# Exercises for spectral methods seismic modelling

## Preliminaries

The goal is to code a simple 1-D and a 2-D acoustic modelling program based on the Fourier method. Time integration will be done in a time stepping manner ($2^{nd}$ order scheme).

For an efficient exercises lesson the participants are asked to stick to the below given names of variables. This helps us to identify bugs in the programs. Don't use lengthy names like "*index_in_x_direction*" or the like. Please don't put any effort into program optimisation or beautiful programming style. Therefore, hard-code parameters instead of writing read-in routines.

Please do not code your programs in an extremely modular way, i.e., don't hide simple operations in subroutines.

If possible use FORTRAN 90. C and Python are acceptable. Other programming languages are not supported. See at the end some comments about Python tools and programming.

## Agenda

- Code a subroutine for calculating the spatial derivate
- Test the subroutine
- Code a 1-D acoustic modelling program
- Code a 2-D acoustic modelling program (if time allows)

## Program layout

```
declaration and allocation of simple variables and arrays
initialization of simple variables like DX, DT etc.
initialization of arrays like C**2, P, PP, CKX

begin of loop over time steps (L=1...NT)
   call of differentiation routine
   add sample of source time function at source location
   time integration
   print max. of pressure field
   write entire pressure field to file
end of loop over time steps
```

Subroutines for the FFT, the source time function, wave number array and a main program for testing your differencing routine will be supplied (no need to program this).

# Variables

Our methods are based on numerical grids of one or two dimensions. Therefore we will store our field variables in one- or two-dimensional arrays.
FORTRAN users, please let your arrays start at index 1.
C users, please let your arrays start at index 0.
These are the languages' defaults.
Real variables need to be of "single" precision.

I, K : loop indices in x- or z-direction
L: loop index for time loop
ISX, ISZ: index of source position
NX, NZ: number of grid points in x- or z-direction
NT: number of time steps

Simple REAL variables:
DT : time increment ($\Delta t$)
DX, DZ : spatial increment ($\Delta x, \Delta z$)
EFHZ: cutoff frequency of the (Ricker-) wavelet
T: time
TMAX: maximum propagation time
XMAX, ZMAX: lengths in x- or z-direction

Array variables (REAL):
P : pressure at time step n ($p^n$)
PP : pressure at previous time step ($p^{n-1}$)
C2 : square of velocity
A: work array

Array variables (COMPLEX):
CKX, CKZ : wave number arrays

One or more dimensional arrays in C should be allocated in the following way:

```
ckx = (complex*) alloc1(nx,sizeof(complex));
p = (float**) alloc2(nx,nz,sizeof(float));
```

In the 2-D case nx refers to the 'fast' index (second index). These allocation functions are found in the file alloc.c.

# Units

Please use the following units:
length in meters, time in seconds, speed in meters/second, frequency in Hertz

# Typical causes for errors

- uninitialised variables (may be different from zero)
- wrong array dimensions or not allocated array space
- wrong variable type (e.g. INTEGER instead of REAL)
- use of, e.g. scalar instead of array variables
- mixing ordinary syntax with array syntax unintentionally
- incompatible units

Therefore, declare any variable. Use "IMPLICIT NONE" in FORTRAN.

# Fourier derivative

Since we need to perform spatial $2^{nd}$ derivative calculations in the modelling algorithm let us first code a subroutine for a 1-D derivative operator with the following name and argument list:

```
SUBROUTINE DIFX(VIN,VOUT,CKX,NX)
```

VIN - input vector (input)
VOUT - output vector (output)
CKX - complex 1-D array of wave numbers, i.e. $ik_x$ or $-k_x^2$ (input)
NX - number of grid points (input)

The procedure for a first or second derivative is as follows:

$$f(x) \stackrel{DFT}{\longrightarrow} F(k) \longrightarrow i\,k_x\,F(k) \stackrel{DFT^{-1}}{\longrightarrow} f'(x)$$

or

$$f(x) \stackrel{DFT}{\longrightarrow} F(k) \longrightarrow -k^2\,F(k) \stackrel{DFT^{-1}}{\longrightarrow} f''(x)$$

Inside this subroutine we have to

- perform a Fourier transform
- multiply any component of the spectrum with the corresponding element of the wave number array
- perform an inverse Fourier transform
- depending on the particular FFT normalisation is needed

(We need to copy the real array VIN to a complex local complex array before applying the FFT.)

The parameter list of the given FORTRAN Fourier transform subroutine looks like this:

```
SUBROUTINE FFORK(LX,CX,SIGNI)
```

LX - length of transform (input)
CX - complex 1-D array (input/output)
SIGNI - real (!) variable: -1.0 - forward, +1.0 - inverse transform (input)

Note: This particular FFT requires that LX must be a power of 2!

The parameter list of the given C Fourier transform subroutine (taken from CWP) looks like this:

```
void pfacc (int isign, int n, complex z[]);
```

isign - integer variable: -1.0 - forward, +1.0 - inverse transform (input)
n - length of transform (input)
z - complex 1-D array (input/output)

Note: This FFT is found in file pfafft.c. Also there you find valid numbers for n. File cwp.h must be included. Complex variables are defined there in the following way:

```
typedef struct _complexStruct {
        float r,i;
} complex;
```

Since C has no complex arithmetic, complex products need to be done 'by hand'. Our functions are real. We therefore need to set the imaginary parts equal to zero.

We also need to initialise the wave number array before the first call to DIFX. For this a subroutine is provided. It is declared like this:

```
SUBROUTINE WAVNUM(CK,N,D,IORD)
```

CK - complex wave number array (actual array CKX or CKZ) (output)
N - length of transform (actual parameters NX or NZ) (input)
D - real spatial increment (actual parameters DX or DZ) (input)
IORD - order of derivative (1 - first, 2 - second derivative, ...) (input)

Inside the subroutine we initialise the array elements according to:

$$k_n = \begin{cases} \frac{2\pi}{N\Delta x} n & \text{if } n = 0, \ldots, [N/2], \\ \\ \frac{-2\pi}{N\Delta x}(N-n) & \text{if } n = [N/2]+1, \ldots, N-1 \end{cases}$$

This applies to arrays where the first index is zero! In FORTRAN set $k_{n+1}$ on the left hand side.
According to IORD, $(i\ k_n)^{IORD}$ needs to be placed into the array CK.

# Testing the derivative routine

Write a small test program to check that your DIFX really works. Since the Fourier method is inherently periodic take the derivative of periodic functions (e.g., 1, 2, 3 ... periods per length of transform). Use sine- or cosine-functions and compare with known values of the derivative. For testing use FFT of 32 in Fortran (30 in C).
Do tests with a first derivative and then test the second derivative in the same way.

If things don't work out properly, simplify the routine and instead of performing a derivative by multiplicating with wave numbers, simply perform forward and reverse Fourier transforms. Check that the output is (apart from round off errors) the same as your input function. Having done this it might be easier to find a possible bug.

# 1-D acoustic modelling

Having a working second derivative subroutine we can start coding the modelling program. Let its name be ac1d.f90 or ac1d.c. Basically we have to program the 1-D wave equation:

$$\frac{\partial^2 p}{\partial t^2} = c^2 \frac{\partial^2 p}{\partial x^2} + S$$

Finite differencing the time derivative:

$$\frac{p^{n+1} - 2\,p^n + p^{n-1}}{(\Delta t)^2} = c^2\,\frac{\partial^2 p^n}{\partial x^2} + S^n$$

Solution for $p^{n+1}$:

$$p^{n+1} = 2\,p^n - p^{n-1} + (\Delta t)^2\,\left[c^2\,\frac{\partial^2 p^n}{\partial x^2} + S^n\right]$$

$S^n$ is supplied by the function FWAVE at time $n \cdot \Delta t$. It is only added at the specified source location defined by the given index ISX on the 1-D grid.

This is the formula which we need to program. The spatial derivative is output of our DIFX subroutine. The last formula has to be repeated many times until $t_{max}$ is reached. It therefore should be placed inside a loop over time steps. Before entering the time loop variables need to be initialised.
At first chose $nx = 128$ in FORTRAN ($nx = 130$ in C), $\Delta x = 10m$, $c = 2000m/s$ and the wavelet's maximum frequency $efhz = 100Hz$. Set the source location $isx = 32$, $t_{max} = 0.5s$ .

The source time function is generated by the function 'fwave', e.g.:

```
REAL FUNCTION FWAVE(TIME,EFHZ)
```

The maximum time step size $\Delta t$ can be calculated form the 1-D stability criterion:

$$\frac{\pi}{2}\frac{c \cdot \Delta t}{\Delta x} \leq 1$$

For less dispersion divide this by five.

To be able to display results write the entire pressure field into a file at any time step. Do it like this in FORTRAN:

```
WRITE(10) P
```

Do it like this in C:

```
fwrite(p,sizeof(float),nx,fp);
```

In C the file needs to be opened before first time writing and closed after the last write call:

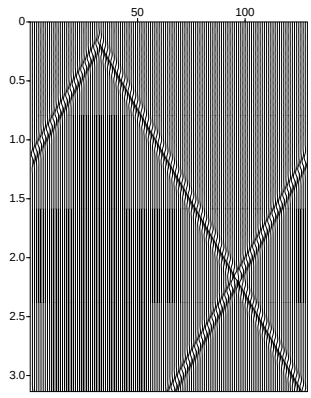
```
fp = fopen("section","w");
...
fclose(fp);
```

Seismograms from FORTRAN output can be displayed using the SU package:

```
suaddhead < fort.10 ns=128 ftn=1 | suflip flip=0 | suxwigb d1=1
```

Seismograms from C output can be displayed using the SU package:

```
suaddhead < section ns=130 | suflip flip=0 | suxwigb d1=1
```




Having seen seismograms from a homogeneous model we can make the model inhomogeneous: Set $c = 4000m/s$ from grid point #65 on. It is helpful to print the maximum pressure inside the array p at any time step in order to watch whether the algorithm runs stable. Exponential growth of the values is an indicator that the algorithm is unstable. Run your program again and display seismograms. What do you observe in the seismogram section? Try to run the program with $\Delta t$ larger than calculated from the stability criterion. What happens?

# 2-D acoustic modelling

The 2-D modelling program (ac2d.f90 or ac2d.c) is based on the 2-D wave equation:

$$\frac{\partial^2 p}{\partial t^2} = c^2 \left[\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial z^2}\right] + S$$

Code two new subroutines (difx2d and difz2d) with respect to the x- and the z-directions. Input and output are 2-D arrays. One call to the subroutine will deliver all x- or z-derivatives of the 2-D array, respectively, e.g.:

```
call difx2d(p,aux1,ckx,nx,nz)
```

Code a 2-D program in analogy to the already available 1-D program. At first chose $nx = nz = 128$ (130), $\Delta x = \Delta z = 10m$, $c = 2000m/s$ and the wavelet's maximum frequency $efhz = 100Hz$. Set the source location $isx = 32$, $isz = 56$, $t_{max} = 0.25s$ .

The 2-D stability criterion is:

$$\frac{\pi}{\sqrt{2}} \frac{c\Delta t}{\Delta} \leq 1$$

Write the entire pressure field at every $10^{th}$ time step column-wise to a file:

```
do i=1, nx
  write(10) p(i,:)
end do
```

Display a snapshot movie:

```
suaddhead < fort.10 ns=128 ftn=1 | suxmovie n2=128 loop=1 clip=1e-7 d1=1
```

Change the subsurface such that the acoustic velocity from the 65th grid point downward is $c = 4000m/s$. What do you observe in the snapshots movie?

C programmers display movies like this:

```
suaddhead < snapshot ns=130 | suxmovie n2=130 loop=1 clip=1e-7 d1=1
```

# Helper scripts

There makefiles for compilation, e.g. Makefile_1d, which is started like: make -f Makefile_1d. You may have to change the compilation command according to the name of your compiler.

There are two scripts to display sections and snapshot movies using SU: sect1d and mov2d. SU display programs can be stopped by typing 'q' if the focus is on the display area. Movies can be stopped/restarted by clicking the right mouse button.

# Python tools and programming

If you have installed Python/Numeric/numarray/Gnuplot/wxPythom it is possible to visualize seismograms also with SismoVi.py program. It is an interactive GUI application based on python and wxPython. In the SismoVi folder there is SVData.f90, a fortran example for writing data in the format that SismoVi can read and display. There is also a short on-line help.

For using SismoVi, click on the application name or execute the command:

```
python  SismoVi.py
```

Then input the file to visualize and modify visualization parameters according to your need.

When programming in Python, it is possible to call SismoVi directly from the code or to visualize the data dynamically (real-time movie). The needed instructions are already coded in the python exercises.

Python is an object oriented scripting language (interactive) that support the procedural style (like C or Fortran). Programming style is very similar to Fortran. For efficiency it is convenient to use the ARRAY SYNTAX approach (like in fortran) that can be used after importing Numeric or numarray math. library. A quick guide is attached.

Main differences with fortran:

- Variables do not need to be declared. They are allocated/deallocated automatically as needed.
- Array start at 0 and end at $N-1$ with $N$ terms.
- Indentation is the base of the programming style and it is compulsory.
- Block of indented instructions are treated as a single instruction