

# Programación Numérica para Geofísica

## PNG 2020-1

Andrés Sepúlveda

Departamento de Geofísica  
Universidad de Concepción

08 Junio 2020

- **Lectura de Archivos**

- ▶ load/save
- ▶ fprintf
- ▶ textread
- ▶ xlsread
- ▶ Estructuras
- ▶ NetCDF/HDF5

- **Lectura de Archivos**

- Paso vital para el análisis de información.
- Distintos tipos de formatos:
- Binarios: dependen de la máquina y lenguaje donde se graba.
- Propietarios: .mat, .fig, .xls
- Convenciones: NetCDF, HDF, ASCII, csv.

# ASCII

- El conocido formato de texto.
- Usualmente no presenta problemas.
- Sin embargo, MAC, Windows, y Linux manejan de forma distinta el cambio de línea.
- Solución Linux: **dos2unix** y **unix2dos**.

# LOAD

- **load** - Archivo de texto (ASCII)

```
> a = load('plano.txt');
```

- **load**

```
> load nombre_muy_complicado_y_largo.txt
```

guarda todo en la variable *nombre\_muy\_complicado\_y\_largo*

- **load** - Archivo tipo .mat (Matlab/Octave)

```
> load Ajuste.mat
```

Carga en memoria, en variables separadas, la información guardada al crear un *.mat*

# SAVE/LOAD

- **save** - graba todo/variables que están en memoria, a un archivo. Se puede especificar el formato (ASCII, Binario). Usa la extensión **.mat**

```
save Ajuste.mat  r2 std meanval median val
save -ascii Ajuste.txt  r2 std meanval median val
```

ASCII preferible para matrices 2D

- **load** - Archivo .mat con muchas variables

```
load  Ajuste r2 std meanval;
stats = load('Ajuste','r2','std','meanval');
```

# FPRINTF

- **fprintf** se usa para grabar archivos en ASCII con un formato regular, que se repite en cada linea.
- Sintaxis:

```
filename=['conexiones_particulas.txt'];  
fid = fopen(filename,'w+');  
for i=1:10  
    fprintf(fid,'%i %i %.1f %i %.1f', matrix(i,:));  
    fprintf(fid,'\n');  
end  
fclose(fid);
```

Alternativa: **dlmwrite**.

# TEXTREAD

- **textread**

```
b = textread('plano.txt');
```

- textread es útil cuando los archivos tiene un formato uniforme

```
[A,B,C,...] = textread(filename,format,N)
```

- Ejemplo

```
mydata.dat -> Sally      Level1 12.34 45 Yes  
[names, types, x, y, answer] = textread('mydata.dat', ...  
    '%s %s %f %d %s', 1)
```

- Ejemplo - Omitiendo columnas

```
mydata.dat -> Sally      Level1 12.34 45 Yes  
[names, types, y, answer] = textread('mydata.dat', ...  
    '%s %s %*f %d %s', 1)
```



## Archivos Excel

```
A = xlsread ('test4.xls', '2nd_sheet', 'C3:AB40');
```

- Lee el archivo test4.xlsread
- Lee la segunda hoja de ese archivo.
- Lee el rango de celdas C3:AB40
- Complementado por **xlswrite**.
- Primo de **csvread** (Comma Separated Value)

- Las estructuras son formas de almacenar los datos que después serán guardados con un **save**
- Una buena organización de los datos facilita su uso.
- Permite agrupar datos de distinto tipo (números, letras).
- Un ejemplo de su uso son las **Bases de Datos**.

- Ejemplo

`'Nicanor Parra' 100 M`

`'Josefa Tapia' 11 F`

- Acá definimos varios **campos**

- ① Nombre Completo, Edad, Sexo.

- ② Podría dividirse en Nombre, Apellido.

- Ejemplo

`persona.nombre='Nicanor'`

- `persona.apellido='Parra'`

- `persona.edad=100`

- Para aumentar la cantidad de registros podemos hacer:

```
persona(2).nombre='Josefa'  
persona(2).apellido='Tapia'  
persona(2).edad=11
```

(esto se llama **asignación directa**).

- ¿ Qué obtendremos de escribir lo siguiente?

```
persona(1)
```

- ```
persona(1)
```

```
ans =
```

scalar structure containing the fields:

```
nombre = Nicanor  
apellido = Parra  
edad = 100
```

- ¿Qué sucede al agregar una nueva persona?

```
persona(3).nombre = 'Pablo';  
persona(3)
```

- Se expande la estructura y los otros campos quedan vacíos

```
persona(3)  
ans =
```

scalar structure containing the fields:

```
nombre = Pablo  
apellido = [](0x0)  
edad = [](0x0)
```

- Todas las estructuras tienen entonces el mismo número de campos.

- La función **fieldnames** nos entrega los nombres de los campos:

```
fieldnames(persona)
ans =
{
    [1,1] = nombre
    [2,1] = apellido
    [3,1] = edad
}
```

- Además de la asignación directa existe el comando **struct**.

```
persona2=struct('nombre','Nicanor','apellido', ...
    'Parra','edad',100)
persona2 =
```

scalar structure containing the fields:

```
nombre = Nicanor
apellido = Parra
edad = 100
```

- (note el uso de ... )

- Podemos agregar varios elementos a la vez

```
persona2=struct('nombre',{'Nicanor','Josefa'},'apellido', ...  
               {'Parra','Tapia'},'edad',{100,11});
```

- Ojo con esto...

¿Qué diferencia hace?

```
persona2=struct('nombre',{'Nicanor','Josefa'},'apellido', ...  
               'Parra','edad',100);
```

- Se usa entonces el operador `.` para acceder a la información.

```
persona2(2).nombre  
ans = Josefa
```

- O con la función **getfield**

```
a=getfield(persona2,{2},'nombre')  
a = Josefa
```

- O también

```
A = [persona2.edad]  
A =
```

```
100 11
```

- Si agregamos un campo más a un elemento

```
persona(1).rut='5.110.040-9'
```

¿Qué pasa?



- En cambio, para eliminar un campo, tenemos **rmfield**  
`persona=rmfield(persona,'rut')`
- Finalmente, para copiar una estructura usamos  
`personajes=persona;`

# Estructuras - Funciones

|                    |                                    |
|--------------------|------------------------------------|
| <b>fieldnames</b>  | Entrega nombres de campos          |
| <b>getfield</b>    | Entrega el contenido de los campos |
| <b>isfield</b>     | Existencia de campo en estructura  |
| <b>isstruct</b>    | ¿Es una estructura?                |
| <b>rmfield</b>     | Elimina un campo                   |
| <b>setfield</b>    | Define el contenido de un campo    |
| <b>struct</b>      | Crea una estructura                |
| <b>struct2cell</b> | Convierte estructura a celda       |

```

%
% Calcula la razon entre las concentraciones
%
test(1).plomo=.007      ; test(2).plomo=0.031;
                        test(3).plomo=.019;
test(1).mercurio=.0021; test(2).mercurio=0.0009;
                        test(3).mercurio=.0013;
test(1).cromo=.0025     ; test(2).cromo=0.017;
                        test(3).cromo=0.10;

function [r1, r2]= concentracion(muestra);
%r1 contiene el cuociente entre mercurio y plomo.
%r2 contiene el cuociente entre plomo y cromo
r1=[muestra.mercurio] ./ [muestra.plomo];
r2=[muestra.plomo] ./ [muestra.cromo];
return
end

```

```
% Grafico de concentraciones de plomo, mercurio y cromo  
% sobre el mismo grafico usando diferentes colores  
plomo=[test.plomo];  
mercurio=[test.mercurio];  
cromo=[test.cromo];  
plot(plomo, 'r'); hold on  
plot(mercurio, 'b')  
plot(cromo, 'y'); hold off
```

# NetCDF

- Un tipo de archivo usado en Geofísica
- Puede ser grabado en una maquina (Linux) y usado en otra (Windows)
- Almacena las variables, y describe el contenido

www.unidata.ucar.edu/software/netcdf/



**unidata**

providing innovative data services and tools to transform the conduct of geoscience

[Login](#) | [Register](#)



[Data](#) [Software](#) [Downloads](#) [Support](#) [Community](#) [Projects](#) [News](#) [Events](#) [About Us](#)

Home / NetCDF

### NetCDF

[Version History](#)

[FAQs](#)

[Documentation](#)

[Download](#)

[Support](#)

[For Developers](#)

[NetCDF Java](#)

[Compatible Software](#)

[NetCDF CDash Tests](#)

[Related Projects](#)

## Network Common Data Form (NetCDF)



NetCDF is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data.

[See the netCDF package overview](#)

### NetCDF News & Announcements

#### NetCDF-Fortran 4.4.0

July 14, 2014

#### NetCDF-Java library and TDS version 4.3.22

May 27, 2014

#### NetCDF 4.3.2

April 23, 2014

### NetCDF Fact Sheet

A [netCDF fact sheet](#) provides a brief overview of the netCDF package and supported languages and platforms.

[View the netCDF fact sheet](#)

### NetCDF Musings

- En Octave

```
octave-octcdf  <- Sintaxis simple  
octave-netcdf  <- Compatible con Matlab  
ncArray        <- Encompasa ambos
```

- En Matlab

Diferentes opciones

- 1 MexCDF: (sintaxis similar a OctCDF)
- 2 Otra sintaxis desde versión 2012.

<http://www.mathworks.com/help/matlab/ref/netcdf.html>

- 3 En Java
- 4 Enlaces Relevantes

- 1 Google: Unidata, NetCDF
- 2 Google: OctCDF, Barth
- 3 Google: NetCDF, Matlab

- Información a guardar

```
> vec_lat = -90:1:90;  
> vec_lon = -179:1:180;  
> [y,x] = meshgrid(pi/180 * vec_lat,pi/180 * vec_lon);  
> mat_temp = cos(2*x) .* cos(y);
```

- Crear archivo (puntero)

```
> nc = netcdf('example.nc','c');
```

'r', 'w'

- Crear **dimensiones**

```
> nc('dim_lon') = 360;  
> nc('dim_lat') = 181;
```

- Definir y llenar **variable**

```
> nc{'longitude'} = ncdouble('dim_lon');  
> nc{'longitude'}(:) = vec_lon;  
> nc{'longitude'}.units = 'degrees_east';
```



- La otra variable (latitud)

```
> nc{'latitude'} = ncdouble('dim_lat');  
> nc{'latitude'}(:) = vec_lat;  
> nc{'latitude'}.units = 'degrees_north';
```

- Una variable con mas información

```
> nc{'temp'} = ncdouble('dim_lat','dim_lon');  
> nc{'temp'}(:) = mat_temp;  
> nc{'temp'}.long_name = 'Temperature';  
> nc{'temp'}.units = 'degree Celsius';  
> nc{'temp'}.valid_range = [-10 40];
```

- Un **atributo global**

```
> nc.history = 'Archivo creado por J.P.';  
> nc.title = 'Ejemplo 1';
```

- Y **cerramos** el archivo

```
> close(nc)
```

- **ncdump** - Para ver el **encabezado** de los archivos

(Linux)

```
$ ncdump -h example.nc | less
```

- Muestra el encabezado y el **contenido** de la variable

```
$ ncdump -v longitude example.nc | less
```

- Visualización rápida del **contenido**

(Mapas)

```
$ ncview example.nc
```

- Para leer el archivo en Octave/Matlab

```
> nc = netcdf('example.nc','r');  
> n_lon = nc('longitude');  
> temp = nc{'temp'}(:);  
> temp_units = nc{'temp'}.units;  
> temp_valid_range = nc{'temp'}.valid_range;  
> global_history = nc.history;
```

- Dimensión tipo **record**

(concatenar archivos)

```
> nc('time') = 0;  
> nc{'time'} = ncdouble('time');  
> nc{'time'}(1:length(time_values)) = time_values;
```

- Factores de escala y sesgo (**offset**)

```
dato = scale_factor * dato + add_offset
```

Atributos

- Valor de relleno (**\_FillValue**)

```
> nc{'temp'}._FillValue = -99999;
```

Atributos

- Valor omitido (**missing\_value**)

```
> nc{'temp'}.missing_value = -99999;
```

Atributos

- OpenDAP

```
> nc = netcdf('http://asterix.rsmas.miami.edu/
             thredds/dodsC/atl-ops-forecast/temp','r');
> temp = nc{'temp'}(end,1,661:996,77:588);
> temp(temp == nc{'temp'}.missing_value) = NaN;
> close(nc);
```

- HDF es otro formato científico muy usado.
- Muy popular en sus inicios entre los astrónomos.
- Posee características similares a NetCDF.
- Su gran ventaja es que permite una compresión de los datos.
- Pero puede ser complejo de instalar.
- De hecho, el formato NetCDF4 es, en realidad, HDF, que se lee con la sintaxis de los comandos de NetCDF3.

- El tema de la compresión de los datos puede ser reforzado a través de la precisión de las cifras.
- Precisión simple:

S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFF  
 0 1            8 9                            31

S: Signo + / -

E: Exponente

F: Bits (1/0)

- Precisión doble:

S EEEEEEEEEEE FFFFFFFF...FFFFFFFFFFFFFFF  
 0 1            11 12                            63

- GPU usan precisión simple.
- Algunos modelos requieren precisión simple, otros doble.
- Aun cuando el cálculo sea en precisión doble, el resultado puede no requerir de precisión doble para su interpretación, e.g. la velocidad de las corrientes marinas.
- $0.2487625 \text{ m/s} \approx 0.2500000 \text{ cm/s}$ , que se puede guardar como  $+0.25 + 5$  ceros