DIY method development with MATLAB: Getting started

Aapo Nummenmaa

Motivation

Going beyond existing methods

- Examples (fMRI preprocessing):
  - Suppose you need to remove every 10th scan from your fMRI time series before doing GLM.
  - Solution 1:
    - LOAD data into MATLAB.
    - Throw away the unwanted volumes.
    - WRITE the modified dataset from MATLAB and input that into GLM.
  - Solution 2 (?):
    - Introduce nuisance regressors into GLM for unwanted time points.
  - Solution 3 (??)
    - Try to find an fMRI analysis program X that has a specific option of omitting volumes.

Re-inventing the wheel minimally…

- MATLAB is a "platform-independent" thing.
  - Scripts and functions written on a MAC work on a PC as such.
- MATLAB is a rather "high-level" programming language.
  - The code is relatively easy to read.
  - Existing programs are relatively easily modified.
- Many programs have been written in MATLAB.
  - MATLAB has a large number of built-in functions.
  - It is likely that something related to your problem already exists.
- Many routines for input/output data from/to MATLAB to different formats have been written.
  - MATLAB itself has extensive tools for reading & writing data.

Learning by doing!

- Unlike philosophy, you cannot learn math, physics, statistics, or any practical stuff *just* by reading books!
  - First, you need to do the simplest case with pencil and paper.
  - Then, you can really learn a great deal more by *simple* MATLAB simulations.
  - Even a *toy* version of the real-world situation gives some kind of concrete picture.
  - ANYBODY CAN DO IT!!!!
- If you want to invent a better wheel, you need to know how the old wheel works!
  - Sometimes it pays off to re-implement something that already exists in some other program.
    - After doing so, you will have a deeper understanding of the problem.

Preliminaries

Getting started: Help!

- MATLAB is your best friend.
  - In many case, the documentation texts are quite informative and educational.
- You even get help to using command “help”:

```
>> help help
   HELP Display help text in Command Window.
      HELP, by itself, lists all primary help topics. Each primary topic
      corresponds to a directory name on the MATLABPATH.
```

- By typing helpdesk into the MATLAB command line, an interactive help system launched.
  - You can start browsing and searching for various things.

The MATLAB editor: Quite convenient!

- Highlight a piece of script & press F9:

```
clear all

addpath /Applications/freesurfer/matlab/;
```

```
>> clear all

addpath /Applications/freesurfer/matlab/;
>>
```

- The Editor also gives you useful “warnings” and even “programming tips”!
- You can start MATLAB without the desktop / editor.
- You can run MATLAB through “Emacs” in a similar fashion.
  - Requires some configuration work.

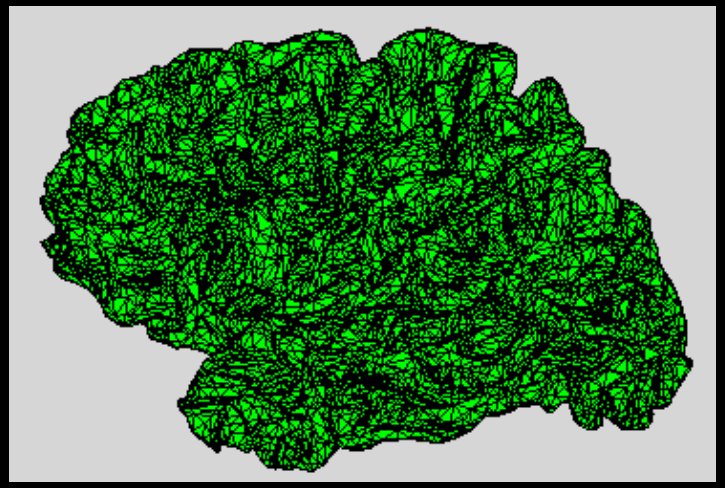
Some ubiquitous commands

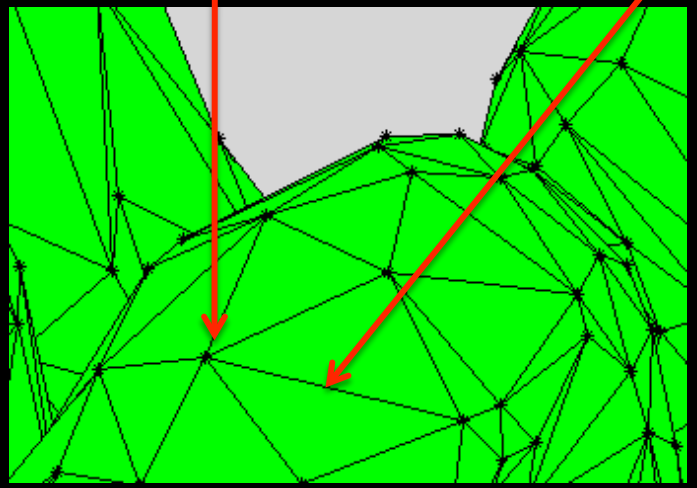
- "size":
  - Tells you the size of a variable
- "zeros":
  - You can create a matrix filled with zeros.
    - Useful for allocating memory.
- ".(*, /, …)" element-by-element operations:
  - X.*Y multiplies elements of equal-sized arrays.
- "repmat":
  - Create a replicate of array:

```
>> x=[1 2]; repmat(x,1,3)

ans =

     1     2     1     2     1     2

>> x=[1 2]; repmat(x,3,1)

ans =

     1     2
     1     2
     1     2
```

Warm-up: Surface rendering with MATLAB

- Discrete surface consists of "vertex points" and "edges":





- Surface or "Patch" objects use triangulation.
- In MATLAB, you need two lists of numbers:
  - "Vertices" are the coordinates of surface points.
  - "Faces" tell which three vertices form a given triangle.

Creating & manipulating patch objects

- Patch is created by specifying the “Faces” and “Vertices”.
  - Command “get” can be used to study the Patch object:

```
get(P_lh);
        AlphaDataMapping = scaled
        Annotation = [ (1 by 1) hg.Annotation array]
        CData = []
        CDataMapping = scaled
        DisplayName =
        FaceVertexAlphaData = []
        FaceVertexCData = []
        EdgeAlpha = [1]
        EdgeColor = [0 0 0]
        FaceAlpha = [1]
        FaceColor = [0 0 0]
```

  - Command “set” can be used to modify the Patch object properties:

```
set(P_lh,'EdgeColor','black','FaceColor','green')
```

Some MATLAB resources

- The Mathworks website has a wealth of info.
  - File exchange: http://www.mathworks.com/matlabcentral/fileexchange/
    - Lots of cool stuff, but be mindful when using these…
- FreeSurfer has a MATLAB toolbox
  - Read surfaces, MRI volumes etc.
- MNE for MEG/EEG analysis has MATLAB toolbox.
- SPM is based on MATLAB.
- Googling "MATLAB tutorial" gives about 9,170,000 results.

Phase 1: The primordial script

Prepare – Compute – Analyze

- Try to organize the necessary steps into 3 categories:
  - *Prepare*
    - Load necessary data, FreeSurfer surfaces, etc.
    - Put these into appropriate MATLAB variables.
  - *Compute*
    - Implement computational routines to get the desired result.
    - This is typically the most “time-consuming” part.
  - *Analyze*
    - Implement “post-processing” methods for visualization of the result in MATLAB.
    - Outputting into other software (FreeSurfer, MNE) etc.

Rationale

- First, write everything in a one big script:
  - Start by just writing comment lines for each task:

```
%%%% 1) LOAD FREESURFER SURFACES

%%%% 2) DEFINE LOCATIONS OF MEG SENSORS

%%% 3) CALCULATE MEG FIELDS FOR ALL SURFACE POINTS
```

  - Then start filling in the necessary pieces of code.
  - You can debug your code as you go:
    - Make sure that your FreeSurfer surface is correctly formatted.
    - Make sure your MEG sensors are what you want.
    - Check that your results look logical & consistent.

“Advanced” uses of the MATLAB editor

- Your “primordial script” is likely become pretty lengthy!
  - Use “cell mode” to move between blocks of script.
    - Inserting `%%` into the beginning of a line creates a cell.
    - You can evaluate the whole cell and jump to next.
    - You can turn the cell mode on/off from the Editor top panel.

```
%%%%CREATE A PATCH OBJECT WITH DESIRED VERTICES & FACES

%%
clf; cameratoolbar; axis equal off;
P_lh=patch('Faces',faces_lh_red,'Vertices',vertices_lh_red);

set(P_lh,'EdgeColor','black','FaceColor','green');

set(P_lh,'Marker','*');

%%
```

“Advanced” uses of the MATLAB editor (cont.)

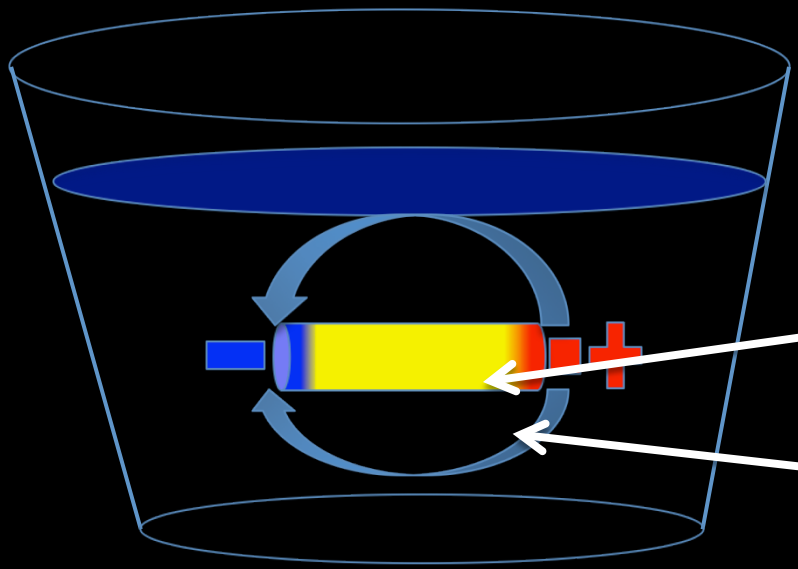
- Use “code folding” toggle hiding parts of script.
  - MATLAB editor can fold pieces of code under command blocks.
  - File -> Preferences… -> Editor / Debugger -> Code Folding
    - You can create “fake logical tests” like `if 1 = = 0` to:
      - 1) determine if a piece script is evaluated of not (change 0 -> 1).
      - 2) To hide the piece of script.

General “advice”

- Try to make it work first!
- Use understandable variable names: “vertices_lh” instead of “v_l”.
  - Your & everybody else’s working memory is limited!
- Don’t worry about the efficiency of the computational implementation.
  - If you need to do something *a couple of times*, it does not matter if it takes 1second or 5 minutes.
- Try to make sure that your result is correct.
  - Use more time to make “sanity checks”.
  - Plot your variables to see if they are what you think!
- Later, you can try to analyze and speed things up.
  - Your slow version will be useful as a reference point.

Example project: Magnetic fields of current dipole

- To get charges going, we must have a current source.
- What happens if a battery gets into salt water?
  - Ionic currents in the water flow to close the circuit.
  - Currents generate a magnetic field (B-field).



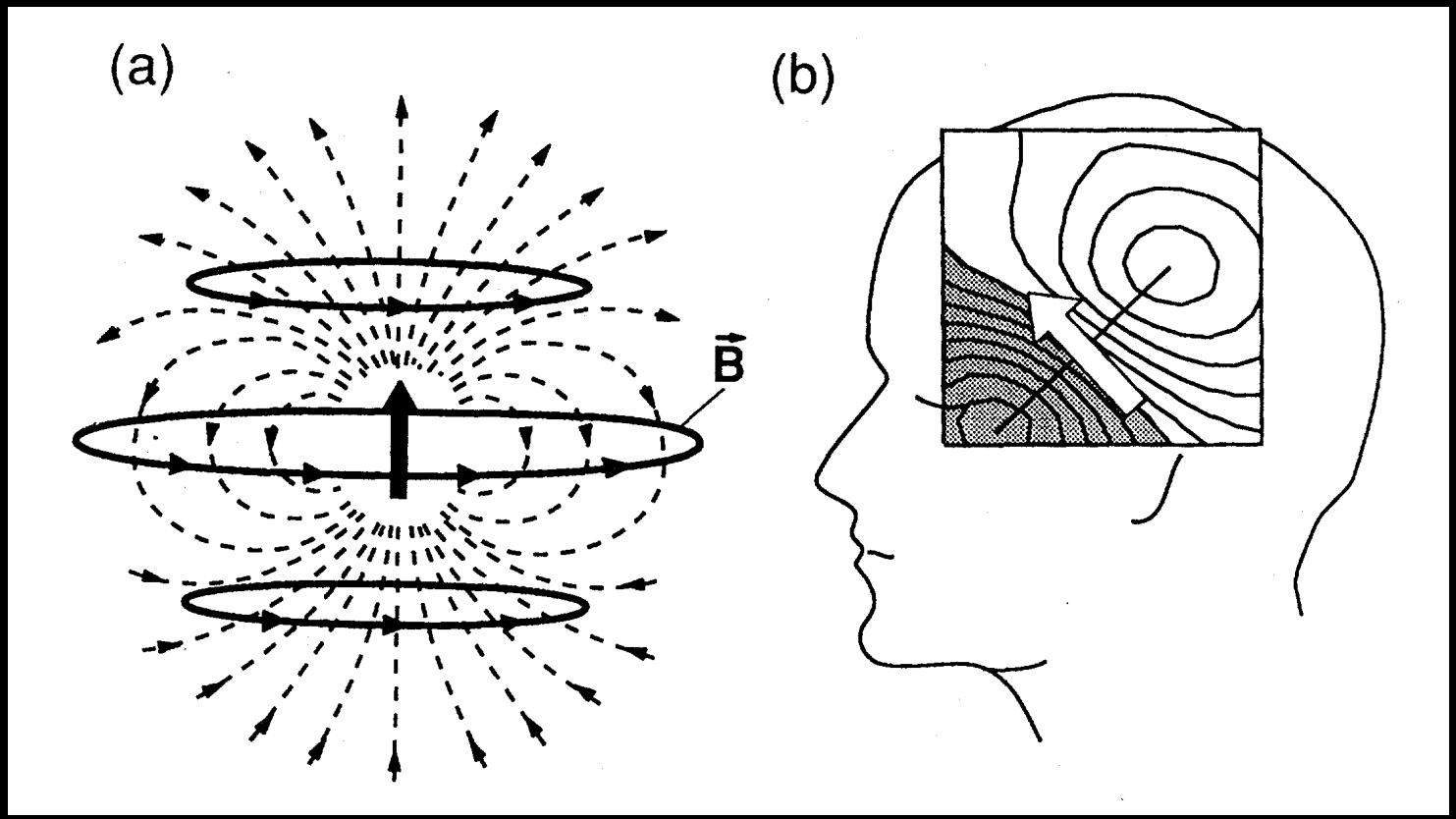
We assume an "infinitely large" bucket:
NOT A VALID ASSUMPTION IN PRACTICE!

Primary current

Volume currents

Magnetic field of the current dipole

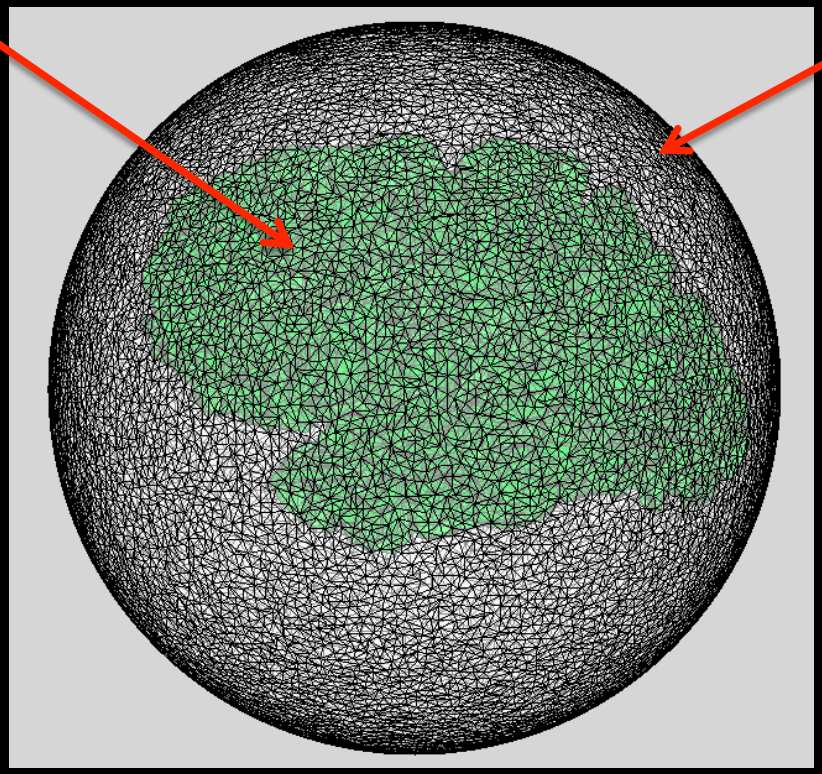
- The B-field circulates the dipole in right-hand sense.



From Hamalainen et al, *Rev. Mod. Phys.*

Step 1: Prepare

- Load FS white matter surface to serve as our "dipole / source space"
- Load FS spherical cortical surface to serve as our "B-field / sensor space".



Step 2: Compute

- The formula for the B-field is:

$$\vec{B}_\infty = (\mu_0 / 4\pi)\vec{Q} \times \vec{D} / \left\|\vec{D}\right\|^3$$

$\mu_0 = 10^{-7}$ [*henries / meter*]

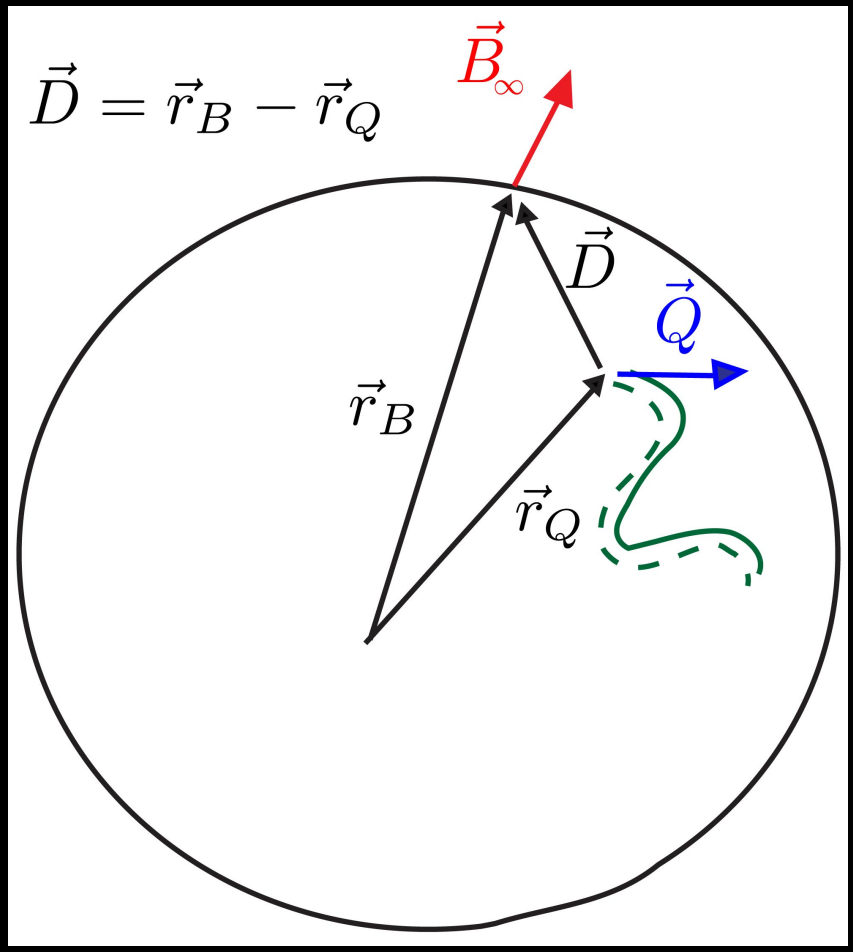
Permeability of vacuum

$\vec{Q}$ [*amperes* × *meter*]

Dipole moment (strength) of battery

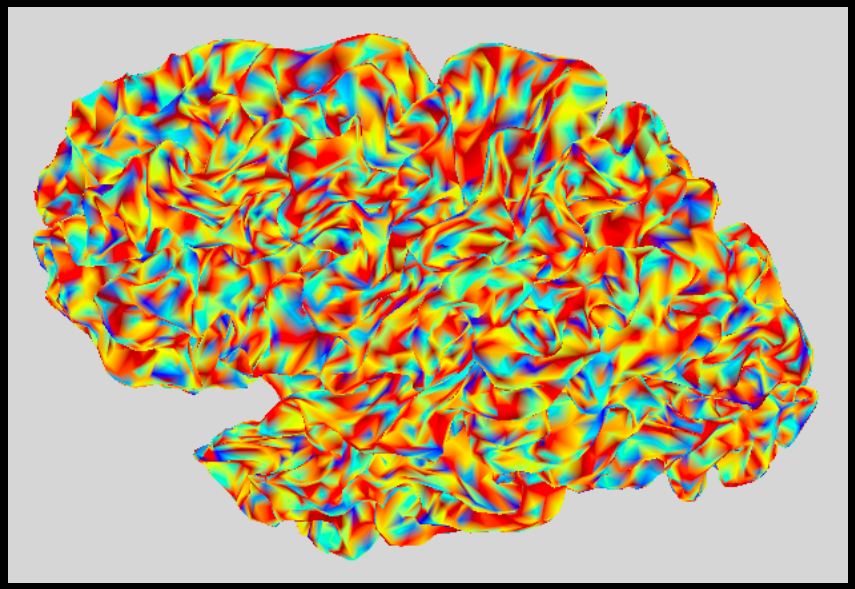
$\vec{D}$ [*meters*]

Distance from source to sensor



Step 3: Analyze

- For the MATLAB “patch” object, we can:
  - Specify a value at each vertex, that will be displayed as a color.
  - This is similar to what FreeSurfer does when “overlaying” fMRI data.



```
set(P_lh ,'EdgeColor','none','FaceColor','interp','FaceVertexCData', our_vector);
```

Phase 2: Writing parts of the script as functions

Minimizing clutter

- At some point, your workspace will be filled with variables:
  - All the temporary variables will remain unless cleared.
  - Increased risk of using wrong variables in wrong places.
- At the same point, you will also start to get annoyed by:
  - Having too many cells & things to evaluate.
  - Going back and forth the script changing values in different places.
- It is time to write some functions!

Example: The preparation block

You could start writing the function as:
[faces_lh_sou_red,vertices_lh_sou_red, …]=prepare_surfaces('surface1','surface2',…)

With a complicated problem, you will have many input and output variables!

```
%%%%%%%%%%%%%%%%%%%%%PREPARE%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% LOAD THE SOURCE SPACE // CORTICAL SURFACE
if 1==1;

source_space_file='lh.smoothwm';

[vertices_lh_sou_fs, faces_lh_sou_fs] = read_surf(source_space_file);

%%%%INDEXING STARTS FROM 0 IN FREESURFER, FROM 1 IN MATLAB

faces_lh_sou=faces_lh_sou_fs+ones(size(faces_lh_sou_fs));

vertices_lh_sou=vertices_lh_sou_fs/1000; %%%DIVIDE BY 1000 TO CONVERT TO METERS

%%%%%%%REDUCE PATCH SIZE TO FACILITATE COMPUTATION & VISUALIZATION

sou_red_frac=0.01;
[faces_lh_sou_red,vertices_lh_sou_red]=reducepatch(faces_lh_sou,vertices_lh_sou,sou_red_

end;
```

Using structures

- Structures are “variables containing variables”:

Beatles.John=[1 0 0 0]; Beatles.Paul=‘Still Alive!’; etc…

- The upshot is that:
  - You will only have the variable “Beatles” in your workspace!
  - You can mix different types of variables: John is numerical, Paul is string…

General approach

- Define inputs, parameters, and options as structures:
  - This allows easy book-keeping of variables.
  - It helps you to actually devise a suitable function!

```
input.X=X; input.Y=Y;
param.alpha=alpha; param.beta=beta;
opt.option1=option1; opt.option2=option2;
```

```
output=function(input,param,opt);
```

- There is price to pay:
  - Your variable names tend to become longer, more typing…
  - For "quick & dirty" implementation, you might want to "return" to original script variables:
    - alpha=param.alpha; X=input.X;
      - You can just copy-paste the original script inside a function…

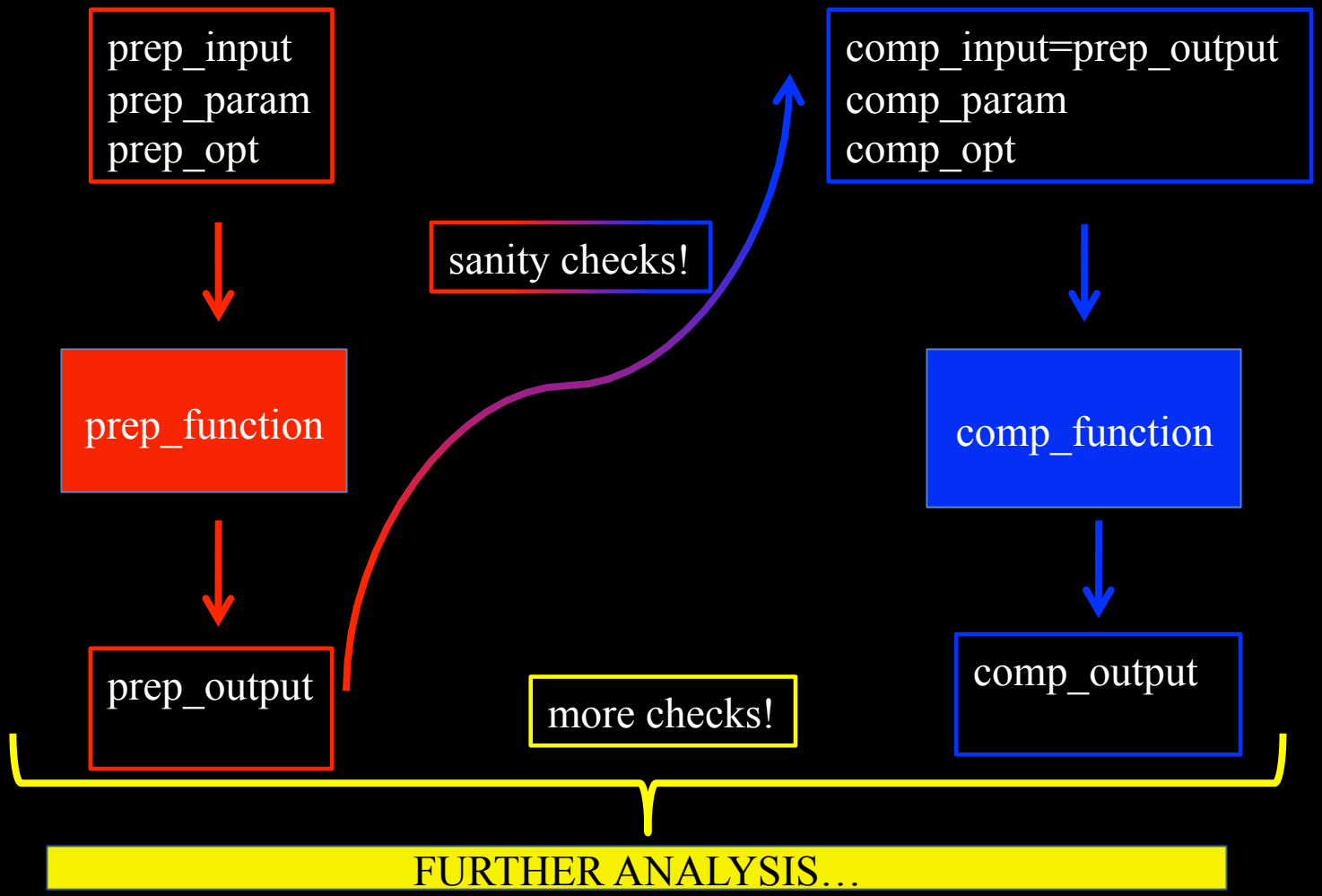
"Bad programming & memory usage!?"

Example: Improved version of the preparation block

- With this approach, the evolved version of the “primordial script” looks like:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%PREPARE%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%DEFINE INPUTS
input_prep.source_space_file='lh.smoothwm';
input_prep.sensor_space_file='lh.sphere';

%%%%DEFINE PARAMETERS
param_prep.sou_red_frac=0.01;
param_prep.sen_red_frac=0.01;

%%%%DEFINE OPTIONS
opt_prep.get_full_surfaces=1;

%%%%DO THE PREPARATION
[ output_prep ] = prepare_surfaces_for_Binf( input_prep,param_prep, opt_prep );
```

Summary: The analysis workflow



Improving computational efficiency

General considerations

- Once you have a working system, you can probably improve it a lot by relatively simple considerations!

- MATLAB = MATrix LABoratory
  - MATLAB likes dealing with matrices and vectors!
  - Most functions accept vectors as inputs.

- Two basic principles:
  - Try to avoid loops!
    - Compute with vectors as much as possible.
  - Pre-allocate memory for your variables:
    - MATLAB can “add” stuff to variables on the fly, but this slows things.

Loop reduction example: computing B_∞

Let's look at my first attempt in calculating B_∞

```
N_sou=size(vertices_lh_sou_red,1);
N_sen=size(vertices_lh_sen_red,1);

Q_x=[1 0 0];          %%%DIPOLE POINTS IN X-DIRECTION
Q_y=[0 1 0];          %%%DIPOLE POINTS IN Y-DIRECTION
Q_z=[0 0 1];          %%%DIPOLE POINTS IN Z-DIRECTION

%%%ALLOCATE SPACE FOR B-field MATRICES

B_inf_Qx=zeros(N_sen,N_sou,3);
B_inf_Qy=zeros(N_sen,N_sou,3);
B_inf_Qz=zeros(N_sen,N_sou,3);

mu0=4*pi*10^(-7);   %%%PERMEABILITY OF VACUUM

for jj=1:N_sen;
    for ii=1:N_sou;

        D=vertices_lh_sen_red(jj,:)-vertices_lh_sou_red(ii,:);

        B_inf_Qx(jj,ii,:)=(mu0/(4*pi))*cross(Q_x,D)/(norm(D)^3);
        B_inf_Qy(jj,ii,:)=(mu0/(4*pi))*cross(Q_y,D)/(norm(D)^3);
        B_inf_Qz(jj,ii,:)=(mu0/(4*pi))*cross(Q_z,D)/(norm(D)^3);


    end;
end;
```

Two nested loops!

Loop reduction example (cont)

```
Q_x=repmat([1 0 0],N_sou,1);          %%%DIPOLE POINTS IN X-DIRECTION
Q_y=repmat([0 1 0],N_sou,1);          %%%DIPOLE POINTS IN Y-DIRECTION
Q_z=repmat([0 0 1],N_sou,1);          %%%DIPOLE POINTS IN Z-DIRECTION

%%%ALLOCATE SPACE FOR B-field MATRICES

B_inf_Qx=zeros(N_sen,N_sou,3);
B_inf_Qy=zeros(N_sen,N_sou,3);
B_inf_Qz=zeros(N_sen,N_sou,3);

mu0=4*pi*10^(-7);   %%%PERMEABILITY OF VACUUM

for jj=1:N_sen;

     D=repmat(vertices_lh_sen_red(jj,:),N_sou,1)-vertices_lh_sou_red(:,:);

     norm_D2=repmat((sum(sqrt(D.^2),2).^3),1,3);

     B_inf_Qx(jj,:,:)=(mu0/(4*pi))*cross(Q_x,D)./norm_D2;
     B_inf_Qy(jj,:,:)=(mu0/(4*pi))*cross(Q_y,D)./norm_D2;
     B_inf_Qz(jj,:,:)=(mu0/(4*pi))*cross(Q_z,D)./norm_D2;

end;
```

Q_x, Q_y, and Q_z and D are now matrices!

Random acts of stupidity

- Try to avoid computing the same thing if not necessary.
- Move everything out of the loop that does not depend on that loop index.

```
for jj=1:N_sen;

     D=repmat(vertices_lh_sen_red(jj,:),N_sou,1)-vertices_lh_sou_red(:,:);

     norm_D2=repmat((sum(sqrt(D.^2),2).^3),1,3);

     B_inf_Qx(jj,:,:)=(mu0/(4*pi))*cross(Q_x,D)./norm_D2;
     B_inf_Qy(jj,:,:)=(mu0/(4*pi))*cross(Q_y,D)./norm_D2;
     B_inf_Qz(jj,:,:)=(mu0/(4*pi))*cross(Q_z,D)./norm_D2;

end;
```

Evaluating performance

- MATLAB has a "profiler" that evaluates time spent in different computations in a detailed manner:

As an alternative to the `profile` function, select **Desktop** > **Profiler** to open the Profiler.

- A for preliminary tests, "cputime" is a handy command:
  - "cputime" counts cumulative CPU time used by MATLAB.
  - This is a better measure than "normal" time!

```
t = cputime; surf(peaks(40)); e = cputime-t

e =

    0.4667
```

Aftermath

Is there such a thing as “too friendly”? Beware!

- MATLAB often sort of tries to guess what you are doing
  - It does not give errors in many “ambiguous” situations.
- You’re allowed to do crazy stuff: instantiate empty variables and keep adding things:

```
>> x=[]; x=cat(1,x,[1 1 1]); x=cat(1,x,[1 1 1])

x =

     1     1     1
     1     1     1
```

- Many functions perform operations along a certain dimension: look for example command “mean”.
  - Always make sure your variables have correct size!!
- The “norm(X)” gives always a number:
  - matrix norm if given a matrix, vector norm if given a vector.

Finalizing, commenting, documenting

- After you have made something useful that works:
  - Some more debugging … (are you sure it works?)
  - Try to get rid of the "bad programming" tricks.
  - Make argument checks so that only correct inputs to functions are accepted.
  - Make enough comments so that at least *you* can read it after a year.
  - Try to write some info in the help / header section for other people that could potentially use your stuff.

Homework problems for aficionados

- Level 1
  - Find all errors, bugs, & stupidities in the script.
  - Get rid of the “bad programming” implementation.
- Level 2
  - From physics, we know that the flux of B_∞ trough any closed surface (such as the sphere) should be zero. Is it?
- Level 3
  - Try to make the computation of B_∞ without any loops!

Final words

- Consult your favorite computer scientist for “good programming style” tips!
  - You should aim ultimately at writing programs that other people can use as well.
- Everything can and *should* be adjusted to personal taste.
  - The point of the talk is just to help you get started.

THANK YOU FOR LISTENING!