MWLIB
A Multitaper Spectrum Estimation Library

Germán Prieto

May 26, 2005

1 Introduction

The *mwlib.a* library is a Fortran 90 Library to estimate the spectra of time series using Dave Thomson's approach (Thomson, 1982), the multitaper spectrum estimation technique. We are using here the adaptive multitaper approach, although you can also get the average multitaper estimate as well as the simple weigthed multitaper estimate.

This subroutine allows you to obtain the spectrum estimation as well as all subproducts, such as a jackknife 95% confidence interval, the y_k's, which contain the phase information, the weights of the different eigenspectra, and also the tapers used (Slepian sequences) and correspondent eigenvalues (energy concentration).

The basic subroutine that is used is `mtspec.f95`, which performs all the calculations. Note that this estimate does not remove line components. The basic way to call this subroutine is

```
call mtspec(npts,dt,x,tbp,kspec,nf,freq,spec)
```

A description of all the variables will follow later. The only part that the user has to decide, is the time-bandwidth product `tbp` and the number of tapers to use `kspec`.

If for example you would also like to have the jackknife error estimate, you could ask for this optional argument when calling the subroutine

```
call mtspec(npts,dt,x,tbp,kspec,nf,freq,spec,err=jack)
```

which will return the appropriate jackknife estimate in the array `jack`, featuring the 5% and the 95% confidence intervals. This feature of optional arguments is available for Fortran 90 and is useful for subroutines which have many arguments that aren't always required.

2 Requirements

The first requirement is to have a Fortran 90 or 95 compiler available in your machine. This library has been tested using the Absoft Compiler in a Mac

OS X system, and using the Sun Compiler in the Unix environment. Since we are taking advantage of features in Fortran 90 such as optional arguments and modules not available in F77, a F90/F95 compiler is needed. Most of the things you can do in F77 can be compiled by F90, so there is no reason not to use F90 instead (of course this is a personal opinion).

A second requirement is that you have to edit the Makefile a little bit to adjust to your specific compiler options. There are small differences between compilers, especially the option to look for modules. You edit the Makefile to be able to search the modules used by this library.

3 Compiling

There is a simple steps in compiling the library. You simply run the Makefile script located in the /mtspec/ folder, to compile all the individual subroutines, as well as compiling the `spectra` module. A library will be created as well as two module files (the object files are removed).

```
mwlib.a
spectra.mod
gutil.mod
```

This is the easy part of compiling. Now you need to be able to use this library in any of your programs. This part is very important for the correct use of this library. The makefile for the Absoft compiler

```
# Objects and Libraries

OBJS = $(path)/mtspec/mwlib.a

# Module locations

MODS = $(path)/mtspec

# Compiler

FC =      f95

# Compiler flags
#   none
FFLAGS =

# Module flag
# Absoft Compiler

MFLAG = -p
MODULE = $(MFLAG)$(MODS)
```

```
# Compile

all : mw_test mt_test jk_test yk_back

%:      %.f95 $(OBJS)
$(FC) $(FFLAGS) $(MODULE)  $< $(OBJS) -o $@


# Clean

clean :
rm mw_test mt_test jk_test yk_back
```

and the Sun compiler

```
# Objects and Libraries

OBJS = $(path)/mtspec/mwlib.a

# Module locations

MODS = $(path)/mtspec

# Compiler

FC =      f95

# Compiler flags
#   none
FFLAGS =

# Module flag
# Sun Compiler

MFLAG = -M
MODULE = $(MFLAG)$(MODS)

# Compile

all : mw_test mt_test jk_test yk_back

%:      %.f95 $(OBJS)
$(FC) $(FFLAGS) $(MODULE) $< $(OBJS) -o $@


# Clean
```

```
clean :
rm mw_test mt_test jk_test yk_back
```

Note how the modules are linked. Modules are linked during the compilation separate from the libraries. This is compiler dependent. The Absoft compiler uses `-p`, meaning path, to look for the modules. The Sun compiler uses `-M` meaning modules and other possibilities such as `-I` or `-i` for the NAG compiler. You have to find out what your compiler options are. The compiler searches the modules in the directory where the program is running, and some predetermined places, so if your module is not there, you need to use this module option. It is a good idea to use this, since it reminds you what exactly you are using.

Finally you will need to `use` the module in your program. The way you use a module in a program is simply to type `use mod_name` before defining any variables. So, as an example, we have

```
program test_mwlib

!
! This is a test program for the multitaper library
! mwlib.a

!*************************************************

   use spectra

   implicit none

   real(8), dimension(npts) :: x

!*************************************************

...

end program test_mwlib
```

Be aware that the use of the module `spectra.mod` is absolutely necessary. This allows the use of optional arguments, making the use of the library much more friendly.

4 Basics

The `mtspec` subroutine is in charge of estimating the adaptive weigthed multitaper spectrum, as in Thomson (1982). This is done by estimating the dpss (discrete prolate spheroidal sequences), multiplying each of the kspec tapers

with the data series, take the fft, and using the adaptive scheme for a better estimation.

As a by product of the spectrum (`spec`), all intermediate steps are estimated, and can be called as optional variables. By products include the phase information in `yk`, the eigenspectra `sk`, the simple weigthed mean spectra `sbar`, the jackknife 95% confidence intervals `err(nf,2)`, the degrees of freedom `se` and the weigths `wt(nf,kspec)` used.

A new feature introduced to the library, is the capability to deal either with single precision or double precision datasets. Still the internal code works in double precision (needed basically by the prolate estimation part), but your Input/Output is going to be single or double precision.

To get ahold of this values, simply call them in the subroutine. Note that the order of calling this variables does matter. But if you want specific variables from the subroutine you can specify them as we will see later.

The subroutine argument list is as follows

```
mtspec (npts,dt,x,tbp,kspec,nf,freq,spec,          &
   yk, wt, err, se, sk, sbar, vn, lambda, theta,   &
   xi, seavg )
```

where all variables are described in the next set of tables

Table 1: The input arguments (double and single precision)

Var	**Double**	**Single**	**Dimension**	**Description**
npts	integer	integer	1x1	number of points in time series
dt	real(8)	real(4)	1x1	the sampling interval
x	real(8)	real(4)	npts x 1	the real time series
tbp	real(8)	real(4)	1x1	the time-bandwidth product
kspec	integer	integer	1x1	the number of tapers to use
nf	integer	integer	1x1	number freq. bins $(npts/2+1)$

Table 2: Standard output arguments

Var	**Double**	**Single**	**Dimension**	**Description**
freq	real(8)	real(4)	nf x 1	the real frequency vector
spec	real(8)	real(4)	nf x 1	the spectrum estimate

The variables from tables 1 and 2 are always required. The order this variables are set during the call of the subroutine is also required. All other variables are optional, meaning that you are not needed to use them, neither you have to define them in the main program.

Table 3: Optional output arguments

Var	Double	Single	Dimension	Description
yk	complex(8)	complex(4)	npts x kspec	the eigencoefficients
wt	real(8)	real(4)	nf x kspec	weights
err	real(8)	real(4)	nf x 2	95% jackknife c.i.
se	real(8)	real(4)	nf x 1	ndf for each freq bin
sk	real(8)	real(4)	nf x kspec	the eigenspectra ($y_k{}^2$)
sbar	real(8)	real(4)	nf x 1	the mean spectrum
vn	real(8)	real(4)	npts x kspec	the Slepian sequences
lambda	real(8)	real(4)	kspec x 1	the eigenvalues of dpss
theta	real(8)	real(4)	kspec x 1	1- eigenvalues of dpss
xi	real(8)	real(4)	1 x 1	the variance efficiency
seavg	real(8)	real(4)	1 x 1	stability (close to one)

Also, be aware that if your input variables are single precision, your output is going to be accordingly assigned, internally by the library. This is all true, except for the output of the prolate eigenvalues and eigenfunctions. The most basic use of the library is

```
call mtspec(npts,dt,x,tbp,kspec,nf,freq,spec)
```

Be aware that the number of frequency points `nf`, has to be $npts/2+1$. This means if the time series is 100 points long, the spectrum has 51 bins, and if the time series is 101 points long, its spectrum is still 51 points long.

The output will then be a frequency vector `freq(nf)` containing all the frequency bins, and the spectrum `spec(nf)` with the power spectrum of the time series.

It might be that you don't want to loose the phase information, or you might want the error estimates. In previous Fortran codes you had to write all these variable names and have them defined at the top of your program, leading to too many unused variables running around.

With the optional arguments the subroutine can only output some of them, without any specific order during the call of the subroutine. You can then get the 95% confidence interval, without the need to call the yk's as well. One would type

```
call mtspec(npts,dt,x,tbp,kspec,nf,freq,spec,err=err)
```

where a third output is now available, `err(nf,2)`. In your main program you have to define this variable by

```
real(8), dimension(nf) :: spec, freq
real(8), dimension(nf,2) :: err
...
```

One can get as many of the optional variables as desired. An example can be

```
call mtspec(npts,dt,x,tbp,kspec,nf,freq,spec,   &
   err=err,yk=yk,wt=wt,xi=xi)
```

and we get the eigencoefficients, the jackknife estimate, the weights of the different eigenspectra and the variance efficiency.

5 Example

Here is a basic example of a single program using the multitaper library. Some output figures are presented at the end of this documentation file. The plotting is performed using the *gplot* subroutine, which allows simply plots to be drawn from inside a fortran program.

```
program mw_test

! The program calls the eigenft subroutine to get the
! eigenspectra, as in Thomson (1982).

!********************
   use spectra
   use gplotxy

   implicit none

!  time series

   integer :: npts
   real(8), dimension(:), allocatable :: x, t

!  spectra and frequency

   integer :: nf

   real(8), dimension(:), allocatable :: spec, freq
   real(8), dimension(:), allocatable :: sbar
   real(8), dimension(:,:), allocatable :: err

!  Spectral parameters

   integer :: kspec

   real(8) :: tbp, dt

!  Others
```

```
   integer :: ios, i, k

   character (len = 100) :: filename

!********************

   write(6,'(a)') 'The number of tapers to get '
   read(5,*) kspec
   write(6,'(a)') 'The time bandwidth product '
   read(5,*) tbp
   write(6,'(a)') 'The time series length '
   read(5,*) npts

   dt = 0.01d0

   if (npts <= 1) then
      write(6,'(a)') '>>> Too few terms in the series'
      stop
   endif

   nf = 1 + npts/2

!  Allocate memory

   allocate(x(npts))
   allocate(t(npts))
   allocate(freq(nf))
   allocate(spec(nf))
   allocate(err(nf,2))
   allocate(sbar(nf))

!  Define time axis

   do i = 1,npts
      t(i) = dble(i)*dt
   enddo

   filename = 'seismic.dat'

   open (11, file=filename, status='old')
   do k = 1, npts
      read(11,*, iostat=ios) x(k)
      if (ios<0) then
         ! End of file encountered
         exit
```

```
      endif
   enddo
   close(11)

!  Get the spectrum estimate

   call mtspec(npts,dt,x,tbp,kspec,nf,freq,spec,   &
      err=err,sbar=sbar)

!  Plot

   call gplot(t,x,output='ts.ps')
   call gplot(freq,spec,logxy='loglog',output='aspec.ps')
   call gplot(freq,sbar,logxy='loglog',output='mspec.ps')

end program mw_test
```

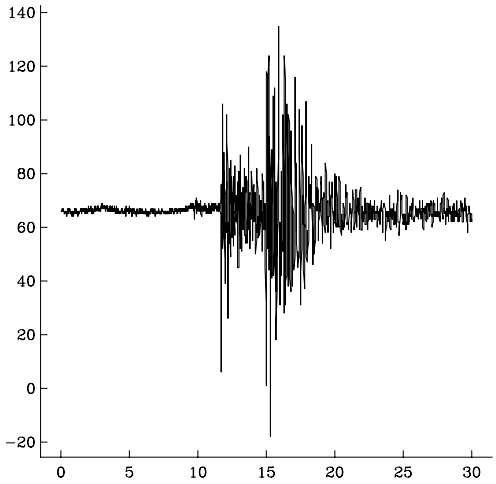

Figure 1: time series

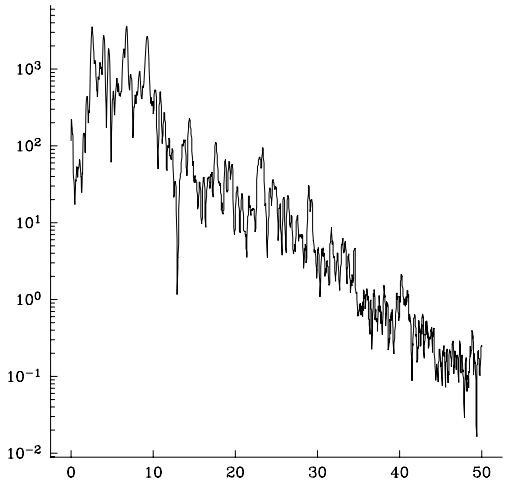

Figure 2: Adaptive Spectral Estimate

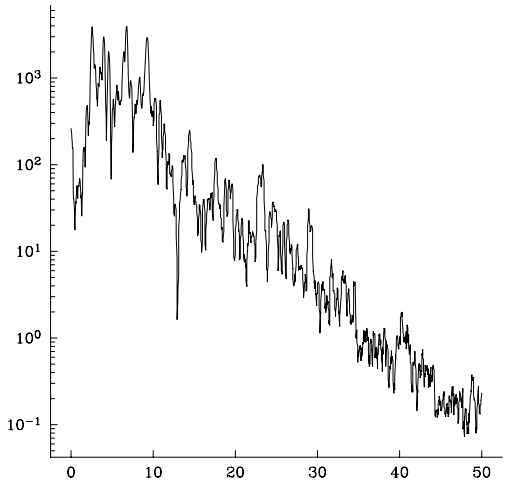

Figure 3: Mean Spectral Estimate

References

Thomson, D. J. (1982). Spectrum estimation and harmonic analysis. In *Proc. of the IEEE*, volume 70, pages 1055–1096.