Using Functional Data Analysis (FDA) to analyze reach trajectories

1.0 – Installation instructions

All of this code requires Matlab with the Statistics toolbox and has been most extensively used (e.g. tested) on Matlab 7.10.0 R2010a. I know that versions prior to Matlab 7 will not run these scripts as the ANOVAN (from the Stat toolbox) function was not extended to include random (repeated) measures prior to version 7.

1.1 – FDA scripts

To download the appropriate Matlab functions go to:
ftp://ego.psych.mcgill.ca/pub/ramsay/FDAfuns/Matlab

Follow the instructions located in: INSTALL.MATLAB.WIN.txt

1.2 – Explanation of all scripts in the zip file

1.2.1 – Accessing Private Matlab functions
Several necessary functions are not immediately accessible in Matlab because they are "Private". I have never found out how to make them public, so my work around is to copy and paste the needed m-files to somewhere on my path. Included in the downloaded zip-file "FDA_for_reach.zip" are the following Matlab m-files:

statgetargs.m
idummy.m
termcross.m

These files will also be available on your system (assuming you have Matlab and the statistics toolbox) if you search for them and you can copy and paste locally if desired.

1.2.2 – Matlab functions written by others
I use several functions written by others. The first are available on the Matlab Central File Exchange (http://www.mathworks.com/matlabcentral/fileexchange/22870-huynh-feldt-epsilon-general-procedure) and are used when correcting for within-subjects (random) factors (e.g. Greenhouse-Geisser correction):

GenCalcHFEps.m
GenOrthogComps.m

The other functions written by others are from the Sequence Production Laboratory (http://www.mcgill.ca/spl/) and integrate some of Dr. Ramsay's FDA code into a more user-friendly format:

evalFDA.m
smoothFDA.m

1.2.3 – Matlab scripts I've written:

One script that I've written to use FDA to normalize my data

normalizeFDA.m

Next, two scripts I use to generate repeated measures functional analysis of variance.

getRMMeans.m: to get the means for each participant for each condition (required as input into a repeated measures design)

fanovan.m: to run the functional anova (this code is an adaptation of Matlab's anovan function and the MAJORITY of the code is from Matlab, with only minor tweaks by me).

Finally, there are two scripts (and accompanying sample data) showing how to run each of the two phases of analysis described below

normalizeExample.m (example data in: sampleTrial.mat)

fdaExample.m (example data in: sampleFDAData.mat)

2.0 - Sources:

The procedures described here rely on the theory and the scripts developed by Dr. Jim Ramsay: http://www.psych.mcgill.ca/misc/fda/

To download the appropriate Matlab functions go to: ftp://ego.psych.mcgill.ca/pub/ramsay/FDAfuns/

For corresponding texts written by Dr. Jim Ramsay and his colleagues:

Functional Data Analysis (Springer Series in Statistics) by J.O. Ramsay & B.W. Silverman

Applied Functional Data Analysis by J.O. Ramsay & B.W. Silverman

Functional Data Analysis with R and MATLAB (Use R) by J.O. Ramsay, Giles Hooker & Spencer Graves

I am also grateful for code I received from Dr. Caroline Palmer and her students Janeen Loehr and Erik Koopmans in the McGill Sequence Production Lab (http://www.mcgill.ca/spl/) that demonstrated the use of some of Dr. Ramsays FDA scripts.

Finally, I am indebted to Dr. Paul Gribble (http://gribblelab.org/) at the University of Western Ontario for introducing me to FDA in his class, and for much help in getting this working.

My colleagues and I have published several recent papers using these techniques if you want demonstrations of them in action:

Gallivan, J.P., Chapman, C.S., Wood, D.W., Milne, J.L., Ansari, D.A., Culham, J.C., & Goodale, M.A. (2011) One to four, and nothing more: Non-conscious parallel object individuation in action. *In Press*. Psychological Science.

Chapman, C.S., & Goodale, M.A. (2010). Obstacle avoidance during online corrections. Journal of Vision. 10(11), 1-14.

Chapman, C.S., Gallivan, J.P., Wood, D.W., Milne, J.L., Culham, J.C., & Goodale, M.A. (2010). Short-term motor plasticity revealed in a visuomotor decision-making task. Behavioral Brain Research. 214(1), 130-134.

Chapman, C.S., Gallivan, J.P., Wood, D.W., Milne, J.L., Culham, J.C., & Goodale, M.A. (2010). Reaching for the unknown: Multiple target encoding and real-time decision-making in a rapid reach task. Cognition. 116(2), 168-176.

3.0 - Introduction:

In this document I will outline how I use FDA techniques to analyze reaching data. Personally, I use an OPTOTRAK system to collect the x, y and z position of infrared markers at a high frame rate (usually ~200 Hz). Of course, any system giving you position data over time (or in theory any functional data) will be amenable to the techniques described here.

For me, the use of FDA is divided into two categories (dealt with as such in this document). The first step is fitting the mathematical functions to the raw data. This then allows for curve normalization. It is this first step that relies on the code written by Dr. Ramsay. The second step is running a functional ANOVA to compare differences in trajectories across time (or space, or whatever your function is measured over). For this step I have made some custom modifications to Matlab's anovan function (note ANOVAN is from the Statistics toolbox and needs to be from a new version of Matlab at least Matlab 7 R14, and possibly as recent as 7.4 R2007b – currently I am running 7.10 R2010a).

4.0 – FDA for normalization

4.1 – Overview

Included in this zip file is the script I created (normalizeFDA.m) which performs normalization of a reach trajectory to either movement time or to any of the three dimensions of space (x, y, z). Essentially this script provides a wrapper for the functions written by Dr. Ramsay so that they can be applied to reach data that has 3-dimensions. This normalization procedure is applied to each trial separately and produces a normalized (i.e. same number of data points) trajectory (required for statistical analysis and averaging etc.). First, a few comments about how I use normalization, then I will describe some specifics of the procedure. For an example, see normalizeExample.m included in the zip file.

4.2 – Limitations

4.2.1 - Pre-processing

It is worth noting that the data I normalize has already undergone some significant pre-processing before I apply the FDA scripts to fit the mathematical functions. Specifically, I fill in any points of missing data (using inpaint_nans, link here: http://www.mathworks.com/matlabcentral/fileexchange/4551) then I apply a low pass filter (usually 10hz) to the raw (and filled) trajectories. I then use custom code to determine the onset and offset of my reaches (these definitions are usually velocity criteria, but vary according to the experiment). So – what actually gets passed to my normalization function is a trimmed reach, with no missing data that is filtered.

It is important to realize that the FDA scripts written by Dr. Ramsay can actually do most of this work for you. By fitting an appropriate mathematical function you can interpolate the missing data and demand a certain degree of smoothness (the same as what you get when you filter your data). I have to admit, my lack of using FDA for this comes from finding FDA after already developing a succinct pre-processing pipeline. I want to experiment with replacing my filling and filtering with FDA but have not done so. Anyone who is or does use FDA for this, please pass along your scripts!

4.2.2 – Lack of curve registration

The way I have currently implemented my normalization script, it does not do any curve registration. For example, you may want to align your normalized reach trajectories at onset, offset *and* at peak velocity. Right now, the normalization code, by the nature of normalization, only aligns at onset and offset. However, Dr. Ramsay has developed a host of functions for more advanced curve registration and theories (refer to the texts listed in section 2) for why registered curves can give you better overall comparisons. I can see this technique being particularly useful for data where specific peaks (e.g. ERP data) are definitive of a certain effect. By aligning data to *several* such landmarks, you improve your averaging and remove some noise. That being said, I have only rudimentarily explored the use of curve registration, and thus, unfortunately, it is currently lacking in my implementation.

4.2.3 – Over-fitting my data

Since my data is already pre-processed (filled and filtered) I have elected to over-fit mathematical functions to my data. As you'll see when you start exploring Dr. Ramsay's texts and scripts you have many choices to make when you are fitting functions to data. For example, one of the simplest functions would be to fit the best straight line, thus giving you one parameter (aka beta, or coefficient) that defines your fit, but likely producing a poor fit. Scaling up, you may want to fit an nth order

polynomial, thus increasing the number of parameters, and improving your fit. Of course, the main trade-off is between the number of parameters of your fit and the goodness of your fit. Usually, when you over-fit your data, you end up fitting to noise, and thus you introduce noise into the remainder of your analysis. You also end up with a whole bunch of parameters that become hard (or impossible) to interpret on their own.

In my case, given my filtering procedure, I am fairly confident that any fluctuations in my trajectory are meaningful components of the movement and not random noise. As a result, I'm not scared that when I over-fit, I am going to introduce noise into my data. Rather, by over-fitting, I feel confident that I am exactly capturing the shape of the movement, and that this shape is preserved in the mathematical definition of the trajectory.

So just how over-fit is my data? Well, it is pretty much as over-fit as you can be. I fit b-splines to my data. Specifically, I use 6th order b-splines (which have been shown to be good at preserving data to the 3rd derivative, or acceleration in our case). When fitting b-splines, you have to decide how many splines to fit. For example, you could fit just one spline 'knotted' at the beginning and end of a trajectory. This example is equivalent of fitting a (in my case) 6th order polynomial. However, if you decide to fit 2 splines to the data, 'knotted' at the beginning, *middle* and end of the data, then you are fitting 2 6th order polynomials to the data, one for the first half and one for the second half. The knotting procedure guarantees a smooth transition between the two splines. You can already see that there are two choices to make when fitting b-splines – what order of polynomial for each spline and how many splines total (determined by the number of 'knots')? In my extreme over-fit case I fit 6th order splines with a knot at *EVERY* data point. So, if I had a 500ms reach, measured at 200Hz, I'd have 100 data points meaning I'd have 99 6th order splines (99 because the number of splines is one less than the number of data points...e.g. in the example with beginning, middle and end, we had 3 data points and 2 splines).

I don't pretend to think this is an ideal solution. As a result of this procedure I get an exact mathematical replica of my data, with an un-interpretable number of parameters. In my ideal world, I would figure out what the least number of least order splines could capture my data while still preserving the majority of the shape. This is exactly the type of solution that Dr. Ramsay proposes and he has several good points about how to proceed to find this 'perfect' fit (it's largely trial and error, iterative, and time consuming). If done properly, you can then make statistical comparisons of the *parameters* of the fit, each of which explains an important component of the data (essentially traditional fMRI GLM analysis is this type of procedure). However, for my purposes, given the nature of my data, I am content to over-fit, normalize and look at the resulting functional statistical comparison (see section 5).

4.3 – Normalization example

To see normalization in action you can use the script normalizeExample.m (which in turn requires the data in sampleTrial.mat, as well as the rest of the functions described in the Install section). This very simple script simply loads in the data, sets the parameters required for normalizeFDA then calls that function. The action therefore, is in the normalizeFDA function.

4.3.1 – Structuring your data

One important point about running this code is how to structure your data. The first point this becomes relevant for normalization is when calling normalizeFDA. Note – normalizeFDA expects data from a SINGLE trial, and returns a normalized version of that trial. For notes on how then to arrange the normalized trial data (of which there will many across an experiment) see section 5.

For the single trial, normalizeFDA expects the data to be in a cell array. Each cell within the array contains the three dimensional data from ONE tracked marker. In many cases we track two markers on participants when pointing, and three or more when grasping. Within the cell the three dimensional data is arranged as columns with x in the first column, y in the second column, z in the third column. So loading in the sampleTrial.mat file, and typing:

```
>>data{1}

ans =

      X      Y      Z (labels added by me)

    2.1177  -26.7850    7.4449

    2.0952  -26.6193    7.4242

    2.0945  -26.2624    7.3948

    2.1306  -25.6141    7.3648

    2.2200  -24.5493    7.3525

    2.3777  -22.9248    7.3913

      …       …       …
```

If this data had a second marker (it doesn't) then data{2} would also contain three columns.

In addition to expecting a cell array of data, normalizeFDA has the following arguments:

```
%toNormalize = a list of IRs that you want to normalize - this refers to
%the indexes of the data cell array

%normalizeFrames = number of frames you want for your normalized
%trajectories

%normalizeType
%1 = to time
%2 = to x distance
%3 = to y distance
%4 = to z distance

%frameRate = the frame rate of data collection
```

So the call:

```
normalizedReach = normalizeFDA(data,1,200,3,150)
```

Means that you are going to normalize data from the first cell in the data cell array, that the output trajectory will have 200 points, that you want to normalize to y-distance and that the original sample was collected at 150hz.

4.3.2 – The process of normalization

Regardless of whether you are doing temporal or spatial normalization, the first step for this process is to fit a mathematical function to the data. There are a bunch of different choices you can make at this step, some of which I spoke of above. The most important ones I made were fitting bsplines to my data (you can choose from any number of basis functions, like sine waves etc.) and to fit a spline between every two data points (as opposed to fitting a handful of splines to the whole trajectory). Finally, you need to decide what smoothing parameter to use. I have stuck with what others have used previously (see work from the SPL lab: http://www.mcgill.ca/spl/) but for more details about all of this, I urge you to read some of Dr. Ramsay's texts (see a list in section 2).

The heart of the curve fitting happens in the script smoothFDA. For example the call:

```
smoothFDA(time,IR1x,'10^-18',1,[],reachLength);
```

Means you fit bsplines to the data in IR1x (the x component of the data you are normalizing). The time variable (defined before as `time = 0:1/frameRate:(reachLength-1)/frameRate;`) just gives a time stamp for each data point in IR1x. The 10^-18 refers to the smoothness penalty, the 1 refers to the fact that I want a knot at every data point (i.e. fit a spline between every 2 data points). The [] would in theory all you to introduce landmarks to your fits (this is the curve registration I spoke of earlier) but I haven't played with this much. Finally by setting the final argument to the length of the reach I specify that I want one fit over the data (IR1x in this case).

Within smoothFDA the actual script doing the fitting is `create_bspline_basis`. At the end of the day smoothFDA is offering a very convenient wrapper for this function which ends of taking many of the same arguments, though they require some manipulation which smoothFDA does for you. If you wanted to play around with OTHER basis functions or other orders of splines, you would want to change the call to something other that create_bspline_basis or you would want to change the norder parameter value from 6 (line 58 in smoothFDA).

You'll notice that each dimension of my spatial data (e.g. x, y, z) gets its own mathematical fit. As far as I know, there is no way to do a single multi-dimensional fit.

The script normalizeFDA.m provides 4 normalizing options (set with the normalizeType input argument) the first to movement time and the last three to each of the three spatial dimensions. The fit to movement time is the most intuitive and most commonly used normalization technique. Each trajectory is defined at x points equally spaced in time (where x is equal to the input argument noramlizeFrames). For example if normalizeFrames = 100, then each point in the normalized data would represent 1 percent of movement time. Since we have a mathematical definition of our reach, we simply extract from the function the x equally spaced points.

I would approach time normalization (actually the same rule applies for all normalization) with some caution. Often studies of reaching report movement time differences between conditions but then also use time normalization to average the trajectories. In my mind this is a serious mistake – if you know there are meaningful time differences across conditions, normalizing these conditions relative to time and then looking for spatial differences can produce erroneous spatial differences. As a general rule I try to avoid normalizing to any parameter that significantly differs across conditions. For example, if my conditions differ in reach distance (e.g. y-distance) I should not normalize to y-distance.

Normalization to one of the three spatial dimensions follows the same logic as normalizing to time, but instead of extracting equally spaced time points you calculate the time points that correspond to equal chunks of space in the dimension you want to normalize to. For example, if you want to normalize to y-distance, and you want 100 normalizeFrames, then each point would correspond to 1 percent of y-distance. That is, we would find the times at which every percent of y-distance had been travelled. Since the hand generally moves slowly at the beginning and end of a movement (bell shaped velocity profile) this means you take points in time that are more spread out at the beginning and end of the movement (less distance over more time) and points in time that are tightly bunched together in the middle of the movement (more distance over less time).

In order to calculate when in time a set of equally spaced points (in the dimension of interest) occur it is necessary to first create a high resolution representation of the dimension to be normalized to. The higher the resolution of this representation, the more accurately you will find the exact moment in time that the trajectory passes the next chunk of space. For more comments on this process, refer to the comments in the normalizeFDA.m script. Once you have the high-res version of the dimension of interest, you find when in time each chunk of space has been fulfilled. For example, imagine we want to normalize to a y-distance of 400mm with 100 points. We would render a high res version of the y-trajectory (e.g. 2000 points) then look for when in time the trajectory was equal to 4mm (400/100) then 8mm then 12mm...and so on until we reached 400mm. We then extract from the two non-normalized dimensions (in this case x and z) trajectories from these newly defined time points.

One important choice I made for this spatial normalization was to normalize to the entire trajectory in a given dimension. That is, regardless of whether the hand was moving forward or backward positive or negative) in a given dimension, that was considered part of the “distance” for that dimension.

Finally, once you’ve extracted from each dimension a trajectory of the desired length (specified by normalizeFrames) normalized to the desired dimension (time or one of the three dimensions of space) the code re-packages the data and returns the normalized reach in a cell array called normalizedReach. Note, we take advantage of the fact that a mathematical function can be differentiated to calculate velocity for each dimension and thus return in each cell (corresponding to each IR) the three columns of normalizedSpace data and three columns corresponding to the velocity in each dimension. It is worth noting that comparing normalized velocities is tricky (i.e. if you’ve normalized them all in time then you’ve stretched them all different amounts) and I’ve found comparing non-normalized velocities (i.e. at each frame of actual movement) is more interpretable.

5.0 – Running functional ANOVAs

5.1 – Overview

Included in this zip file is the script I created (fanovan.m) which performs a functional ANOVA on one-dimension (e.g. lateral deviation or x) of a set of normalized trajectories. As with a regular ANOVA you are comparing across different conditions. The script is just a modification of the Matlab Statistics toolbox function anovan.m extended to a 2D matrix as opposed to 1D vector input. Here I only discuss examples of a functional *repeated measures* ANOVA as those are the only kind I have ever performed. In principle, the logic should extend to non-repeated measures ANOVAs but I have not actually tested this. Here I provide a few comments about some considerations when using functional ANOVAs, then walk through the steps involved. For an example, see fdaExample.m which uses data from sampleFDAData.mat.

5.2 – Statistical violation and functional interpretation

My implementation of a functional ANOVA is no different than running an ANOVA at every time point across a set of normalized trajectories. In fact, all I've done is tweak the Matlab Statisitcs toolbox function anovan.m to extend to all points in a matrix instead of just using a vector of data. The instinctive reaction is that this violates all sorts of statistical sense in that you are performing what amounts to hundreds (depending on how many data points are in each normalized trajectory) of statistical comparisons. What is imperative to realize is that while in practice you are performing many statistical tests, you are only doing so as a surrogate for ONE statistical comparison of an entire function. For example, imagine that you normalized your trajectories to 10 points. Would the fANOVA be any more or less valid than if you normalized your trajectories to 100 points, or even 1000 points for that matter? No. In fact, intuition tells you that the comparisons across these examples (10pt versus 100pt normalization for example) should give you the exact same results where the two examples would overlap (every 10th point would be shared). It should not be the case that the 10pt example is evaluated at a more liberal statistical threshold than the 100pt example.

In fact, it is the very notion that a mathematically defined function (as our normalized trajectories are) is scale invariant (i.e. can be defined with 10, 100, 1000 or any number of points) that drives home the point that what is being conducted is a single functional comparison that is merely implemented as many individual ANOVAs. It is required that functional data are input into the fANOVA and that functional statistical tests are returned. That is, in a conventional ANOVA, we put in a series of discrete data points (i.e. single points in time) and what gets returned is a single F-statistic with a corresponding single p-value. With a functional ANOVA, we put in a series of functional data vectors and what gets returned is a functional F-statistic with a corresponding functional p-value. This highlights an extremely important point. If you are analyzing functional data with a functional ANOVA it is imperative that you show the entire functional output. Being selective (i.e. interpreting only one region of a functional result) does violate statistical principles.

5.3 – FDA example

To see an example of an FDA see the fdaExample.m (which in turn requires the data in sampleFDAData.mat, as well as the functions described in the Install section).

5.3.1 – Structuring your data

Recall that the normalization procedure produces a vector for each dimension of a single trajectory that all have an equal number of frames. Imagine now that you are running an experiment where you get 320 trajectories from each participant. To structure this data such that it can be analyzed using an fANOVA you need to collect the data from each spatial dimension together in one matrix (recall, you can only run an fANOVA over one dimension at a time. In theory, it would be possible to perform an fMANOVA, but I have never tried this). So, consider you have 320 reaches from each of 4 participants normalized to 200 points: you'd have 1280 x-vectors, 1280 y-vectors and 1280 z-vectors (all vectors of length 200). For any dimension (e.g. x) you could thus create a 1280 x 200 matrix (number of trajectories by number of normalized points). This is exactly how I have structured my data and exactly what the fanovan.m function expects. I have added one level of structure to my data by storing each of the x, y and z matrices in one Matlab struct. Therefore, if you load in sampleFDAData.mat and type:

```
>> fdaMat


fdaMat =


    x: [1277x200 double]

    y: [1277x200 double]

    z: [1277x200 double]
```

You can see that each dimension of all 1277 reaches (3 trials were removed due to recording errors) is stored in one field of a struct called fdaMat. Typing:

```
>>fdaMat.x

ans =


   -3.1642   -4.6190   -4.7930   -4.9305   -5.0305… (to 200 values)

    0.1193   -0.6987   -1.3455   -1.8777   -2.3212

   -2.9035   -3.2173   -3.5417   -3.7164   -3.8190

   -0.0690   -0.2038   -0.1180    0.0815    0.3278

   -0.1727   -1.5653   -1.7505   -1.6561   -1.4079

       .
       .
       .
To 1277 trials
```

Reveals that you have a matrix for the x dimension where every row corresponds to the x-data for one normalized trajectory (therefore goes from 1:number of reaches) and every column corresponds to a point in the normalized dimension (e.g. percent time or percent distance, therefor 1:number of normalized points). The same structure is true for the y (fdaMat.y) and z (fdaMat.z) dimensions. When

actually performing the fANOVA, you'd pass in the matrix for one dimension of the data (e.g. fdaMat.x – actually when performing a repeated measures ANOVA you need to pass a matrix that reflects subject means, but more on that below).

Of course, we still need to group all of these trials (1277 in the example) according to the relevant conditions (or levels of independent variables). This is accomplished by creating a group cell array that contains cells for each grouping factor (or independent variable). Therefore, in the example data, if you type:

```
>> group

group =

    [1x1277 double]    [1x1277 double]
```

You see that we have two grouping factors (as indicated by the fact that we have 2 cells). Critically, within each grouping factor/cell you see that we have a vector with the same number of values as we have trials. That is, for every trial, we have provided the code for that trial on each of the factors. This is equivalent to using a program like SPSS where you might store raw data as a column of data, then have additional columns that code for each row, what condition that data corresponds to for a given factor.

In the example data, the first grouping factor corresponds to the type of display that participants saw when performing their reaches. We had 16 display types so each value within group{1} will be from 1 to 16. Thus typing:

```
>>group{1}

ans =

     6    13     4     2    11… to 1277 values
```

Shows you the coding values for the first grouping factor. Note, since the data within each cell of "group" is one dimensional it does not matter if it is arranged as a column or row (current example is a row).

It is possible to have multiple grouping factors (independent variables) in the group cell-array and each would just occupy its own cell with its own set of values. For simplicity, the current example has this one grouping factor that codes for condition. The last grouping factor codes for subject.

When using fanovan to run repeated measures functional ANOVAs (which is the only way I have used it thus far) the LAST grouping factor MUST identify the participant. That is, you could have any number of grouping factors (independent variables) stored in the cell array "group", but if you are running repeated measures the last factor always codes for subject. In the example, the last grouping factor is the one in the second position of the cell array (since the cell array is only of size 2). If you had 3 independent variables, then the subject code would be stored in the 4 element of the group cell-array. A good way to check that the subject codes are at the end of the group cell array is to type:

```
>> group{end}

ans =
```

```
1      1      1      1      1… to 1277 values (subject codes)
```

Which shows that the vector stored in the final position (indexed with the "end" index) correspond to the subject number. Since each participant contributed ~320 trials for the example data, the switch in the group{end} from 1 1 1 1… to 2 2 2 2… occurs at index 321 (indicated participant 1 contributed 320 trials).

5.3.2 – Getting subject's mean data for use in repeated measures fANOVA

The structure of data described in section 5.3.1 explains how to structure the raw data. However, when performing a repeated measures ANOVA you are actually comparing subject's mean data against each other – that is, you've moved from raw data to averaged data.

Included in the zip file is a function: getrmmeans.m which I've written to calculate subject's average trajectory data for each condition. It takes as input one dimension of the matrix data explained above (e.g. fdaMat.x) and the group cell-array (IMPORTANT it expects the last group cell to contain subject coding) and returns a new matrix of data where each row corresponds to a participants mean trajectory in one of the conditions (or, if you had multiple grouping factors, in one combination of the two conditions) and a new group variable coding the new average matrix data. With the example data, if you wanted to get the 4 subjects mean data across the 16 conditions specified by group{1} you would type:

```
[meansx,newGroupx] = getRMMeans(fdaMat.x,group);
```

Which would return a 64x200 matrix in meansx and a cell array in newGroupx with two cells and within each cell would be a 64 element vector. The 64 rows of meansx would correspond to each of 4 participants mean performance in 16 conditions (4*16 = 64).

Often, I want to perform an fANOVA on some subset of conditions. For example, if you wanted to perform a pairwise contrast, you would want data from only 2 conditions (see section5.3.4 below for more on this). As another example, for the sample data, I wanted to look only at reaches made to displays containing a target only in the upper left and/or upper right corners. There were four conditions that met this criterion:

Where group{1} == 1 (target in upper left and right, reach ends left)
Where group{1} == 2 (target in upper left and right, reach ends right)
Where group{1} == 13 (target in upper left only)
Where group{1} == 14 (target in upper right only)

To access ONLY this data, I can index into my group{1} cell using the find command as follows:

```
idx = find(group{1}==1 | group{1}==2 | group{1}==13 |group{1}==14);
```

"idx" now contains reference to only the indexes of elements belonging to one of those four conditions. If we pass this subset of data to the getrmmeans function, it will return means on ONLY the four conditions we've specified in idx. For example typing:

```
[meansx,newGroupx] = getRMMeans(fdaMat.x(idx,:),{group{1}(idx)
group{2}(idx)});
```

Will return a meansx with 4 rows per participant (therefore 16 total rows) and a group cell array with two cells, each containing a 16-element vector. Notice that we pass only a subset of the fdaMat.x data as specified by the (idx,:) index – that is we are saying get us only the rows of fdaMat.x that correspond to the idx variable we defined. Likewise, we generate a subset of the grouping data by constructing a cell array during the function call that has only a the idx subset of data in each cell.

With the new mean data, you can now plot mean trajectories and run the fANOVA (see subsequent sections for more on this).

5.3.3 – Running a repeated measures fANOVA

Now that you have the data formatted as explained above, you are ready to run the fANOVA. Continuing from the example above, we use the data from the sampleFDAData.mat and select only those trials with a target only in the upper left and/or upper right corners (that is, we used the idx variable as described above to get a meansx matrix of functional averages from each of our subjects in each condition and a newGroupx cell array that categorizes those conditions). We can then run the fANOVA by typing:

```
>> [px,corrPx,tx,statsx] = fanovan(meansx, newGroupx, 'model','full',
'random',length(newGroupx),'varnames',{'condition' 'subject'});
```

The meansx and newGroupx variables are described above. The remaining arguments in the fanovan function call refer to different options (see comments in the fanovan.m for more details):

The 'model' option allows you whether you want to look for only main effects, only main effects plus 2-level interactions or the full model (main effects and all levels of interaction). Selecting the full model is appropriate unless you have a specific reason to restrict your analysis otherwise.

The 'random' option is critical to running the repeated measures ANOVA. In fact, Matlab versions prior to version 7 did not allow for random factors and thus the fanova was much more difficult to implement. However, the 'random' option allows you to specify which grouping variables are to be treated as random factors. In my case, I always only have one random factor (subject) and it is always stored in the final position of my group cell array – thus using length(newGroupx) always returns the index position of my only random factor. Finally, the 'varnames' option allows you to assign names to the different grouping factors. In this case, there are only two, my condition and my subject.

The fanovan will return three important pieces of data. A matrix containing the functional p-values for each main effect and interaction in your model, a matrix containing the corrected p-value (only differs from the p-value matrix for repeated measures designs with independent variables with more than 2 levels) and the stats data structure which contains a lot of detailed information about the resulting statistical test.

That stats data structure is a work in progress and currently is used more for information than for function. When using the anovan function for tests without random factors, the stats data structure can be used to perform multiple comparisons (post-hoc tests of means). If you want to know more about

the stats structure see the documentation for the anovan.m function. I implement follow-up tests to significant fanovas as explained below.

One relevant piece of information available in the stats structure is in the terms field. From the above (where the stats structure is named statsx) typing:

```
>> statsx.terms

ans =

     1     0
     0     1
     1     1
```

Shows the terms that make up each row in the functional p-value. So, in this case, it shows that the first row corresponds to the effect for only first grouping variable (condition = 1) and not the second (subjects = 0). The second row corresponds to the effect for only the second grouping variable. The third row corresponds to the interaction (both terms go into this row).

It is important to note that when running a repeated measures ANOVA the error term is no longer the error associated with the entire design, but rather the error associated with each subject (i.e. how each subject varies across the conditions). Thus the appropriate error term for this single factor design is the condition by subject interaction. This is evident when inspecting a few of the fields from the stats structure returned from the fanovan function. Following from the example, typing:

```
>> statsx.msterm

ans =

  1.0e+003 *

  Columns 1 through 5

    0.0003    0.0009    0.0017    0.0027    0.0040…
    0.0008    0.0022    0.0032    0.0050    0.0073…
    0.0002    0.0002    0.0002    0.0003    0.0003…
```

Shows the values for the mean squared error for each of the three terms (recall, terms correspond to the stats.terms field explained above). Most importantly, focus on the third term which represents the condition x subject interaction. Typing:

```
>> statsx.msterm(3,:)

ans =

  Columns 1 through 5

    0.2307    0.2009    0.2247    0.2546    0.2896…
```

Here we see the values for just the interaction term. Notably, the values for this interaction term correspond exactly to the values for the *error* used for the other terms in the fanova. This is shown by examining the msdenom field of the stats structure:

```
>> statsx.msdenom

ans =

  Columns 1 through 5

    0.2307    0.2009    0.2247    0.2546    0.2896
    0.2307    0.2009    0.2247    0.2546    0.2896
         0         0         0         0         0
```

Here we see that the error used for both of the first two terms (and critically for our interests, the first term, which represents our condition) is the interaction of condition with subject. Note that in a repeated measures design where you input the subject means, there is no error for the subject by condition interaction. To compute the corresponding f-statistics, you use the appropriate msterm / msdenom. The p-value is the computed using the F-cumulative-distribution function in Matlab (fcdf) with the appropriate degrees of freedom. All of this is calculated in the fanovan function if you want to see the specific commands for these operations.

I have added an extra step into the fanovan function whereby a corrected degrees of freedom is calculated to correct for sphericity (correlations among conditions). This calculation is not vectorized and thus is not optimized for Matlab – it simply calculates the correction factor (epsilon) for every point in the function and applies it (I use the epsilon calculation functions from the Matlab central file exchange, see section 1). I am not sure this is an ideal solution and I welcome any improvements or suggestions for this.

I'm not going to belabor any more statistical details, but for those interested, I'd recommend you dig into the fanovan function a bit more to see how the anova is being computed. Very briefly, Matlab implements a model comparison approach whereby two models are compared that differ only in deletion of one term (a full vs. reduced model – for example the model with a main effect of condition vs one without that effect). If you want more details regarding this approach, I refer you to the excellent text book by Maxwell and Delaney: Designing Experiments and Analyzing Data: A Model Comparison Perspective (http://www.amazon.com/Designing-Experiments-Analyzing-Data-Perspective/dp/0805837183).

5.3.4 – Running follow up tests (i.e. functional pair wise comparisons)

Running a pair-wise comparison follows the identical steps outlined above for running the functional anova in general. The difference is that you identify only two levels of a factor to compare, rather than the multiple (in the example above, 4) levels used in the omnibus fANOVA. As the example in fdaExample.m shows to follow up the original test that compared reaches made to displays containing a target only in the upper left and/or upper right corners we wanted to perform a functional pair-wise comparison of a two-target trial that ended left versus a one-target trial that ended left. First we identify only those trials:

```
>> idx = find(group{1}==1 | group{1}==13);
```

Then we get the subject means for only those conditions:

```
>> [meansx,newGroupx] = getRMMeans(fdaMat.x(idx,:),{group{1}(idx)
group{2}(idx)});
```

Finally we run the fANOVA with only these two levels compared:

```
>> [px,corrPx,tx,statsx] = fanovan(meansx, newGroupx, 'model','full',
'random',2,'varnames',{'condition' 'subject'});
```

Note that a repeated measures comparison involving only two levels of a given factor will never need to be corrected for sphericity.

The same process could be applied to compare the two-target right versus one-target right trials (here I only show the step to get the appropriate indexes):

```
>> idx = find(group{1}==2 | group{1}==14);
```

And of course to compare the two-target left and two-target right trials (only indexing shown):

```
>> idx = find(group{1}==1 | group{1}==2);
```

5.3.5 – Interpreting and viewing the results of an fANOVA

Once you've got your functional anova result (captured by the functional p-value) it's really up to you to choose how you want to display that result – with one IMPORTANT rule: It is essential that whatever method you choose for displaying your result you display the ENTIRE functional result. The power of the functional anova is that we treat an entire function as a single piece of data, but as a consequence we must treat the result as a single output. To selectively show only those areas of a function that are significant is a violation of the spirit of the functional analysis and of statistical principles.

In the fdaExample.m file I show one simple way of plotting the results which is to plot the p-value across the function (in this case across percent Y-distance). Then, you can add in a line which corresponds to some alpha value (here I use the conventional $p<0.05$). If you want, you can plot the px and corrPx values to see how the correction for sphericity changes the result (remember that there is only a difference for fANOVAs with more than 2 levels of a factor). If you are interested in some other ways we've visualized the result, I encourage you to check out some of our recent papers (see section 2 for a list).

Finally – if you want to plot the results for the functions (i.e. not the stats, but the data itself) you use a similar procedure as described above. That is, you identify the trials associated with a condition of interest, then plot only those trials (or an average of those trials) using the plot3 command. Of course, if your data aren't 3D then using the regular plot command would suffice. In fdaExample, I show how to plot the grand average for the four conditions that were compared in the fanova.

```
idx1topL = find(group{1}==13); %this will be green
idx1topR = find(group{1}==14); %this will be black
idx2topL = find(group{1}==1); %this will be blue
```

```
idx2topR = find(group{1}==2); %this will be red

%plot in 3D
figure; hold on;
plot3(mean(fdaMat.x(idx1topL,:)),mean(fdaMat.y(idx1topL,:)),mean(fdaMat.z(idx
1topL,:)),'g','linewidth',2);

plot3(mean(fdaMat.x(idx1topR,:)),mean(fdaMat.y(idx1topL,:)),mean(fdaMat.z(idx
1topL,:)),'k','linewidth',2);

plot3(mean(fdaMat.x(idx2topL,:)),mean(fdaMat.y(idx1topL,:)),mean(fdaMat.z(idx
1topL,:)),'b','linewidth',2);

plot3(mean(fdaMat.x(idx2topR,:)),mean(fdaMat.y(idx1topL,:)),mean(fdaMat.z(idx
1topL,:)),'r','linewidth',2);
```

It is worth noting that plotting the grand average of all trials of one condition (as shown above) is not exactly representative of a repeated measures ANOVA. Rather, plotting the average of participant's mean trajectories is better. In cases where each participant contributed the identical number of trials to each condition, then this point is moot – however, as is often the case, we remove bad (or error) trials, and thus each participant might not contribute the same number – skewing grand average results toward the behavior of participants who contributed the most trials to a given condition.

To calculate and plot this average of averages, we use the getrmmeans function to produce subject averages for each condition (in each dimension). We then use the grouping variable returned by the getrmmeans function to index the condition of interest. Finally, we use plot3 to plot our results:

```
%get subject averages (as above when doing FANOVA) but do so for each
%dimension.  Note all the newGroup will be identical
idx = find(group{1}==1 | group{1}==2 | group{1}==13 |group{1}==14);
[meansx,newGroupx] = getRMMeans(fdaMat.x(idx,:),{group{1}(idx)
group{2}(idx)});
[meansy,newGroupy] = getRMMeans(fdaMat.y(idx,:),{group{1}(idx)
group{2}(idx)});
[meansz,newGroupz] = getRMMeans(fdaMat.z(idx,:),{group{1}(idx)
group{2}(idx)});

%get the appropriate indexes for each condition.  Recall that the
%getRMMeans function renumbers your conditions (assings 1 through the
%number of unique conditions, in this case 4)
idx1topL = find(newGroupx{1}==3); %this will be green
idx1topR = find(newGroupx{1}==4); %this will be black
idx2topL = find(newGroupx{1}==1); %this will be blue
idx2topR = find(newGroupx{1}==2); %this will be red

%plot in 3D - add in labels and titles
figure; hold on;
plot3(mean(meansx(idx1topL,:)),mean(meansy(idx1topL,:)),mean(meansz(idx1topL,
:)),'g','linewidth',2);

plot3(mean(meansx(idx1topR,:)),mean(meansy(idx1topL,:)),mean(meansz(idx1topL,
:)),'k','linewidth',2);
```

```
plot3(mean(meansx(idx2topL,:)),mean(meansy(idx1topL,:)),mean(meansz(idx1topL,
:)),'b','linewidth',2);

plot3(mean(meansx(idx2topR,:)),mean(meansy(idx1topL,:)),mean(meansz(idx1topL,
:)),'r','linewidth',2);
```

Finally, it is often worthwhile to show both the functions and the error associated with the functions. However, as anyone who reports repeated measures knows, selecting the correct error term to show for this type of analysis is tricky. We have most often elected to represent error as the average of participant's mean standard error. That is, for the error, we show an average of how far each person was away from their own mean at each point in the curve. Again, if you are interested, I encourage you to take a look at some of our recent papers, listed in section 2. In the case of a functional pairwise contrast, a useful – and statistically meaningful - way of visualizing the results is to plot the difference between the two functions across time (or whatever the unit of the function is) and then show the 95% confidence interval of that difference. Where the 95% CI band diverges from 0, that difference is significant at 0.05.

6.0 – Conclusion

My use of functional data analysis and functional anovas has just started and is sure to evolve. So far it has offered a set of tools for dealing with reach trajectory data that conventional statistics did not. It is far from perfect, but it has certainly been useful for me. As a result, this document and the associated code is very much a work in progress - if you find any errors, or have any suggestions, recommendations or additions, I would love to hear them.