
fg='qdata -f '+startTime+' -t '+endTime+' -o "'+scratchWorkPath+'qdata.tmp'+\
 '" '+site+'.'+'BK'+'.'+channel

#TEMPORARY IRRITATING HACK:
dD='..D.'
if len(site)==4:
            dD='.D.'
            if re.search('Q',channel):
                        fg=fg+'.10'
                        dD='.10.D.'
            if re.search('T',channel):
                        fg=fg+'.20'
                        dD='.20.D.'
#END TEMPORARY IRRITIATING HACK

#Process quan then ascii files in scratchWorkPath
print fg
os.system(fg+suppressOut)
fg='quan2asc '+'-o '+scratchWorkPath+' '+\
 scratchWorkPath+'qdata.tmp'
print fg
os.system(fg+suppressOut)
removeTMP = '/usr/bin/rm -f '+scratchWorkPath+'*.tmp'
os.system(removeTMP + suppressOut)
time.sleep(1)

#Send .d files back to ulfem
fg='mv '+scratchWorkPath+'*.d '+scratchWorkPath+\
 site+'.'+channel+'.'+startTime+'-'+endTime+'.d'
print fg
os.system(fg);
remotePath='/home/ulfem/data/40hz/'
os.system('ls '+scratchWorkPath)
sendhome = 'scp '+scratchWorkPath+'*.d '+\
```

```
 remoteServer+':'+remotePath
print sendhome;
os.system(sendhome)
deletedees = '/usr/bin/rm -rf '+scratchWorkPath+'*.d'
os.system(deletedees)
print "\n"
#sys.exit()
```

ulfem_getAscii.py

```
#gets ULFEM data from NCEDC, changes to ascii .d files
#gets list of sites from /SITES/sites.lst
#usage: python ulfem_getAscii.py YYYY ddd
#example usage: python getAscii.py 2008 185 L/B
#this command gets 1 day of 1hz or 40hz data, if the last
#parameter is L or B, respectively.


def zeroPadStr(x, fin_length):
	zro='0'
	xstr=str(int(x))
	if len(xstr)==fin_length:
		zps=xstr
	else:
		for i in range(0,fin_length-len(xstr)):
			xstr=zro+xstr

	return xstr

import re
import os
import sys
import time

print "Acquiring ASCII data from quake.geo.berkeley.edu:"

#determine start time, duration, and sampling rate of download TS
year=sys.argv[1]
ddd=int(sys.argv[2])
sr=sys.argv[3]
dur='1d'
HH='00'
MM='00'

#get ULFEMpath from sys directory setPaths.whatever
#a = os.popen("echo $ULFEMpath")
#ULFEMpath='/scr/em1/kappler/ULFEM/'
ULFEMpath='/home/dc1/darcy/ulfem_scripts/'
remotePath='/home/ulfem/data/'+year
if sr=='L':
		remotePath+='_1hz/'+zeroPadStr(ddd,3)+'/'
elif sr=='B':
		remotePath+='_40hz/'+zeroPadStr(ddd,3)+'/'
scratchWorkPath='/home/dc1/darcy/scratch/ulfem/'
remoteServer='ulfem@ulfem.stanford.edu'

#get sites list from /sys/sites.lst
sitesFile='/home/dc1/darcy/ulfem_scripts/sites.lst'
f=open(sitesFile,'r')
sites=f.readlines()
f.close()
```

```
network='BK'

#runs remote script that checks to see if there's a directory on 'remoteServer'
#for 'year'. if not, it makes the directory.
os.system('ssh '+remoteServer+' sh /home/ulfem/data/manageDirs.sh '+year+' '+zeroPadStr(ddd,3))

#cleans scratch directory
os.system('/bin/rm -f '+scratchWorkPath+'*.tmp '+scratchWorkPath+'*.d')
sys.stdout.flush()

#Get data from all the sites
for site in sites:
  site=site[0:len(site)-1]
  fName=ULFEMpath+'channels/'+site+'.sns'
  cmd="f=open('"+fName+"','r')"
  exec(cmd)
  chList=f.readlines()
  f.close()
  for ch in chList:
        if len(ch)>1:  #avoid empty strings
              ch=ch[0:len(ch)-1]
              fg='qdata -f '+str(year)+'.'+zeroPadStr(ddd,3)+\
               '.'+HH+':'+MM+' -o "'+scratchWorkPath+'qdata.tmp'\
               +'" -s '+dur+' '+site+'.'+network+'.'+sr+ch

              ####TEMPORARY IRRITATING HACK:
              dD='..D.'
              if len(site)==4:
                    dD='.D.'
                    if re.search('Q',ch):
                          fg=fg+'.10'
                          dD='.10.D.'
                    if re.search('T',ch):
                          fg=fg+'.20'
                          dD='.20.D.'
              ####END TEMPORARY IRRITIATING HACK

              ###Process quan then ascii files in scratchWorkPath
              print fg
              sys.stdout.flush()
              os.system(fg)
              fg='quan2asc '+'-o '+scratchWorkPath+' '+\
               scratchWorkPath+'qdata.tmp'
              print fg
              sys.stdout.flush()
              os.system(fg)
              removeTMP = '/usr/bin/rm -f '+scratchWorkPath+'*.tmp'
              os.system(removeTMP)
              time.sleep(0.2)

              ###Send .d files back to ulfem
              rmCmd = '/usr/bin/rm -f '+scratchWorkPath
              remotePath='/home/ulfem/data/'+year
              if sr=='L':
```

```
            remotePath+='_1hz/'+zeroPadStr(ddd,3)+'/'
elif sr=='B':
            remotePath+='_40hz/'+zeroPadStr(ddd,3)+'/'
os.system('ls '+scratchWorkPath)
# These cmds remove files that don't end in
# 000000.d
sendhome = 'scp '+scratchWorkPath+'*000000.d '+\
 remoteServer+':'+remotePath
os.system(sendhome)
os.system(rmCmd+'*')
print "\n-----------------------------------\n"
sys.stdout.flush()
```

ulfem account cron jobs

```
SHELL=/bin/bash
#PATH=/sbin:/bin:/usr/sbin:/usr/bin
#MAILTO=root
#HOME=/home/ulfem

# Downloads the previous day's data to fill in any potential gaps
10 17 * * * /bin/sh /home/ulfem/scripts/getYesterdaysData.sh 20`date +\%y` `date +\%j` L

# Gets 1 Hz data every 30 minutes.
0,30 * * * * sh ~/scripts/remote_getAscii.sh 20`date +\%y` `date +\%j` L

# Deletes 40 Hz data files that were not accessed within the last 15 minutes
0 * * * * find ~/data/40hz/ -name "*.d" -type f -amin +15 -delete

# Deletes week-old files in the ~/data/userImg directory.
# Note: right now set to one day for test purposes.
0 * * * * find ~/data/userImg/ -name "*" -type f -ctime +7 -delete

# Deletes data directories older than 10 years.
0 0 1 1 * sh ~/scripts/decadizer.sh
```

Appendix B: Imageplot

Imageplot

```
function imageplot(stationI, channelI, starttimeI, endtimeI, UTCoffsetI,outputI,typeI)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function outputs an image file plotting ultra-low-frequency
% electromagnetic data, based on specified start and end times, stations,
% channels, and other preferences. Notice: if you want to generate a plot,
% it's recommended that you just use the Web interface, located at
% http://pangea.stanford.edu/research/group/crustal/ulfem/form.html.
% That will parse the arguments for you.
%
% Arguments:
% stationI: the station you want to plot, such as 'BRIB'. If you
%    want to plot multiple stations on one graph, enter in the format
%    {'BRIB','JRSC'}, or any combination you like. The program
%    automatically scales the image and stacks the subplots so that they're
%    parallel to one another.
% channelI: similar format as station. You can also input an array of
%    channels. Naturally, any permutations of stationI and channelI have to
%    be in the form n-by-one, one-by-n, or n-by-n.
% starttimeI is in the in format: YYYY.MM.DD.hh:mm:ss. Like the previous two
%    arguments, you can input an array of time.
% endtimeI: Second verse, similar to the the first.
% offsetI: the UTC offset. For instance, the offset for PDT time is -7.
% imgfilenameI: the name of the image file. Filename is foo outputs
%    foo.jpg.
% outputI: If this value is 'console', then it shows the graph on the
%    screen. Otherwise, it saves the file as insert_value_here.jpg
% typeI: pwr (power density), spec (spectrogram), or plot (plot!)
%
% Examples:
```

```
% imageplot('BRIB',{'LT1','LT2','LT3'},
%     '2008.8.1.3:00:00','2008.8.2.3:00:00',-7,'k','foo',1,0)
% OR
% imageplot({'MHDL','BRIB','SAO'},{'LT1','LT2','LT3'},
%     '2008.8.1.3:00:00','2008.8.2.3:00:00',-7,'k','foo',1,0)
% OR
% imageplot({'MHDL','BRIB'},'LT1',
%     '2008.8.1.3:00:00','2008.8.2.3:00:00',-7,'k','foo',1,1)
% for 8-1-2008 3:00:00AM - 8-2-2008 3:00:00AM PDT (3 AM = 10 AM UTC)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%

    % Turns off matlab warnings.
    warning off all;
    % Persistent variables are global within the m-file. They're used
    % to simplify argument passing. Ah, the persistence of memory.
    persistent station channel starttime endtime UTCoffset type;
    persistent UTCstart UTCoffsetElement UTCend UTCstartDay UTCendDay imgfilename data output;
    persistent startdatenum enddatenum datevector datavec loopnum showGraph;
    persistent channelElement stationElement startElement endElement typeElement;
    persistent CPUstartTime;
    % This puts all the parameters in the right formats (ie, strings become
    % numbers) and also puts them into persistent form.
    initiateParameters(stationI, channelI, starttimeI, endtimeI,...
        UTCoffsetI, outputI, typeI);
    % This function is the prime mover;
    makeEachSubplot();
    if(~showGraph), saveImage(); clearVars(); end


    function makeEachSubplot()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    % This function is the prime mover. It loops through, creating each
    % subplot and adds it to the figure. Ultimately, it controls every
    % function not called above.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
        for loopnum=1:1:length(channel) %#ok<FXUP>
            processTimeout(); %kills processes that have taken too long
        channelElement=channel{loopnum};
            stationElement=station{loopnum};
        UTCoffsetElement=str2num(UTCoffset{loopnum});
            startElement=starttime{loopnum};
            endElement=endtime{loopnum};
                    startdatenum = datenum(startElement);
                    enddatenum = datenum(endElement);
            typeElement=type{loopnum};
            getData();
            if strcmp(typeElement,'pwr')==1
                pwrData();
            elseif strcmp(typeElement,'plot')==1
                plotData();
            end
        end
    end

    function initiateParameters(stationI, channelI, starttimeI,...
        endtimeI, UTCoffsetI, outputI,typeI)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % this function takes the input arguments and transfers their
        % values to corresponding persistent variables, which are
        % global within the m-file. Also, the function formats the input
```

```
    % arguments, such as turning strings into numbers matlab can
    % handle.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CPUstartTime measures when the process starts so that it can cut
% off after a certain amount of time as defined in processTimeout()
CPUstartTime = cputime;
    station=processQuotes(stationI);
    channel=processQuotes(channelI);
    starttime=processQuotes(starttimeI);
    endtime=processQuotes(endtimeI);
    UTCoffset=processQuotes(UTCoffsetI);
    output=processQuotes(outputI);
    type=processQuotes(typeI);
    if (strcmp(output,'console'))
        showGraph = true;
        imgfilename = 'null';
    else
        showGraph = false;
        imgfilename = processQuotes(output);
    end
    %if (ischar(UTCoffset)), UTCoffset=str2num(UTCoffset); end;
    if (ischar(showGraph)), showGraph=str2num(showGraph); end;
    % turns station, channel inputs into cell arrays.
    station = str2cellarray(station,0);
    channel = str2cellarray(channel,0);
    starttime = str2cellarray(starttime,1);
    endtime = str2cellarray(endtime,1);
    type = str2cellarray(type,0);
UTCoffset = str2cellarray(UTCoffset,0);
    % If stationI is input as a string, the program assumes that it was
    % called as an executable, and doesn't show a visible plot.
    matchInputLength();
    % whether or not to show the graph
    if(showGraph==0)
        figure('visible','off');
    else
        figure(1);
    end
end

% I was working on matchInput length, which makes the station, channel, and
% time inputs cell arrays of the same length. Note: I need to change
% dateVectorize to datestr2cellarray, which turns '{datestr1,datestr2}'
% string into a cell array. The imageplot function should only process date
% strings now. Note 2: there aren't any applications that require other
% types of input, are there?

function matchInputLength()

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This finds the longest cell array inputted, and makes all the other
% cell arrays the same length, inserting the first value into the new
% spots. If you input {'brib','jrsc'} and {'LT1'}, it'd spit out
% {'brib','jrsc'} and {'LT1','LT1'}.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    maxlen=max([ length(channel) length(station) length(starttime) length(endtime) ]);
    station(1,length(station)+1:maxlen)={station{1,1}};
    channel(1,length(channel)+1:maxlen)={channel{1,1}};
    starttime(1,length(starttime)+1:maxlen)={starttime{1,1}};
    endtime(1,length(endtime)+1:maxlen)={endtime{1,1}};
end

function str = processQuotes(str)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% This replaces double quotes with single quotes, because that's
% what matlab wants. TO DO: this should be done with
    % input = regexprep(input,"","");. I have no time at the moment!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % If 'str' is not a string, then there's no quotes to remove!
    varClass = whos('str');
    varClass = varClass.class;
    if (~strcmp(varClass,'string'))
        return;
    end
    quotes=strfind(str,"");
    % If there's no quotes, then it's fine
    if (isempty(quotes))
        return;
    end
    % otherwise, find and replace " with '.
    str(quotes)="";
    if (quotes(1)==1 && quotes(length(quotes))==length(str))
        str=str(2:length(str)-1);
    end
end

function dv = dateVectorize(in)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % This function turns a date vector string or a string of the
    % format YYYY.MM.DD.mm:hh:ss into a date vector. The latter format
    % was created due to errors involved with sending a date vector
    % string through a shell script.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % If there's brackets in it, it's probably the string
    % representation of a date vector
    if(~isempty(strfind(in,'[')))
        dv=str2num(in); %#ok<ST2NM>
    % otherwise, it's in the format YYYY.MM.DD.mm:hh:ss. This parses
    % that format into the date vector format.
    else
    % finds indices of dots, spaces, and colons, in array form
    dots=strfind(in,'.');
    colons=strfind(in,':');
        dv = [in(1:dots(1)-1) ' ' ...
              in(dots(1)+1:dots(2)-1) ' ' ...
              in(dots(2)+1:dots(3)-1) ' ' ...
              in(dots(3)+1:colons(1)-1) ' ' ...
              in(colons(1)+1:colons(2)-1) ' ' ...
              in(colons(2)+1:length(in))];
        dv = str2num(dv);
    end
end

function strCell = str2cellarray(input,isTime)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Turns string, '{"PKD","BRIB"}' into a cell array, similar
% to how str2int changes strings into integers.
% If it's dealing with time input, isTime needs to be set to '1' so
% that it can dateVectorize the times.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    % If it's already a cell array, our work here is done.
    if (iscell(input))
        strCell=input;
        if (isTime)
```

```
            for i=1:length(strCell)
               strCell{i} = dateVectorize(strCell{i});
            end
        end
        return;
    end
    % strip apostrophes
    input = regexprep(input,'''','');
    % strip curly braces
    input = regexprep(input,'{','');
    input = regexprep(input,'}','');
    % find commas
    commas = strfind(input,',');
    % preallocate strCell size
    strCell = cell(1,length(commas)+1);
    % parses list, now just in the form this,is,my,list. Puts each
    % element separated by a comma into its own cell.
    startIndex = 1;
    for i=1:1:length(strCell);
       if ( i > length(commas) )
          stopIndex = length(input);
       else
           stopIndex = commas(i)-1;
       end
       str=input(1,startIndex:stopIndex);
       startIndex = stopIndex+2;
       if(isTime)
           str = dateVectorize(str);
       end;
       strCell(1,i)={str};
    end
end

function getData()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% getData finds the right d files from /home/ulfem/data/, strings them
% together if necessary. Also, it produces the data vector
% for plotting.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (startdatenum >= enddatenum)
        error('*** Error: you have entered  an invalid range of dates. Please go back and try again. ***');
    end
    UTCstart = startdatenum - UTCoffsetElement/24;
    UTCend = enddatenum - UTCoffsetElement/24;
    UTCstartDay = dayOfYear(UTCstart);
UTCendDay = dayOfYear(UTCend);
    if strcmp(typeElement,'plot')==1
        get1hzData();
    else
        get40hzData();
    end
end

function get40hzData()
    channelElement = ['B' channelElement];
    fortyHzDir = '/home/ulfem/data/40hz/';
    % gets the contents of the forty hz directory
    filestruct = dir(fortyHzDir);
    % deletes entries for first two "files," which are just "." and
    % ".."
    filename='';
    fileStart=0;
    fileEnd=0;
```

```
0276        works=0;
0277        % Loops through the data direcory, looking for a file that has data
0278        % for the appropriate channel, for the appropriate date range.
0279        for k=1:size(filestruct)
0280            filename = filestruct(k).name;
0281            [works, fileStart,fileEnd] = findFile(filename);
0282            if(works==1)
0283                break;
0284            end
0285        end
0286        if(works==0)
0287            error('*** Error: cannot find appropriate 40hz data. ***');
0288        end
0289        datavec = textread([fortyHzDir filename],'%f','headerlines',1);
0290        startSample = round((UTCstart-fileStart)*86400*40)+1;
0291        tickInterval=365*24*60;
0292        endSample = round(startSample+(UTCend-UTCstart)*86400*40)-1;
0293        if (size(datavec,1) < endSample)
0294            error(['*** Error reading downloaded 40 hz data. ***'...
0295                ' End sample exceeds start sample by ' ...
0296                num2str(endSample-size(datavec,1))]);
0297        end
0298        datavec = datavec(startSample:endSample);
0299    end
0300
0301    function [works,filestart,fileend] = findFile(filename)
0302    % this takes a file with a name of the format
0303    % JRSC.BQ2.2009.079,00:03-2009.079,10:03.d
0304    % and returns the start and end times
0305        filestart = -1;
0306        fileend = -1;
0307        dots=strfind(filename,'.');
0308        commas=strfind(filename,',');
0309        dash=strfind(filename,'-');
0310        colons=strfind(filename,':');
0311        % if the file is not named correctly, it passes over it
0312        if ((size(dots,2)~=5) || (size(commas,2)~=2) || (size(dash,2)~=1))
0313            works = 0;
0314            return;
0315        end
0316        fStation = filename(1:dots(1)-1);
0317        fChan    = filename(dots(1)+1:dots(2)-1);
0318        fsYear   = filename(dots(2)+1:dots(3)-1);
0319        fsDay    = filename(dots(3)+1:commas(1)-1);
0320        fsHour   = filename(commas(1)+1:colons(1)-1);
0321        fsMin    = filename(colons(1)+1:dash(1)-1);
0322        feYear   = filename(dash(1)+1:dots(4)-1);
0323        feDay    = filename(dots(4)+1:commas(2)-1);
0324        feHour   = filename(commas(2)+1:colons(2)-1);
0325        feMin    = filename(colons(2)+1:dots(5)-1);
0326        datenum(['01-Jan-' fsYear ' ' fsHour ':' fsMin])
0327        if(strcmp(fChan,channelElement)==0 || strcmp(fStation,stationElement)==0)
0328            disp(['Wrong channel or station for ' filename]);
0329            works=0;
0330            return;
0331        end
0332        filestart = datenum(['01-Jan-' fsYear ' ' fsHour ':' fsMin]) + str2num(fsDay)-1;
0333        fileend = datenum(['01-Jan-' feYear ' ' feHour ':' feMin]) + str2num(feDay)-1;
0334        if ~( UTCstart - filestart > -0.01 && fileend - UTCend > -0.01)
0335            disp(['Wrong times for ' filename '. Filestart, end; required start, end ' datestr(filestart) ' ' datestr(fileend) ' ' datestr(UTCstart) ' '
datestr(UTCend) ' ' ]);
0336            works=0;
0337            return;
0338        end
0339        works=1;
0340    end
0341
```

```
function get1hzData()
    channelElement = ['L' channelElement];
    % This gets 1hz data in d files from the  ~/data directory
    numDays = floor(enddatenum)-floor(startdatenum)+1;
data = nan(86400,numDays);
    % Location of d files specified below
    dataDir='/home/ulfem/data/';
    % This loop sequentially reads in the .d files.
    for k=0:1:(numDays-1)
    processTimeout(); %kills processes that have gone on too long
    dv = datevec(startdatenum+k);
    year = dv(1);
    doy = zeroPadNum(floor(datenum(startdatenum+k-datenum([year 00 00 00 00 00]))));
        yearStr = num2str(year);
        doyStr = num2str(doy);
        % if you're plotting, this uses 1hz data; otherwise, 40hz.
        filePath = strcat(dataDir,yearStr,'_','1hz/',doyStr,'/');
        fileName = strcat(stationElement,'.',channelElement,'.',yearStr(3:4),doyStr,'000000.d');
        fileLoc  = strcat(filePath,fileName);
        % next two lines put readin in a matrix that's 86400x1, even if
        % there isn't a complete data set.
        readin = nan(86400,1);
        if (exist(fileLoc,'file')==0)
            disp(['*** Warning: Data for ' stationElement ' ' channelElement ' on ' ...
                datestr(startdatenum+k,'dd-mmm-yyyy') ' UTC '...
                'does not exist on our servers. '...
                '*** Filename: ' fileLoc]);
        else
            %disp(['Using ' fileName]);
    readin2 = textread(fileLoc,'%f','headerlines',1);
            readin(1:size(readin2,1))=readin2;
        end
        data(:,k+1) = readin;
    end
    startsec = round((UTCstart - floor(UTCstart))*86400);
    endsec = startsec + round((UTCend-UTCstart)*86400);
datavec = data(startsec+1:endsec+1);
    if isnan(datavec)
       datavec(1)=0;
   %error('*** Error: sorry, no data exists on our servers for that channel and date range. ***');
    end
    datevector = linspace(startdatenum,enddatenum,length(datavec));
end

function doy = dayOfYear(inDate)
% turns datenum into day of year in numerical form
    year = str2num(datestr(inDate,'yyyy'));
    doy = floor(inDate - datenum([year 0 0 0 0 0]));
end

function str = zeroPadNum(num)
    str = num2str(num);
    if (num < 10)
        str = ['00' str];
    elseif (num < 100)
        str = ['0' num2str(num)];
    end
end

function plotData()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plotData plots the graphs and takes care of the cosmetics.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        subplot(length(channel),1,loopnum), plot(datevector,datavec)
        % The following bases the y-range on the standard deviation of the
        % data, so that the y-range isn't dependent on spikes.
        stdDev = std(datavec(~isnan(datavec)));
        % show 5 stdDevs above and below the average
        halfRange = 8*stdDev;
        % this is a work-around to fix the axis if there's no data
        if(halfRange == 0)
            halfRange = 1;
        end
        avg = mean(datavec(~isnan(datavec)));
        axis('tight');
        ax = axis;
        axis([ ax(1:2) avg-halfRange avg+halfRange]);
        % This puts the ticks in places that are human-friendly,
         % ie, every 30 minutes.
        drawTicks(gca,5);
        set(gca,'XGrid','on');
%       rotateticklabel(gca,-15);
        labelGraph();
    end

    function pwrData()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    % pwrData makes a spectrogram
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    windowSize=2^14;
        numWindows=300;
        frequencyBins=128;
        windowSpacing=floor( size(datavec,1)/numWindows - windowSize );
        F = logspace(-2,1,frequencyBins);
        disp(F(1:10)');
        %disp(['Number of samples: ' num2str(size(datavec,1)) ' <br>']);
        %disp(['Window size: ' num2str(windowSize) ' <br>']);
        %disp(['Number of windows: ' num2str(numWindows) ' <br>']);
        %disp(['Frequency bins: ' num2str(frequencyBins) ' <br>']);
        %disp(['Window spacing: ' num2str(windowSpacing) ' <br>']);
        %tic
        [S,f,t,p] = spectrogram(datavec,windowSize,-windowSpacing,F,40,'yaxis');
        %toc
        set(gcf,'renderer', 'zbuffer');
        % t is output as an array of seconds. This converts it to an array
        % of datenums, given the correct starting time.
        t = t/86400 + startdatenum;
        subplot(length(channel),1,loopnum), surf(t,f,10*log10(abs(p)),'EdgeColor','none');
        colorbar;
        axis xy; axis tight; colormap(jet); view(0,90);
        set(gca, 'YScale', 'log');
        % This puts the ticks in places that are human-friendly,
         % ie, every 30 minutes
        drawTicks(gca,5);
    end

    function labelGraph()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    % labels the graph!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
        if (str2num(datestr(startdatenum,'dd'))==str2num(datestr(enddatenum,'dd')))
            daysString=[' on ' datestr(startdatenum,'dd-mmm-yyyy')];
```

```
    else
        daysString=[' from ' datestr(startdatenum,'dd-mmm-yyyy') ' to ' datestr(enddatenum,'dd-mmm-yyyy')];
    end
    title(['Activity of ' channelElement ' at ' stationElement daysString]);
    timezoneName='';
if (UTCoffsetElement==-8)
    timezoneName = 'PST';
elseif (UTCoffsetElement==-7)
    timezoneName = 'PDT';
end
xlabel(['Time (UTC' num2str(UTCoffsetElement) ' - ' timezoneName ')']);
    if strcmp(typeElement,'pwr')==1
        ylabel('Frequency (hz)');
    else
        ylabel('Machine counts');
    end
    alltext=findall(1,'type','text');
    set(alltext,'FontSize',15);
    %set(alltext,'fontunits','normalized');
end

function drawTicks(gca,numTicks)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This puts the ticks in places that are human-friendly,
% ie, every 30 minutes. Based on the array minDivs (see below), it
% tries to find the type of division that most nearly results in having
% the number of ticks specified by numTicks. For instance, it might
% choose to plot ticks every 1 minute, 30 minutes, or 1 month. If
% things aren't turning out the way you like, the best way is probably
% to add more minute divisions, so the program has more to work with.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    oneMin = datenum([0000 00 00 00 01 00]); % one minute in serial date time
    % this puts dates into minutes to make things easier
    startdatemin = startdatenum/oneMin;
    enddatemin = enddatenum/oneMin;
    % interval between labeled ticks
    ltickInterval = findTickInterval(numTicks);
    % where the labeled ticks should start
    ltickStart = ltickInterval*ceil(startdatemin/ltickInterval);

%         if (ltickInterval < 1*60) || (ltickInterval > 48*60)
%             ltickArray = (ltickStart:ltickInterval:enddatemin)*oneMin;
%             set(gca,'XTick',ltickArray);
%             datetick('x',getDateFormat(startdatenum,enddatenum),'keeplimits','keepticks');
%         else
        if (ltickInterval < 60)
            tickInterval = ltickInterval;
    elseif (ltickInterval < 12*60)
    tickInterval = 60;
        elseif (ltickInterval==12*60)
            tickInterval=2*60;
        elseif (ltickInterval<=24*60)
            tickInterval=6*60;
        elseif (ltickInterval<=48*60)
            tickInterval=12*60;
        elseif (ltickInterval<=8*24*60)
    tickInterval=48*60;
    elseif (ltickInterval<=30*24*60)
    tickInterval=10*24*60;
    elseif (ltickInterval<=3*30*24*60)
     tickInterval=30*24*60
    elseif (ltickInterval<=365*24*60)
    tickInterval=365*24*15;
```

```
0527          end
0528          tickStart = tickInterval*ceil(startdatemin/tickInterval);
0529          tickArray = (tickStart:tickInterval:enddatemin)*oneMin;
0530          set(gca,'XTick',tickArray);
0531          datetick('x',getDateFormat(startdatenum,enddatenum),'keeplimits','keepticks');
0532          tOffset = 0;
0533          % finds tick offset
0534          for k=1:size(tickArray,2)
0535              % this should be ==, but I changed it to < to fix
0536              % rounding issues.
0537              if (ltickStart - tickArray(1,k)/oneMin) < .0001
0538                  tOffset = k;
0539                  break;
0540              end
0541          end
0542
0543          xticks=get(gca,'XTickLabel');
0544          newxticks(1,1:size(xticks,2)) = ' ';
0545          for k=1:size(xticks,1)
0546              newxticks(k,:)=' ';
0547          end
0548          for k=tOffset:(ltickInterval/tickInterval):size(xticks,1)
0549              newxticks(k,:)=xticks(k,:);
0550          end
0551          set(gca,'XTickLabel',newxticks);
0552      end
0553
0554      function tickInterval = findTickInterval(numTicks)
0555          oneMin = datenum([0000 00 00 00 01 00]); % one minute in serial date time
0556          interval=(enddatenum-startdatenum)/oneMin;
0557          % minDivs are the different tick intervals the program can choose.
0558          % For instance, 1 = one minute, 5 =  ***five minutes, 4*7*24*60 = 40320
0559          % minutes, or one month.
0560          minDivs =  [1 2 5 10 15 30 1*60 2*60 3*60 4*60 6*60 12*60 24*60 48*60 8*24*60 ...
0561                      30*24*60 3*30*24*60 365*24*15];
0562          % This function chooses the minute division that results in the
0563          % number of ticks that most closely matches numTicks. "Junk" is a
0564          % throwaway variable, needed to get minDiv Index.
0565          % for.
0566          [junk minDivIndex] = min(abs(minDivs*numTicks - interval));
0567          % tickInterval: number of minutes between ticks.
0568          tickInterval = minDivs(minDivIndex);
0569      end
0570
0571      function th=rotateticklabel(h,rot,demo)
0572      %ROTATETICKLABEL rotates tick labels    %   TH=ROTATETICKLABEL(H,ROT) is the calling form where H is a handle to
0573      %   the axis that contains the XTickLabels that are to be rotated. ROT is
0574      %   an optional parameter that specifies the angle of rotation. The default
0575      %   angle is 90. TH is a handle to the text objects created. For long
0576      %   strings such as those produced by datetick, you may have to adjust the
0577      %   position of the axes so the labels don't get cut off.
0578      %
0579      %   Of course, GCA can be substituted for H if desired.
0580      %
0581      %   TH=ROTATETICKLABEL([],[],'demo') shows a demo figure.
0582      %
0583      %   Known deficiencies: if tick labels are raised to a power, the power
0584      %   will be lost after rotation.
0585      %
0586      %   See also datetick.
0587
0588      %   Written Oct 14, 2005 by Andy Bliss
0589      %   Copyright 2005 by Andy Bliss
0590
0591      %DEMO:
0592      if nargin==3
0593          x=[now-.7 now-.3 now];
```

```
    y=[20 35 15];
    figure
    plot(x,y,'.-')
    datetick('x',0,'keepticks')
    h=gca;
    set(h,'position',[0.13 0.35 0.775 0.55])
    rot=90;
end

%set the default rotation if user doesn't specify
if nargin==1
    rot=90;
end
%make sure the rotation is in the range 0:360 (brute force method)
while rot>360
    rot=rot-360;
end
while rot<0
    rot=rot+360;
end
%get current tick labels
a=get(h,'XTickLabel');
%erase current tick labels from figure
set(h,'XTickLabel',[]);
%get tick label positions
b=get(h,'XTick');
c=get(h,'YTick');
%make new tick labels
if rot<180
    th=text(b,repmat(c(1)-.1*(c(2)-c(1)),length(b),1),a,'HorizontalAlignment','right','rotation',rot);
else
    th=text(b,repmat(c(1)-.1*(c(2)-c(1)),length(b),1),a,'HorizontalAlignment','left','rotation',rot);
end
set(th,'FontSize',26)
end

function saveImage()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% saveImage outputs the image file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %the following code is a work-around to be able to specify image resolution
    %IMPT: instructs MATLAB to print using the same size as that shown on the screen
    imagedir = '/home/ulfem/data/userImg/';
    set(1,'PaperPositionMode','auto');
    %Resize figure Window
    if strcmp(typeElement,'pwr')
        set(1, 'Position',[0 0 1000 500*length(channel)]);    %Resize figure Window
    else
        set(1, 'Position',[0 0 1000 200*length(channel)]);    %Resize figure Window
    end
    %---print figure---%
    %print('-djpeg90',[imgfilename '.jpg']);
    saveas(gcf,[imagedir imgfilename '.png'],'png');
    %save([imgfilename '.mat']);
    disp('image saved.');
end

function dateFormat = getDateFormat(startdatenum,enddatenum)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function finds the best format for the ticks based on the
% length of time plotted. If it's plotting a few hours, then would do
```

```
% HH:MM PM, a few days, dd-mm-yyy HH:MM:SS. This is imperfect, though,
% due to the limited amount of matlab date strings. TO DO: look into
% improving this?

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    oneMin = datenum([0000 00 00 00 01 00]); % one minute in serial date time
    startdatenum = startdatenum/oneMin; % converts time into
    enddatenum = enddatenum/oneMin;     % minute units
    interval=enddatenum-startdatenum;
    % Makes date labels that are consisent with the ranges. If you're just looking
% a few minutes, it gives HH:MM. A few years, dd-mmm-yyyy.
    if (interval >= 175*24*60)
    dateFormat = 'dd-mmm-yyyy';
elseif (interval >= 24*60)
        dateFormat = 'dd-mmm HH:MM';
    elseif (interval > 2)
        dateFormat = 'HH:MM';
    else
        dateFormat = 'HH:MM:SS';
    end
end

function processTimeout()
% this function tests to see if the script has been running longer than 3 minutes.
% If so, it cuts off.
   processTime = cputime - CPUstartTime;
   if (processTime > 5*60)
  error('*** Error: this program has used more than 5 minutes of CPU time and must be terminated. Please try analyzing less data and try again ***');
   end
end

function clearVars()
    close all;
    clear all;
end
end
```

Appendix C: Web interface scripts

Process.php

```php
<?php
// process.php takes all the user parameters supplied by index.html,
// sends them to a matlab executable on ulfem.stanford.edu (in mitchell, rm 463), and then
// gets the image that ulfem produces.
ini_set('display_errors', '1');
error_reporting(E_ALL);
echo "<html><head>\n".
   "<title>ULFEM Data</title>\n".
   " <SCRIPT language=\"JavaScript\">function submitform(){document.contactform.submit();}</SCRIPT>\n".
         "<script type=\"text/javascript\" src=\"js/googleAnalytics.js\"></script>\n".
   "</head><body>\n";
$con = connect();
$imageplotDir="/home/ulfem/imageplot/";
$ulfemWebDir="/WWW/research/groups/crustal/ulfem/";
$outstart = str_replace(" ","-",$_POST["starts"][0]);
$outend   = str_replace(" ","-",$_POST["ends"][0]);
```

```
$outfile   = implode("_",$_POST["stations"]) . "_" . implode("_",$_POST["channels"]) . "[" . $outstart . "-" .
$outend . "]";
//require('40hzDownloader.php');
//get40hz($con);
$MatArgs = processMatArgs($outfile);
// This function is the prime mover! It connects to the remote computer,
// sends the commands, and gets the images
echo getImage($MatArgs,$outfile,$imageplotDir,$ulfemWebDir,$con);
// Google analytics code
echo '<script type="text/javascript">'.
        'var gaJsHost = (("https:" == document.location.protocol) ? "https://ssl." : "http://www.");'.
        'document.write(unescape("%3Cscript src=\'" + gaJsHost + "google-analytics.com/ga.js\'
type=\'text/javascript\'%3E%3C/script%3E"));'.
        '</script>'.
        '<script type="text/javascript">'.
        'try {'.
        'var pageTracker = _gat._getTracker("UA-10117039-1");'.
        'pageTracker._trackPageview();'.
        '} catch(err) {}</script>'.
        '</body></html>';

#################################################################
##########       HELPER FUNCTIONS       ################
#################################################################

function processMatArgs($outfile){
        //this function takes the post data and generates the matlab parameter string.
        //example: '{'BRIB','BRIB'}' '{'Q2','Q3'}' '{'2009.08.01.11:39:00','2009.08.01.11:39:00'}'
        //                              '{'2009.08.02.11:39:00','2009.08.02.11:39:00'}' '-7'
        //                              'BRIB_BRIB_Q2_Q3[2009.08.01-11:39:00-2009.08.02-11:39:00]' '{'plot','plot'}'
        $stations = $_POST["stations"];
        $channels = $_POST["channels"];
        $starts = $_POST["starts"];
        $ends = $_POST["ends"];
        $offsets = $_POST["offsets"];
        $types = $_POST["types"];
        // this defines the electric channels for each station
        $stats = array("BRIB","JRSC","MHDL","PKD","SAO");
        $statQ = array("BRIB"=> array("Q2","Q3","Q5","Q6"),
                                "JRSC"=> array("Q2","Q3","Q5","Q6"),
                                "MHDL"=> array(), //no q for mhdl :(
                                "PKD"=> array("Q2","Q3","Q4","Q5"),
                                "SAO"=> array("Q2","Q3"));
        // All stations have T1,T2,and T3 channels, so we don't need
        // a "statT" matrix.
        // This replaces an "all stations" selection with the specific stations.
        for($i=0; $i<sizeof($stations); $i++){
                if (strcmp($stations[$i],"all")==0){
                        //deletes "all" from the array"
                        unset($stations[$i]);
                        //renumbers array
                        $stations = array_values($stations);
                        for($j=0; $j<sizeof($stats); $j++){
                                array_insert($stations,$stats[$j],$j+$i);
```

```
                        if ($j>0){
                                array_insert($channels,$channels[$i],$j+$i);
                                array_insert($starts,$starts[$i],$j+$i);
                                array_insert($ends,$ends[$i],$j+$i);
                                array_insert($offsets,$offsets[$i],$j+$i);
                                array_insert($types,$types[$i],$j+$i);
                        }
                }
                $i = $i+sizeof($stats)-1;
        }
}
// This replaces "all" with "T" and "Q" so that the next two
// loops can then insert the specific channels.
for($i=0; $i<sizeof($channels); $i++){
        if (strcmp($channels[$i],"all")==0){
                //deletes "all" from the array"
                unset($channels[$i]);
                //renumbers array
                $channels = array_values($channels);
                array_insert($channels,"T",$i);
                array_insert($channels,"Q",$i+1);
                array_insert($stations,$stations[$i],$i+1);
                array_insert($starts,$starts[$i],$i+1);
                array_insert($ends,$ends[$i],$i+1);
                array_insert($offsets,$offsets[$i],$i+1);
                array_insert($types,$types[$i],$i+1);
                $i=$i+1;
        }
}
//This loop replaces "T" with T1, T2, and T3
for($i=0; $i<sizeof($channels); $i++){
        if (strcmp($channels[$i],"T")==0){
                //deletes "T" from the array"
                unset($channels[$i]);
                //renumbers array
                $channels = array_values($channels);
                for($j=0; $j<3; $j++){
                        $chanNum = $j+1;
                        array_insert($channels,"T$chanNum",$j+$i);
                        if ($j>0){
                                array_insert($stations,$stations[$i],$j+$i);
                                array_insert($starts,$starts[$i],$j+$i);
                                array_insert($ends,$ends[$i],$j+$i);
                                array_insert($offsets,$offsets[$i],$j+$i);
                                array_insert($types,$types[$i],$j+$i);
                        }
                }
                $i=$i+2;
        }
}

//This loop replaces "Q" with the appropriate electric channels
//print_r($channels);echo "<br><br>";
//print_r($stations);echo "<br><br><hr><br><br>";
```

```
	for($i=0; $i<sizeof($channels); $i++){
		if (strcmp($channels[$i],"Q")==0){
			$qArr = $statQ[$stations[$i]];
			//deletes "Q" from the array"
			unset($channels[$i]);
			//renumbers array
			$channels = array_values($channels);
			if (sizeof($qArr)>0){
				for($j=0; $j<sizeof($qArr); $j++){
					array_insert($channels,$qArr[$j],$j+$i);
					if ($j>0){
						array_insert($stations,$stations[$i],$j+$i);
						array_insert($starts,$starts[$i],$j+$i);
						array_insert($ends,$ends[$i],$j+$i);
						array_insert($offsets,$offsets[$i],$j+$i);
						array_insert($types,$types[$i],$j+$i);
					}
				}
				$i = $i+sizeof($qArr)-1;
			}
			//if there are no q channels, it deletes all the entires at that index.
			else {
				unset($stations[$i]);
				$stations = array_values($stations);
				unset($starts[$i]);
				$starts = array_values($starts);
				unset($ends[$i]);
				$ends = array_values($ends);
				unset($offsets[$i]);
				$offsets = array_values($offsets);
				unset($types[$i]);
				$types = array_values($types);
				$i--;
			}
		}
		//print_r($channels);echo "<br><br>";
		//print_r($stations);echo "<br><br><hr><br><br>";
	}
	//I use replace_str to convert "2009.08.02 01:08:00" to "2009.08.02.01:08:00"
	$paramStr = makeParamStr($stations).' '.
		makeParamStr($channels).' '.
		str_replace(' ','.',makeParamStr($starts)).' '.
		str_replace(' ','.',makeParamStr($ends)).' '.
		makeParamStr($offsets).' '.
		//"\"-7\" ".' '.
		"\"$outfile\" ".' '.
		makeParamStr($types);
	if (sizeof($channels)>15)
		returnError("<b>Error: this Web site cannot currently handle more than 15 plots at a time. Please try requesting plots separately.</b><br><br>".detailsAndContactUs("The MATLAB script was not invoked because there were too many parameters.",$paramStr,"If you have any questions or comments, please"));
	return $paramStr;
}
```

```
function array_insert(&$array,$val,$pos){
   $array2 = array_splice($array,$pos);
   $array[] = $val;
   $array = array_merge($array,$array2);
   return $array;
}

function makeParamStr($array){
        $retStr="\"{";
        for ($k=0; $k<sizeof($array); $k++){
                $suffix="','";
                if ($k==sizeof($array)-1)
                        $suffix="'}\"";
                $retStr=$retStr.$array[$k].$suffix;
        }
        return $retStr;
}

function getImage($MatArgs,$outFile,$imageplotDir,$ulfemWebDir,$con){
        // SSH'es into ulfem.stanford.edu, remotely runs a matlab executable to generate
        // a plot specified by $MatArgs, copies it back to the Web server, and then outputs the image file name.
        $con = connect() or returnError('Cannot connect to remote server!');
        $remoteCommand  = 'cd '.$imageplotDir.'; ';
        $remoteCommand .= 'sh '.$imageplotDir.'run_imageplot.sh /usr/local/MATLAB ' . $MatArgs;
        $remoteCommand .= '  2>&1;';
        // Runs remoteCommand on ulfem.stanford.edu
        $stream = ssh2_exec($con,$remoteCommand) or
                returnError("Unable to run command on the data server. RemoteCommand:
".$remoteCommand);
        $data = processStream($stream);
        findMatlabError($data,$remoteCommand);
        return "<h1>Here are the requested plots:</h1><br>\n".
                "<img src=\"data/userImg/$outFile.png\"><br><br>".handleWarnings($data).
                //"<br>Click <a href=\"userGenMat/$outFile.mat\">here</a> to download the MATLAB file<br> ".
                detailsAndContactUs($data,$remoteCommand,"If you have any questions or comments,
please");
}

function fixDate($inDate){
        // This turns YYYY.MM.DD hh:mm:ss into YYYY.MM.DD.hh:mm:ss. This php script is passed the
        // former format so the user input is more readable.
        $inDate = str_replace(' ','.',$inDate);
        return addQuotes($inDate);
}

function addQuotes($inStr){
        // The addQuotes function turns This Is My Example String into "This Is My Example String"
  return stripslashes('\"' . $inStr . '\"');
}

function processStream($stream){
        // This collects the text output from the matlab program
        stream_set_blocking( $stream, true );
        $data = "";
```

```
        while( $buf = fread($stream,4096) ){
                $data .= $buf;
                $data .= "\n";
        }
        fclose($stream);
        return $data;
}

function connect(){
        // Try to initate connection to ulfem.stanford.edu
        if(!($con = ssh2_connect("ulfem.stanford.edu", 22,array("hostkey"=>"ssh-rsa")))){
                // returnError emails the Web master with errors. You can change the contact email
                // address within the returnError code.
                returnError('Unable to establish ssh2 connection');
                return false;
        } else {
                // Try to log  into ulfem.stanford.edu with username ulfem, password ulf-mag00.
                // Note: make this use a public/private key connection!
                if(!ssh2_auth_pubkey_file($con, "ulfem", "/home/sesweb/.ssh/id_dsa.pub",
"/home/sesweb/.ssh/id_dsa")) {
                returnError('Authentication Failed');
                        return false;
                }
        }
        return $con;
}

function handleWarnings($matOutStr){
        $retStr = "";
        $warnStrStop = 0;
        while(true){
                $startIndex = stripos($matOutStr,'*** Warning: ',$warnStrStop+3);
                if ($startIndex !== false){
                        if ($warnStrStop==0)
                                $retStr .= "<b>Matlab warnings:</b><br>";
                        $warnStrStart = $startIndex + 13;
                        $warnStrStop  = stripos($matOutStr,'***',$warnStrStart);
                        $warnStr = substr($matOutStr,$warnStrStart,$warnStrStop-$warnStrStart)."<br>";
                        $retStr .= $warnStr;
                }
                else return $retStr;
        }
}

function findMatlabError($outputString,$remoteCommand){
        // This searches the text that matlab outputs for the string "error." If it exists,
        // then there has been an internal matlab error, which it displays in a user-friendly
        // format.
        // if it cannot find the string "error," it returns false
        if (stripos($outputString,'error') === false && stripos($outputString,'???') === false){
                return;
        }
        // If matlab outputs 'required d file', then it means that a d file isn't available,
        // so the program tells the user data isn't available.
```

```
        // TO DO: tell which file!
        $startIndex = stripos($outputString,'*** Error');
        if ($startIndex !== false){
            $errorStringStart = $startIndex + 4;
            $errorStringStop  = stripos($outputString,'***',$errorStringStart);
            $errorString = substr($outputString,$errorStringStart,$errorStringStop-$errorStringStart);
            $errorMessage = "<b>".$errorString."</b><br><br>\n";
                returnError($errorMessage.detailsAndContactUs($outputString,$remoteCommand,
                        "If you believe this error is due to a technical glitch, please"));
        }
        returnError("<b>MATLAB error!<br><br>".detailsAndContactUs($remoteCommand,$outputString,
                        "If you believe this error is due to a technical glitch, please"));
}


// If there's an error, it shows an error screen
function returnError($errorMessage){
        echo "<h1>Error!</h1><img src=img/error.jpg><br><br>";
        //$errorMessage = ereg_replace("\n",'<br>',$errorMessage);
        echo $errorMessage."<br><br>\n";
        exit();
}

function detailsAndContactUs($outputString,$remoteCommand,$linkMessage){
        $details = "<b>Remote command:</b><br>$remoteCommand<br><br><b>Matlab
output:</b>\n$outputString";
        $details = ereg_replace("\n","<br>",$details);
                //ereg_replace("<","&lt;",$details).
                //ereg_replace(">","&gt;",$details).
                $details = ereg_replace("&","(ampersand)",$details);
                $details = ereg_replace("\"","&#39;",$details);
    $outputString = "<form name=\"contactform\" action=\"contactus.php\" method=\"POST\">".
                "<input type='hidden' name='matlab' value=\"$details\">".
                "$linkMessage <A href=\"javascript:submitform()\">contact us.</A>\n</form>\n\n";
        $details = ereg_replace(" ","+",$details);
        $detlink = "Click <a href=\"echo.php?text=$details\" target=\"_blank\">here</a> for more details.";
        $outputString .= $detlink;
        return $outputString;
}

?>
```