User guide General Monotone Model

Michael R. Dougherty, University of Maryland

This document describes how to implement MatLab® code for running statistical tests using the General Monotone Model (GeMM).

Mathematical details of GeMM are provided in:

Dougherty, M. R. & Thomas, R.P. (in press). Robust decision making in a non-linear world. *Psychological Review*.

Users should cite this paper when using GeMM in research papers or presentations.

I. Overview

The General Monotone Model (GeMM) is a statistical algorithm for predicting rank orders from a set of predictor variables. The primary difference between GeMM and standard Least Squared statistical procedures is that GeMM fits data using the method of paired comparisons to minimize the number of rank-order inversions between the predicted and observed values, rather than minimizing the squares differences between the predicted and observed variables. The use of paired comparisons to fit the data means that GeMM can be used with both ordinal or interval scaled variables and that it is robust to non-linearity's in the criterion variable. The predicted values produced by GeMM are metrically scaled (i.e., the distances between adjacent values can be interpreted). However, these metric values are not guaranteed to be in the original units as GeMM does not include an intercept term. A function to estimate the quantitative values is currently under construction.

The matlab code written by Michael Dougherty enables you to use GeMM on your own data. At present, this code is written to perform all-subsets regression. All subsets regression finds the best overall model that trades parsimony for fit. In essence, GeMM finds the configuration of the parameters that maximizes the rank-order correspondence between the model's prediction and the observed criterion variable.

The matlab code *does not* automatically examine interaction terms. Interaction terms would have to be entered in as a separate column of data in the input dataset. Having done this, the code does not treat columns containing interactions any differently

The code was written to allow for you to use both least-squares and GeMM. Hence, it is easy to compare the predictive accuracy of GeMM with that of OLS using the same model selection criterion. Additionally, there are a lot of options left in the code from when it was in the development stage. These options allow the user to test out alternative versions of the fit criterion and to generally explore the behavior of GeMM on their own.

The user can choose between three model selection criterion. These three fit criteria are the Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC) and the Risk Inflation Criterion (RIC). It is recommended that the monotone version of the Bayesian Information Criterion be used.

The "Quick guide to using GeMM" provides the parameter settings for running GeMM using settings that have been extensively tested.

II. Required Toolboxes and functions

Toolboxes

Statistics Toolbox

GeMM uses the least-squared solution to seed the search for regression parameters (the beta weights). The regression function is called from Matlab® to generate a subset of the beta weights that get entered into genetic algorithm.

Communications Toolbox

This toolbox is needed for optimizing the genetic algorithm function, buildbetas_pie_sample_GA.m. The default setting for buildbetas_pie_sample_GA.m is to use graycodes to construct all possible configurations of the beta weights variables (excludes interaction terms). If you do not have access to the communications toolbox, you will need to open the buildbetas_pie_sample_GA and switch off the use of the graycodes. This is done simply by changing the following line of code:

```
gray ='onn';
```

to

```
gray ='off';
```

Note, the gray codes specifications are produced only for data sets that include 9 or fewer predictors. If your dataset includes more than 9 predictors, gray codes are automatically shut off.

Files and functions

GeMM.m
This is the main program for running GeMM analyses. This script calls all remaining functions.

buildbetas_pie_sample_GA.m
This function constructs beta weight vectors using a genetic algorithm.

gemmfit_v3.m
This function estimates fit index and model selection procedure on estimation sample.

gemmcross_v3.m
This function to computes prediction accuracy on cross validation sample.

normalizedata.m

This function rescales the data to help optimize parameter search. Because GeMM does not need an intercept parameter, it does not estimate precise quantitative values. To narrow the parameter search space and optimize the search for the best fit beta weights, its advisable to range normalize the criterion variable. This is done by default by GeMM, but is easily set to use the original values or the rank transformation

dichomizedata.m

This function performs median splits on predictors. This is used only to compare GeMM with the Take The Best Heuristic, which requires dichotomous predictors.

givedeciles.m

This function computes a decile matrix, which is used to determine prediction accuracy for extreme scores. This is currently in beta testing (Jan 4, 2011).

III. Using GeMM on your own data

Quick Guide Overview

This section provides a quick step-by-step procedure for analyzing your data with GeMM. Before proceeding, however, it's important to point out that the GeMM analysis is very computationally intensive, so it may take a bit for GeMM to finish crunching the numbers. The computational intensity stems from two components: The fit criterion and the parameter search. The fit criterion in GeMM requires that GeMM examine the dominance structure of all pairwise comparisons, of which there are N(N-1)/2, for each vector of beta weights. The number of beta vectors is determined by the settings used on the genetic algorithm.

To illustrate the computational intensity, imagine a data set with 100 observations. There are exactly 100(100-1)/2=4950 paired comparisons to be made between the observed criterion variable and the predicted values. Each beta weight yields a different set of predicted values. What we want to do is find the beta weights that maximize the rank order correspondence between the criterion and predicted values. This requires that we create 100's or 1000's of vectors of beta weights. Using a genetic algorithm, say we create an initial population of 6000 vectors of beta weights. This means there are 6000 x 4950 = 29,700,000 paired comparisons on this initial population alone. But we're not done, because what we want to do is find the *best* weight vector, which may not have been included in the initial population. So, we select out the good vectors from initial population and use these to create a new population of 6000 vectors, which are then used to fit the data once again. This process repeats until the solution appears stable. If we repeat this process 10 times then we have 6000 x 4950 x 10 = 297,000,000 paired comparisons.

Despite this computational complexity, GeMM runs surprisingly fast on most modern computers for modest sample sizes (N <=100). However, note that the number of paired comparisons is an exponential function of N, so as sample size increases, there will be greater increases in the time needed for GeMM to complete. With sample sizes of greater than 500, you might want to run the analysis over night.

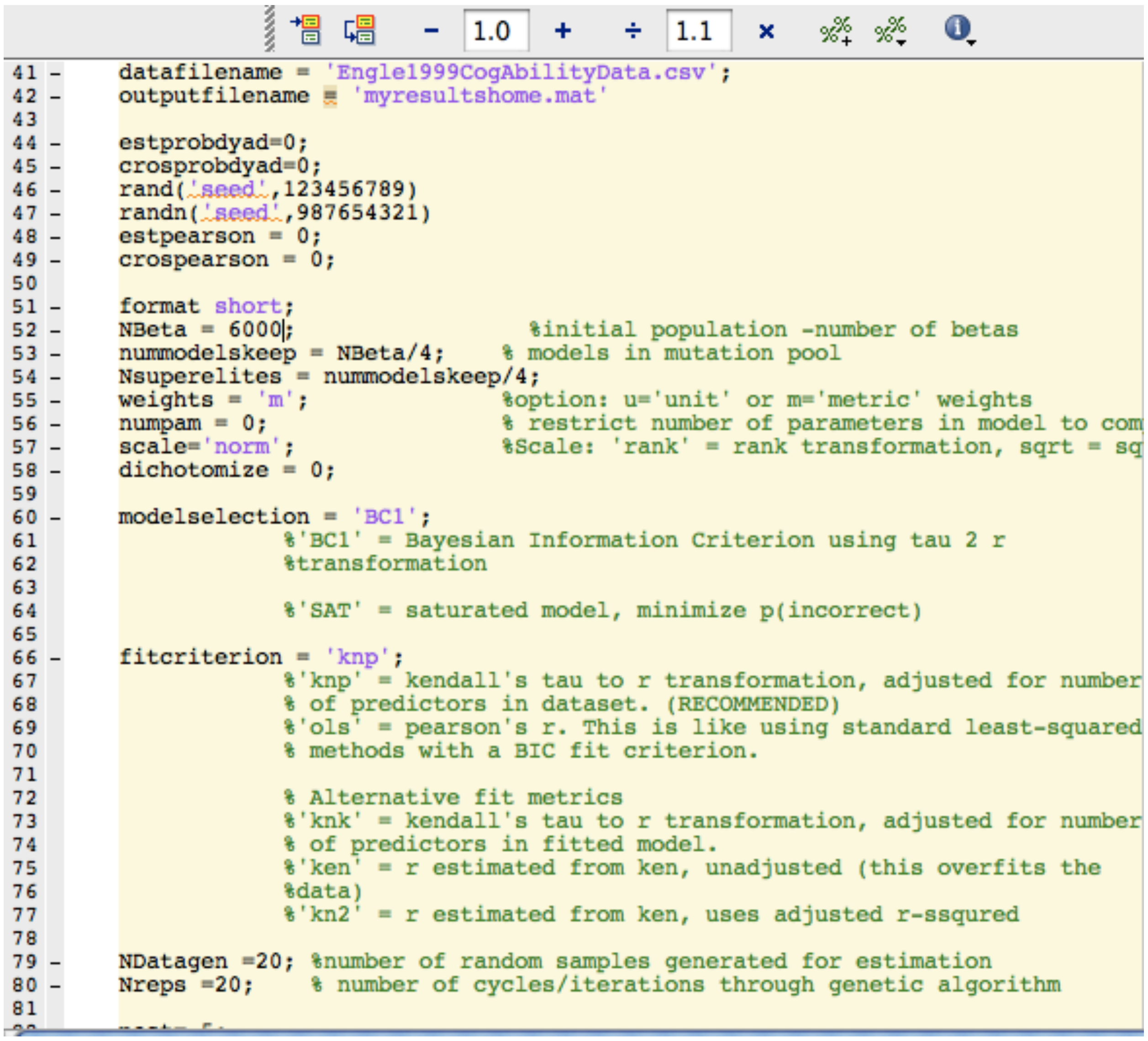
Computational time will also increase with the number of predictors in the dataset. GeMM is currently optimized to work with datasets with 9 or fewer predictors, but it will work with any number of predictors, so long as N > (P+1), where P is the number of predictors.

Step-by-Step procedure

Step 1. Open MatLab.

Step 2. Make sure all of the source files are in the same directory.

Step 3. Open *GeMM.m*



```
41 -     datafilename = 'Engle1999CogAbilityData.csv';
42 -     outputfilename = 'myresultshome.mat'
43
44 -     estprobdyad=0;
45 -     crosprobdyad=0;
46 -     rand('seed',123456789)
47 -     randn('seed',987654321)
48 -     estpearson = 0;
49 -     crospearson = 0;
50
51 -     format short;
52 -     NBeta = 6000;                  %initial population -number of betas
53 -     nummodelskeep = NBeta/4;     % models in mutation pool
54 -     Nsuperelites = nummodelskeep/4;
55 -     weights = 'm';                 %option: u='unit' or m='metric' weights
56 -     numpam = 0;                    % restrict number of parameters in model to com
57 -     scale='norm';                  %Scale: 'rank' = rank transformation, sqrt = sq
58 -     dichotomize = 0;
59
60 -     modelselection = 'BC1';
61                 %'BC1' = Bayesian Information Criterion using tau 2 r
62                 %transformation
63
64                 %'SAT' = saturated model, minimize p(incorrect)
65
66 -     fitcriterion = 'knp';
67                 %'knp' = kendall's tau to r transformation, adjusted for number
68                 % of predictors in dataset. (RECOMMENDED)
69                 %'ols' = pearson's r. This is like using standard least-squared
70                 % methods with a BIC fit criterion.
71
72                 % Alternative fit metrics
73                 %'knk' = kendall's tau to r transformation, adjusted for number
74                 % of predictors in fitted model.
75                 %'ken' = r estimated from ken, unadjusted (this overfits the
76                 %data)
77                 %'kn2' = r estimated from ken, uses adjusted r-ssqured
78
79 -     NDatagen =20; %number of random samples generated for estimation
80 -     Nreps =20;    % number of cycles/iterations through genetic algorithm
81
```

Step 4. Set data file name in the following line (approximately line 41):

```
datafilename = 'Engle1999CogAbilityData.csv';
```

Datafilename must be column formatted, such that the criterion variable (the variable you wish to predict) is in the first column and the predictor variables are in columns 2 through the P+1, where P is the total number of predictors. The MatLab function csvread is called to read the data file.

MatLab supports many file types, but use of other file types will require that you modify the source code.

Step 5. Set output file name (approximately line 42).

```
outputfilename = 'myresultshome.mat'
```

Data are saved as a .mat (MatLab binary format) in a data structure called simresults. The results will also be displayed at the end of the analysis.

Step 6. Set fit criterion
If you wish to model data using GeMM, set fitcriterion = ‘knp’.
If you wish to model data using OLS, set fitcriterion = ‘ols’.

Step 7. Set model selection criterion.

Several options for model selection are included in the software. But only the Bayesian Information Criterion has been extensively tested with GeMM.

Bayesian Information Criterion (finds model that minimizes the BIC)
modelselection = 'BC1';

Akaike Information Criterion (finds model that minimizes the AIC)
modelselection= ‘AIC’;

Risk Inflation Criterion (Finds model that minimizes the RIC)
modelselection=’RIC’;

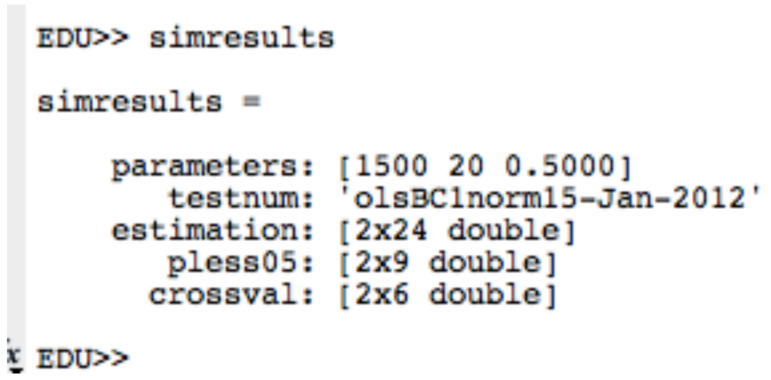
Saturated model (finds model that minimizes 1-tau, which will most often include all parameters)
modelselection=’SAT’;

Step 8. Run the code

Interpreting the output.

The output of a GeMM analysis is a vector of numbers providing various measures of estimation accuracy or fitness, a vector representing the selected model parameters, and the parameter values provided by the standard OLS regression. For now, you can ignore the OLS weights, and focus on what is described below.

Data are stored in a data structure called simresults:



```
EDU>> simresults

simresults = 

    parameters: [1500 20 0.5000]
       testnum: 'olsBC1norm15-Jan-2012'
    estimation: [2x24 double]
       pless05: [2x9 double]
      crossval: [2x6 double]

EDU>>
```

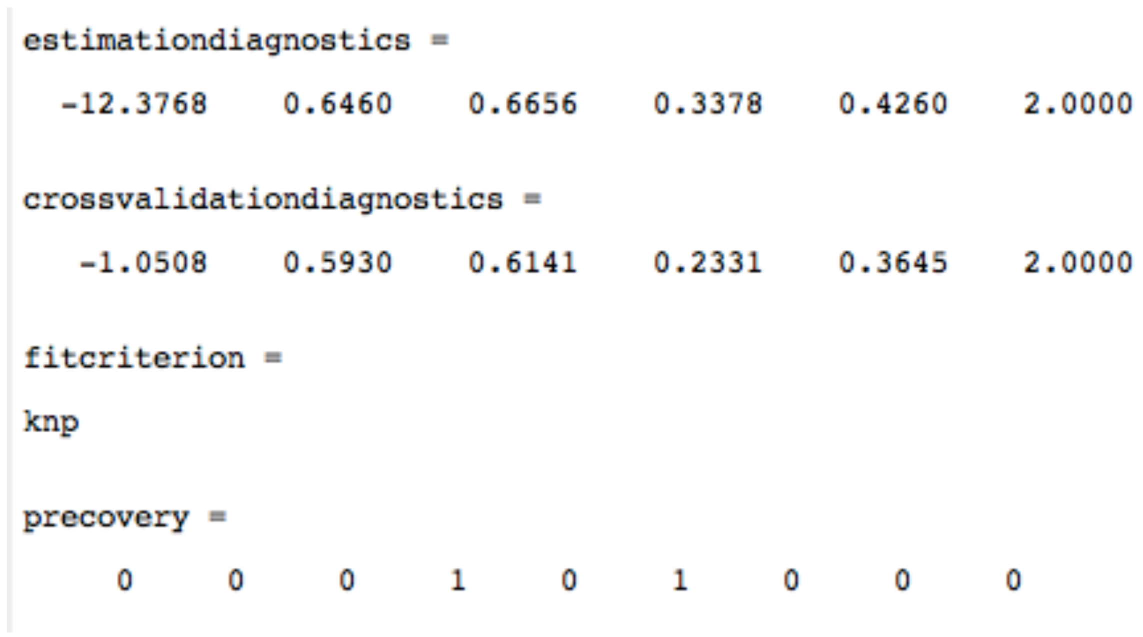
“parameters” is a vector with three values: The number of vectors that are used in the genetic algorithm to build each generation’s population. The initiation population is 4x this number. The second value is the number of elites retained at each generation. The third value is the percentage of the data included in estimation.

Testnum is a string containing fit type (e.g., OLS versus KNP), model selection procedure (BC1), how data were rescaled (norm) and the date of the test.

“estimation” is a vector containing the fit index, the GeMM estimated beta weights, and the OLS estimated beta weights (these can be used comparatively to GeMM’s estimated weights).

“pless05” is a vector of 1’s and 0’s indicating which parameters were significant using OLS regression (these can be used for a quick and dirty assessment of how well GeMM agrees with the OLS-wald solution).

“crossval” is a vector containing the predictive accuracy of the estimated GeMM. This data structure is filled only if pest<1.0. “pest” is the proportion of the data used for estimation, it can range from 0<pest<=1.0.



```
estimationdiagnostics =

  -12.3768    0.6460    0.6656    0.3378    0.4260    2.0000


crossvalidationdiagnostics =

   -1.0508    0.5930    0.6141    0.2331    0.3645    2.0000


fitcriterion =

knp


precovery =

     0     0     0     1     0     1     0     0     0
```

Model fit: The first six numbers provide various measures of prediction accuracy and model fit.

Fit Index	P(concordance)	Success	Tau	Pearson r	k
-12.3768	0.6460	0.6656	0.3378	0.4260	2.0000

"Fit Index" is an index of model fit, as given by the model selection procedure used: BIC, AIC, or RIC (whichever is chosen). Better fit is indexed by a smaller (more negative) value. The null model will have a value of zero; models with fit criterion less than 0 are preferred to the null. *Note, the BIC computed using knp cannot be compared to a BIC computed using OLS. To compare the two, compute the value of BICt' using the value of tau estimated from the OLS model.*

For GeMM, the BIC is computed by approximating r from tau (this is the BICt' provided in Dougherty & Thomas, in press). Kendall (1970) showed that r could be estimated from tau using sin((pi/2)(tau)). For our purposes, we bias tau downward using sin((pi/2)(tau*omega)) where omega = (N-P-1)/(N) and P is the number of predictors in the dataset.

"P(concordance)" is simply the proportion of concordant pairs, C/[N(N-1)/2], where C is the number times the dominance structure of the predicted and observed pairs agree. See Dougherty & Thomas for further details.

"Success" is a metric that corrects the concordance rate by a guessing. When the predicted value for a pair of observations is tied, success takes this into account by assigning half of the guesses as concordances and half as disconcordances.

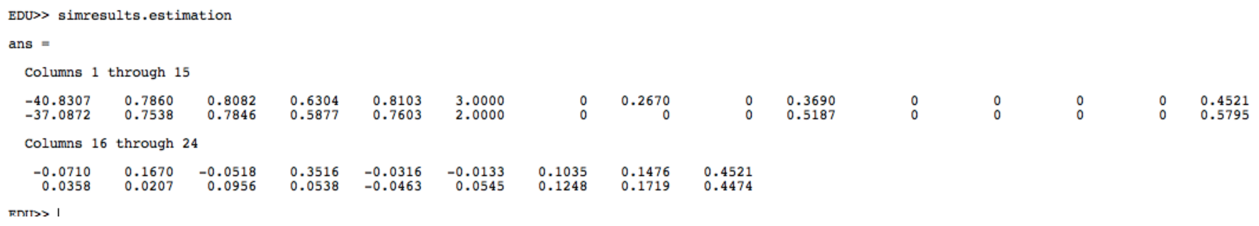
“Tau” is the Kendall’s tau between the observed and predicted values. This is analogous to a multiple r, but provides a measure of monotonic fit, rather than linear fit.

“Pearson r” is the Pearson correlation between the observed and predicted values.

“k” is the number of non-zero predictors in the model, which indexes the complexity of the resulting model. Importantly, this is merely a frequency count of the number of parameters in the final model.

Model Parameters: The second set of numbers provides the beta weights for the selected model. The number of values here depends on the number of predictors (P) in the data set. So, if you used 9 predictors to predict, say income, then there will be a vector of length 9 concatenated to the end of the fit metrics. Some of these beta weights might be zero. In fact, the number of non-zero beta weights in this vector should be equal to **k.**

The figure below provides the output data for simresults.estimation for a data set that included 9 potential predictor variables. The output is for two random samples from a data set that included 133 total participants. Columns 1 – 5 give the various metrics of model fit, with the 6 column indicating how many predictors were ultimately selected by the model. For the first random sample, there are 3 predictors selected. Columns 7 through 15 (7+P-1) provide the weight estimates for the selected predictors. There are 3 predictors included in the best-fit model: predictors 2, 4 and 9.



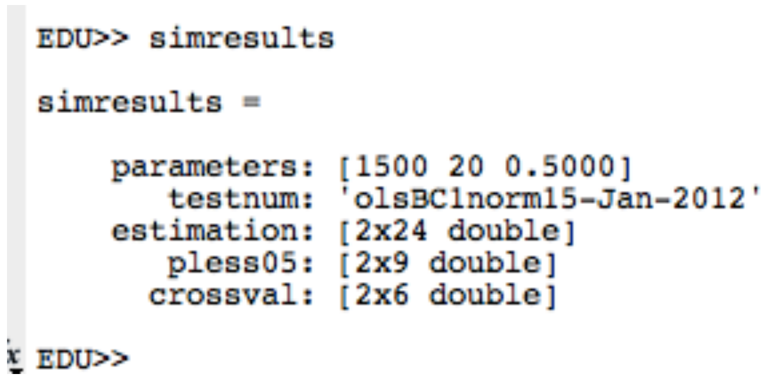
```
EDU>> simresults.estimation

ans =

  Columns 1 through 15

  -40.8307    0.7860    0.8082    0.6304    0.8103    3.0000         0    0.2670         0    0.3690         0         0         0         0    0.4521
  -37.0872    0.7538    0.7846    0.5877    0.7603    2.0000         0         0         0    0.5187         0         0         0         0    0.5795

  Columns 16 through 24

   -0.0710    0.1670   -0.0518    0.3516   -0.0316   -0.0133    0.1035    0.1476    0.4521
    0.0358    0.0207    0.0956    0.0538   -0.0463    0.0545    0.1248    0.1719    0.4474

EDU>> |
```

The columns 16 – 24 show the OLS derived betas for the same data. Note that the betas aren’t equivalent, suggesting that the monotone solution isn’t equivalent to the best linear solution.

Conducting split-half cross-validation

A particularly diagnostic method for evaluating the quality of a statistical modeling solution is to use cross-validation methodology. In cross-validation, one is interested in using the estimated parameter for prediction. That is, instead of focusing on the how well a model 'fits' a specific data set, we are often interested in how well the derived model predicts new observations. We can get some sense of predictive accuracy by using cross-validation procedures.

The GeMM matlab script includes an option for conducting simple cross-validation analyzes on your data by dividing any dataset into two parts. The parameter setting for *"pest"* sets the proportion of the total data set to include for estimation. With pest=0.5, 50% of the data are used to estimate the beta weights, and the remaining 50% are used for cross-validation.



```
EDU>> simresults

simresults = 

    parameters: [1500 20 0.5000]
       testnum: 'olsBC1norm15-Jan-2012'
    estimation: [2x24 double]
       pless05: [2x9 double]
      crossval: [2x6 double]

EDU>>
```

The variable simresults.crossval will provide you with the fit statistics the cross-validation sample for each Monte Carlo run.

[more will be added soon]!