Data Analysis in Geophysics

Good Practices for Algorithm Writing

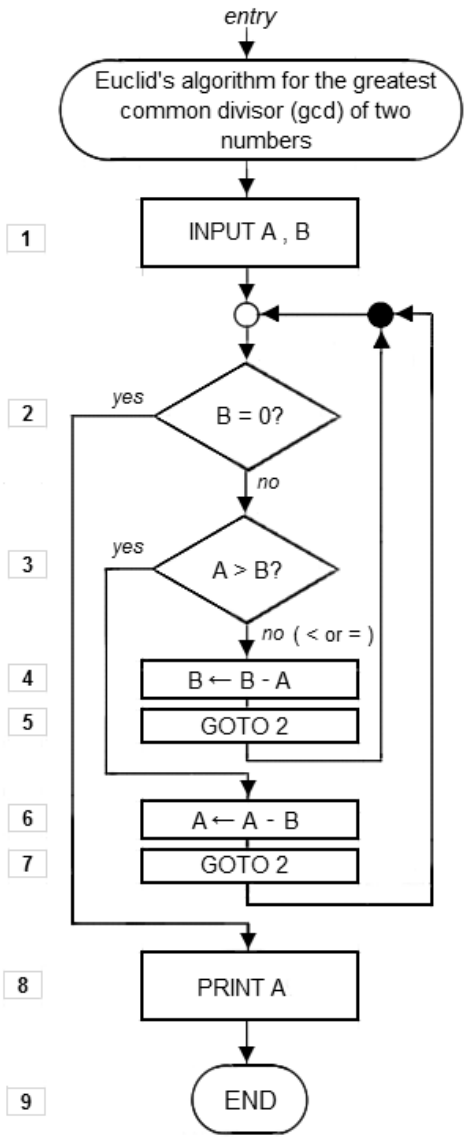
Fall 2013

Demián Gómez

What is an Algorithm?

- **From Wikipedia:** A step by step procedure for calculations, data processing and automated reasoning.
- Algorithms can be applied for any specific task. Like Sheldon's friendship algorithm:

  http://www.youtube.com/watch?v=k0xgjUhEG3U



How do you design an algorithm?

- It is something difficult to teach. There are many designing techniques, but each person has its own methods. It's like riding your bike: you learn by doing it!
- One very important thing is how you implement your algorithm on a computer program.
- A good implementation of a computer algorithm is one that is clear and general enough to be reused if needed.

What do we need to write a computer algorithm?

- Variables.
- Constants.
- Decision blocks.
- Loops.
- Functions or subroutines.

What additional tools do we have to write a good algorithm?

- Indirect addressing.
- Comments and code blocks.
- (among others)

Variables and Constants

- Try to use as many parameters as possible.

Good Code

```
% read the wav file
corner1 = 80;
corner2 = 980;
[y fs] = wavread('wwv2.wav');
% construct a butter filter
[z100, p100] = butter(2,[corner1/(fs/2) (corner1 + 25)/(fs/2)]);
[z1000, p1000] = butter(2,[corner2/(fs/2) (corner2 + 25)/(fs/2)]);
%filter the data% get 100 Hz subcarrier
d = filtfilt(z100, p100,y);
% get 1000 Hz sync pulse
sync=filtfilt(z1000, p1000,y);
% construct the time
vectort=linspace(0,length(d)/fs,length(d))';
subplot(3,1,1); plot(t,d, t, sync);
```

Variables and Constants Cont.

- Bad code

```
% read the wav file
[y] = wavread('wwv2.wav');
% construct a butter filter
[z100, p100] = butter(2,[80/(20050/2) 105/(20050/2)]);
[z1000, p1000] = butter(2,[980/(20050/2) 1005/(20050/2)]);
%filter the data% get 100 Hz subcarrier
d = filtfilt(z100, p100,y);
% get 1000 Hz sync pulse
sync=filtfilt(z1000, p1000,y);
% construct the time
vectort=linspace(0,length(d)/20050,length(d))';
subplot(3,1,1); plot(t,d, t, sync);
```

Constants

- If you have values that don't change throughout your program, it is better to use constants to make sure you are not accidentally changing their values.
- However, constants in Matlab are not easy to define.

```
classdef MyConstants
        properties (Constant = true)
                SECONDS_PER_HOUR = 60*60;
                DISTANCE_TO_MOON_KM = 384403;
        end
end
```

Constants Cont.

- A little bit better in other languages:

C, C++:

```
#define pi 3.1415
```

Basic:

```
Const pi = 3.1415
```

Fortran:

```
Parameter (pi=3.1415)
```

Decision blocks

- To write a decision block, figure out what you are trying to decide and write the minimum amount of code inside the IF block.
- i.e., don’t repeat code in the IF ELSE block.

IF block Example: Good Code

```
if (decision == value)
   var = 1;
else
   var = 0;
end

Result1 = operation(var);
Result2 = operation(Result1, var);
```

IF block Example: Bad Code

```
if (decision == value)
   var = 1;
   Result1 = operation(var);
   Result2 = operation(Result1, var);
else
   var = 0;
   Result1 = operation(var);
   Result2 = operation(Result1, var);
end
```

IF vs Switch Blocks

- If there are more than two branches in your IF statement, you might consider using a switch statement rather than IF ELSEIF, ELSE.
- Switch blocks are a little more clear than if statements (to read), although in Matlab, switch statements are slower than IFs. In many languages this is the other way around.

If vs Switch Example 1

```
switch index
    case 1
         string=[string '1'];
         data_bit = [one; zeros(length(pulse) - length(one), 1)];
    case 2
         string=[string '0'];
         data_bit = [zero; zeros(length(pulse) - length(zero), 1)];
    case 3
         string=[string 'M'];
         data_bit = [mark; zeros(length(pulse) - length(mark), 1)];
    case 4
         string=[string '-'];
         data_bit = [none; zeros(length(pulse) - length(none), 1)];
end
```

If vs Switch Example 2

```
if (index == 1)
   string=[string '1'];
   data_bit = [one; zeros(length(pulse) - length(one), 1)];
elseif (index == 2)
   string=[string '0'];
   data_bit = [zero; zeros(length(pulse) - length(zero), 1)];
elseif (index == 3)
   string=[string 'M'];
   data_bit = [mark; zeros(length(pulse) - length(mark), 1)];
else
   string=[string '-'];
   data_bit = [none; zeros(length(pulse) - length(none), 1)];
end
```

Switch Tricks

```
switch true
    case index == 1
         string=[string '1'];
         data_bit = [one; zeros(length(pulse) - length(one), 1)];
    case index == 2
         string=[string '0'];
         data_bit = [zero; zeros(length(pulse) - length(zero), 1)];
    case index == 3
         string=[string 'M'];
         data_bit = [mark; zeros(length(pulse) - length(mark), 1)];
    case index ~= 1 && index ~= 2 && index ~= 3
         string=[string '-'];
         data_bit = [none; zeros(length(pulse) - length(none), 1)];
end
```

While Loops and For Loops

- Although these two statements have the same functionality, FOR loops are meant to count a predetermined number of elements.
- WHILE loops are meant to count an unknown number of elements or are supposed to be used to do something until a condition is met.

For Loop Example

- This is an example of a predefined number of elements in a FOR loop.

```
for i=2:100
   MyVect(i,1) = MyVect(i-1,1)*2;
end
```

Stupid way of using a For Loop

- This is an example of a FOR loop incorrectly used (although it works).

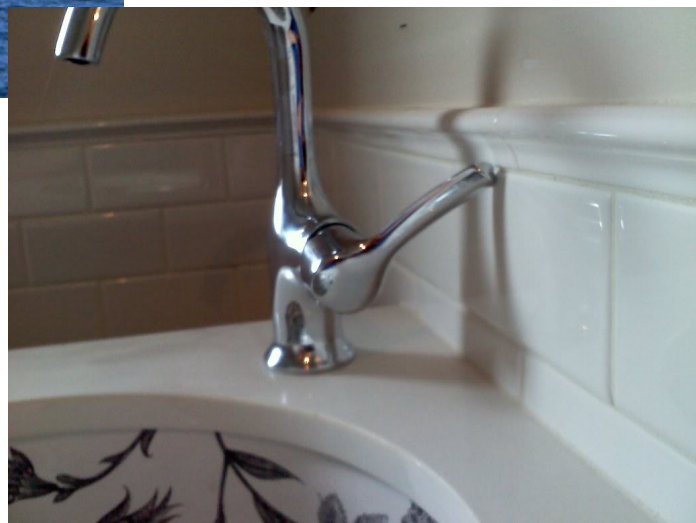
```
for i=1:100000000
    if (condition == value)
          % do whatever in here
          break;
    end
End
```

- This code is correct.

```
While condition ~= value
    % do whatever in here
end
```

Functions and Subroutines

- When writing a program, you should think about the future so that stuff like this doesn't happen:



Functions and Subroutines

- You also want to make sure that the code is CLEAR (easy to understand).
- To do so, write your code using functions and subroutines that will do common operations. This will allow you to reutilize code in the future and will help you to avoid unexpected problems.

Subroutine Example

```
clear all
clc
a = 1; b = 2;
Result = suma(a, b);
```

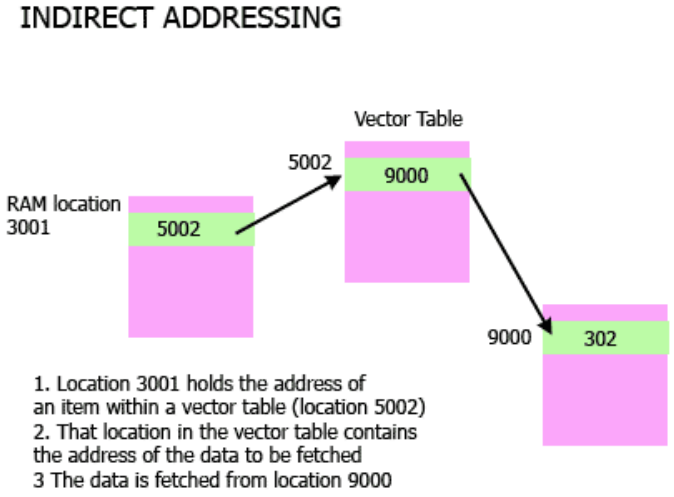
On a different file you might have:

```
function [x] = suma(val1, val2)
   x = val1 + val2;
end
```

The next time you write a program and you need the function “suma”, you just need to copy this file from one directory to another.

Final word on indirect addressing

- Indirect addressing can be very helpful to write shorter and better code.
- This technique allows you to access data from a vector table that holds the address of the actual values that you need.



Indirect addressing

- Matlab doesn’t really do indirect addressing, but indexing is similar.
- You use the index to determine the location in memory of the data we want to access.

Structure Indexing example

- Load a SAC file into memory (use sacread).
- The returned structure will contain a series of fields with all the information about that file (sampling interval, event location, etc).
- These are 88 fields (approx). If you want to write a function that returns a specific field using its ordinal number, you would need a VERY long switch statement, like this…

Structure Indexing example

```
function value = return_sac_header(item)
    global sacdata;
    global is_file_open;

    if is_file_open
       switch item
         case 8
                  value = sacdata.delta;
         case 9
                  value = sacdata.depmin;
         case 10
                  value = sacdata.depmax;
         . . .
         . . .
    end
end
```

Structure Indexing example

```
function value = return_sac_header(item)
    global sacdata;
    global is_file_open;

    if is_file_open
       fields = fieldnames(sacdata);

       if item >= 8
        value = sacdata.(fields{index})
       end
    end
end
```

Other Tricks and Tips

- **ALWAYS**, indent and separate your code. Never write code without separating it into blocks.
- If you don’t indent or separate your code you might end with something like this:

Try to read this code

```
%close all
% script to plot results
if isunix() slash = '/'; else slash = '\'; end
site_list = []; for j=1:size(SITE_LIST,1) site = SITE_LIST(j,:); if nargin == 5
for h=1:size(plot_sites,1)
if strcmp(plot_sites(h,:), site) == 1
site_list = [site_list; site];
end
end
else
site_list = [site_list; site];
end
end
```

Now, try this one

```
%close all
% script to plot results

if isunix()
    slash = '/';
else
    slash = '\';
end

site_list = [];

for j=1:size(SITE_LIST,1)
    % load each site to make the plot
    site = SITE_LIST(j,:);

    if nargin == 5
        for h=1:size(plot_sites,1)
            if strcmp(plot_sites(h,:), site) == 1
                site_list = [site_list; site];
            end
        end
    else
        site_list = [site_list; site];
    end
end
```

Comment your code!

- It is always a good idea to comment what you are doing by adding a simple human readable sentence that explains the purpose of the line (even if it seems stupid). I guarantee than after 3 weeks without looking at your code, the stupid comment helps a lot. Make sure the comments make sense!
- In Matlab, vectorization of code can make it pretty unintelligible, so it's always better to explain what you did.

Example of unintelligible code

```
 % make a matrix with the values on the first column
tifb = [-obs(:,12)/K*f2 zeros(n, m)];
index = 1:m*n;
index = reshape(index,n,m);
index = bsxfun(@minus, index, n*(obs(:,15)-1));
```

Much better

```
 % make a matrix with the values on the first column
tifb = [-obs(:,12)/K*f2 zeros(n, m)];
 % make a vector of n*m elements
index = 1:m*n;
 % reshape it to be n x m
index = reshape(index,n,m);
 % subtract from each elem the val on matrix obs-1 multiplied by n
 % this will create a matrix with values that are 0 at the position of the
 % receiver.
index = bsxfun(@minus, index, n*(obs(:,15)-1));
```