Nonlinear Regression Analysis of FMRI Time Series Data

B. Douglas Ward
Biophysics Research Institute
Medical College of Wisconsin
email: ward@mcw.edu

May 18, 2000

Abstract

The cross-correlation coefficient (1),(2) has been used extensively for the detection of a given "signal" (usually a periodic waveform) in FMRI time series data. This works well when the signal is completely known, or known up to a scaling constant. However, if only the functional form of the signal is known, with the signal itself being a nonlinear function of several unknown parameters, then a different approach is required to detect the presence of the signal buried in noise.

Section 1 describes Program 3dNLfim, which was developed to provide nonlinear regression analysis of *AFNI* 3d+time data sets. The nonlinear regression is accomplished by calculating a least squares fit of the time series data to a user specified model of the data. Program 3dNLfim makes a separate least squares estimate of the model parameters for each voxel in the input time series data set. Program output options include an *AFNI* 'bucket' dataset containing the estimated model parameters, various other parameters related to the signal waveform, and the R^2 and F-statistics for significance of the nonlinear model at each voxel location.

Section 2 describes Program plug_nlfit, an *AFNI* "plug-in", which displays the nonlinear least squares fit of the user specified signal waveform on top of the actual time series data for voxels of interest. Program plug_nlfit is the interactive version of the batch command program 3dNLfim.

Program 3dTSgen, which is decribed in Section 3, provides a means of generating artificial time series data, and storing such data into an *AFNI* 3d+time dataset. The time series data is generated using the operator specified signal and noise models. Such artificial time series data is useful in several ways: 1) Testing of statistical analysis programs for significance of the results. 2) Calculation of the statistical power of a test. 3) Design of experiments.

The above programs have access to a variety of separately compiled signal and noise models, which may be selected by the user. Alternatively, the user may define his own signal model, and add that model to those already accessible by the programs. Section 4 describes how the user may add models to the system.

1 Program 3dNLfim

1.1 Purpose

Program 3dNLfim was developed to provide nonlinear regression analysis of *AFNI* 3d+time data sets. The nonlinear regression is accomplished by calculating a least squares fit of the time series data to a user specified model of the data. The program comes with a selection of separately compiled signal and noise models, which may be chosen by the user. Alternatively, the user may define his own signal model, and add that model to those already accessible by the program (see the section on model definition). Program 3dNLfim makes a separate least squares estimate of the model parameters for each voxel in the input time series data set. Program output options include an *AFNI* 'bucket' dataset containing the estimated model parameters, various other parameters related to the signal waveform, and the R^2 and F-statistics for significance of the nonlinear model at each voxel location.

This program is intended for use in a "batch" processing mode. See program plug_nlfit for description of an interactive version of this program. Programs 3dNLfim and plug_nlfit are complementary in nature; they may be used in combination for model building, data exploration, and data analysis. Also, the associated program 3dTSGen may be useful for experimental design and model validation.

1.2 Theory

The cross-correlation coefficient (1),(2) has been used extensively for the detection of a given "signal" (usually a periodic waveform) in FMRI time series data. This works well when the signal is completely known, or known up to a scaling constant. However, if only the functional form of the signal is known, with the signal itself being a nonlinear function of several unknown parameters, then a different approach is required to detect the presence of the signal buried in noise. Program 3dNLfim was developed to detect an arbitrary signal waveform, where the user specifies the signal as a function of certain parameters. A nonlinear optimization algorithm is used to find the least squares estimate of these parameters. Before describing the nonlinear regression analysis, we first recall the linear regression approach.

1.2.1 Linear Regression Models

A linear regression model is a linear function of its parameters. For example,

$$Y_i = \beta_0 + \beta_1 t_i + \varepsilon_i, \quad i = 1, \ldots, n,$$

is a linear function of the (unknown) parameters β_0 and β_1 (here, ε_i is additive noise). Since we are considering time series data, the independent variable t_i corresponds to time. Of course, this could be generalized to a function of r independent variables X_{i1}, X_{i2}, ..., $X_{i,r-1}$ (and the constant $X_{i0} \equiv 1$) :

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_{r-1} X_{i,r-1} + \varepsilon_i.$$

Using matrix notation

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & X_{11} & \cdots & X_{1,r-1} \\ 1 & X_{21} & \cdots & X_{2,r-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & X_{n1} & \cdots & X_{n,r-1} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{r-1} \end{bmatrix}, \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}.$$

the above equation can be written:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

The linear regression problem is then to find an estimate $\mathbf{b}$ of the vector of unknown parameters

$$\mathbf{b} = \hat{\boldsymbol{\beta}},$$

which provides a good "fit" to the data. The time series data is then estimated by:

$$\hat{\mathbf{Y}} = \mathbf{X}\mathbf{b}$$

The usual criterion for estimating $\mathbf{b}$ is to minimize the error sum of squares:

$$\begin{aligned} SSE &= Q(\mathbf{b}) = \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2 \\ &= \left(\mathbf{Y} - \hat{\mathbf{Y}}\right)^t \left(\mathbf{Y} - \hat{\mathbf{Y}}\right) \end{aligned}$$

It is easy to show that

$$\mathbf{b} = \left(\mathbf{X}^t\mathbf{X}\right)^{-1} \mathbf{X}^t\mathbf{Y}$$

is the least squares estimate of $\boldsymbol{\beta}$. Thus, solving the linear regression problem reduces to simple linear algebra.

1.2.2 Nonlinear Regression Models

A nonlinear regression model is a nonlinear function of its parameters. These models arise frequently. For example, a nonlinear model that is used to represent drug response is given by the difference of two exponentials (3):

$$Y_i = k\left[e^{-\alpha_1(t_i - t_0)} - e^{-\alpha_2(t_i - t_0)}\right].$$

where Y_i is the response at time t_i, and the model parameters k, t_0, α_1, and α_2 are unknown (and so must be estimated). Note that this model applies only for $t_i \geq t_0$. Also, if it is assumed that the above signal is measured in the presence of noise, including a constant offset plus linear trend, then those terms must be added to the model as well. Therefore, the full model would be:

$$Y_i = \gamma_0 + \gamma_1 t_i + k\left[e^{-\alpha_1(t_i - t_0)} - e^{-\alpha_2(t_i - t_0)}\right] u(t_i - t_0) + \varepsilon_i$$

where

Y_i = measured time series data ($i = 1, \ldots, n$);
γ_0 = constant offset term;
γ_1 = coefficient of linear trend;
k = multiplicative constant;
t_0 = time delay;
α_1 = elimination rate constant;
α_2 = absorption rate constant;
ε_i = Gaussian noise, i.i.d. $N(0, \sigma^2)$;
and $u(t)$ is the unit step function:

$$u(t) = \begin{cases} 0, \text{ for } t < 0, \\ 1, \text{ for } t \geq 0. \end{cases}$$

Thus, the full model has six unknown parameters: γ_0, γ_1, k, t_0, α_1, and α_2.

Another drug response model uses the gamma-variate function (4):

$$Y_i = k(t_i - t_0)^r e^{-\frac{t_i - t_0}{b}}$$

or, more completely:

$$Y_i = \gamma_0 + \gamma_1 t_i + k(t_i - t_0)^r e^{-\frac{t_i - t_0}{b}} u(t_i - t_0) + \varepsilon_i$$

with the six unknown parameters: γ_0, γ_1, k, t_0, r, and b.

More generally, we can write a nonlinear time series model as

$$Y_i = f(t_i, \gamma_0, \gamma_1, \ldots, \gamma_{p-1}) + \varepsilon_i$$

where the measurement at time t_i is a nonlinear function of t_i and the p unknown parameters γ_0, γ_1, ..., γ_{p-1}, with additive noise ε_i. As in the linear case, this can also be written using matrix notation:

$$\mathbf{Y} = \mathbf{f}(\mathbf{X}, \boldsymbol{\gamma}) + \boldsymbol{\varepsilon}$$

The objective is to find the least squares estimate of the vector of parameters $\boldsymbol{\gamma}$:

$$\mathbf{g} = \hat{\boldsymbol{\gamma}}$$

and the corresponding time series estimate:

$$\hat{\mathbf{Y}} = \mathbf{f}(\mathbf{X}, \mathbf{g})$$

However, unlike the linear regression case, there is no "closed-form" expression for $\mathbf{g}$ which minimizes the error sum of squares.

1.2.3 Least Squares Estimation

Since there is no analytical expression for the least squares estimate of the model parameters, one alternative is to do a direct search to find the combination of parameters which minimizes the error sum-of-squares.

In general, this is a difficult problem. There are numerous algorithms for performing nonlinear optimization, but each method has its limitations. There is no guarantee that an algorithm will converge to the *global* minimum (i.e., global "least squares") instead of a *local* minimum. Indeed, some algorithms may not converge at all. Also, many algorithms require that the function being minimized be differentiable everywhere; however, this is not true for many cases of practical interest. Some algorithms may be suitable for an interactive mode, where the operator can terminate program execution if the algorithm seems to be diverging. This is obviously not acceptable for analysis of an FMRI data set containing many thousands of voxels (time series).

Moreover, it is desirable to be able to incorporate constraints into the optimization process, i.e., the parameter estimates are restricted to a finite range of values. This is desirable for two reasons: 1) There are usually physical limitations on realistic values for certain parameters (e.g., if a certain parameter represents time delay of a response to an external stimulus, then a negative value would represent a non-causal solution). An "optimal solution" which is not physically realizable has little value. 2) Restricting the parameter space that must be searched helps to reduce the search time, while reducing the likelihood of finding a local rather than a global minimum. However, imposing constraints (essentially, an infinitely high barrier) presents a formidable problem for the search algorithm, particularly those algorithms which rely upon the existence of derivatives of the function being minimized.

The method used (presently) by program 3dNLfim consists of 2 parts: a random search, followed by the nonlinear simplex algorithm (5). The present implementation of the algorithm was adapted from (6). For the random search phase, parameter values are selected at random over the user specified range of constraints for each parameter. A total of $nrand$ (which number may be user specified) random vector parameters are created by the program. For each random vector, the corresponding time series model is generated, and the error sum of squares is obtained (by comparing with the actual time series for that voxel). The $nbest$ best parameter vectors ("best" meaning lowest SSE) are retained, and each is used as the initial starting point for the simplex optimization algorithm.

1.2.4 F-test for Significance of the Nonlinear Regression

Determining whether the time series for a particular voxel corresponds to a given signal waveform can be expressed in terms of a statistical hypothesis test. The null hypothesis is:

$$H_o : \text{time series is ``noise''}$$

and the alternative hypothesis is:

$$H_a : \text{time series is ``signal + noise''} \; .$$

Suppose that "noise" is modeled by a polynomial in t_i of degree $r-1$, plus independent Gaussian random variates. And suppose the "signal" waveform is represented by some function h, which is a function of time and of certain parameters. The observed "signal" is then represented by the sum of the signal and the noise. The hypotheses can then be expressed as:

$$\begin{aligned} H_o &: \; Y_i = \beta_0 + \beta_1 t_i + \cdots + \beta_{r-1} t_i^{r-1} + \varepsilon_i \\ H_a &: \; Y_i = \gamma_{n,0} + \gamma_{n,1} t_i + \cdots + \gamma_{n,r-1} t_i^{r-1} + f(t_i, \gamma_{s,0}, \ldots, \gamma_{s,p-1}) + \varepsilon_i \end{aligned}$$

Here, the alternative hypothesis includes the noise model in addition to the "pure" signal waveform. Note that we explicitly use different symbols for the coefficients of the same terms in the noise model as compared with the signal+noise model, i.e., β_0 and $\gamma_{n,0}$, ..., β_{r-1} and $\gamma_{n,r-1}$. This is to emphasize the fact that it is not (in general) true that $\beta_0 = \gamma_{n,0}$, ..., $\beta_{r-1} = \gamma_{n,r-1}$. A test of the null hypothesis is made by first determining the parameters which yield the least squares fit for the noise model (also called the "reduced" model) and the parameters which yield the least squares fit for the signal plus noise model (also called the "full" model). We have reason to reject the null hypothesis if the error sum of squares from fitting the full model ($SSE(F)$) is much less than the error sum of squares from fitting the reduced model ($SSE(R)$). However, if $SSE(F)$ is only slightly smaller than $SSE(R)$, then we do not have reason to reject the null hypothesis. Consider the test statistic F^{**} :

$$F^{**} = \frac{MS(\text{Regression})}{MS(\text{Error})} = \frac{\frac{SSE(R) - SSE(F)}{df_R - df_F}}{\frac{SSE(F)}{df_F}}$$

where df_R is the number of degrees of freedom for the reduced model, and df_F is the number of degrees of freedom for the full model. Specifically, we have:

$$\begin{aligned} df_R &= n - r, \\ df_F &= n - (r + p), \\ \text{so } df_R - df_F &= p. \end{aligned}$$

By the above reasoning, we see that a large value for F^{**} indicates that signal is present, whereas a small value for F^{**} suggests that only noise is present. For large sample size n, the statistic F^{**} has an approximate $F(df_R - df_F, df_F)$ distribution under the null hypothesis (7).

Program 3dNLfim calculates the F^{**} statistic for each voxel, and appends these values as one of the sub-bricks of an *AFNI* "bucket" dataset (if so requested by the user).

1.2.5 Coefficient of Multiple Determination

The coefficient of multiple determination, R^2, can be used as an indicator for how well the full model fits the data. We define R^2:

$$R^2 \equiv 1 - \frac{SSE(F)}{SSE(R)}$$

Roughly speaking, R^2 is the proportion of the variation in the data (about the baseline) that is explained by the full model. Note that, for every voxel, $0 \le R^2 \le 1$. (R^2 is a generalization of the square of the correlation coefficient computed in the fim programs).

Program 3dNLfim calculates R^2 for each voxel, and appends these values as one of the sub-bricks of an *AFNI* "bucket" dataset (if so requested by the user).

1.2.6 t-test for Significance of Individual Parameters

When developing linear regression models, it is useful to know the significance of the individual parameters that constitute the model. This may help identify terms in the model that may be safely discarded in order to simplify the model. Therefore, linear regression programs often provide the t-statistics for the individual parameters in the model.

For the linear regression model, the variance-covariance matrix for the regression coefficients is given by:

$$\mathbf{s}^2(\mathbf{b}) = MSE \cdot \left(\mathbf{X}^t\mathbf{X}\right)^{-1}$$

For the nonlinear regression case, we do not have a linear relation between the observation vector $\mathbf{Y} = \mathbf{f}(\mathbf{X}, \boldsymbol{\gamma}) + \boldsymbol{\varepsilon}$ and the vector of parameters $\boldsymbol{\gamma}$. However, we can make a linear approximation in the neighborhood of the vector parameter estimate $\mathbf{g} = \hat{\boldsymbol{\gamma}}$:

$$\mathbf{s}^2(\mathbf{g}) = MSE \cdot (\mathbf{D}^t\mathbf{D})^{-1}$$

where $\mathbf{D}$ is the matrix of partial derivatives evaluated at $\boldsymbol{\gamma} = \mathbf{g}$:

$$D_{ik} = \left[\frac{\partial f(t_i, \boldsymbol{\gamma})}{\partial \gamma_k}\right]_{\boldsymbol{\gamma}=\mathbf{g}}$$

These partial derivatives can be estimated numerically by:

$$D_{ik} \approx \frac{y(t_i, \gamma_0, \ldots, \gamma_k + \delta, \ldots, \gamma_{r+p-1}) - y(t_i, \gamma_0, \ldots, \gamma_k, \ldots, \gamma_{r+p-1})}{\delta}$$

Then, for large sample size n, the statistic t^{**} :

$$t^{**}[k] = \frac{g[k]}{s(g[k])}$$

has an approximate $t(n-p)$ distribution under the null hypothesis (7).

Program 3dNLfim calculates the t^{**} statistic for each voxel, and appends these values as sub-bricks of an *AFNI* "bucket" dataset (if so requested by the user).

1.2.7 Summary

The overall procedure is summarized below.

1. Calculate linear regression fit of the *reduced model*:

$$\mathbf{b} = \left(\mathbf{X}^t\mathbf{X}\right)^{-1}\mathbf{X}^t\mathbf{Y}$$

$$\hat{\mathbf{Y}} = \mathbf{X}\mathbf{b}$$

2. Calculate error sum of squares for the *reduced model*:

$$SSE(R) = Q(\mathbf{b}) = \sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

3. If $SSE(R)$ is very small, then stop. There is no need to fit the full model.

4. Generate random parameter vectors in the (constrained) parameter space Ω:

$$\mathbf{g}_i \in \Omega,\ i = 1, \ldots, nrand.$$

5. Evaluate each of these random parameter vectors, and keep the best (lowest SSE) of these parameter vectors: $\mathbf{g}_j$, $j = 1, \ldots, nbest$.

6. For $j = 1, \ldots, nbest$ do:

   A. Use $\mathbf{g}_j$ as the initial guess for the parameter vector.

   B. Perform nonlinear optimization:

      a. Obtain new estimate $\mathbf{g}'_j$.
      b. If $\mathbf{g}'_j$ violates constraints, set $SSE = \infty$.
      c. Otherwise, calculate $SSE = Q(\mathbf{g}'_j)$.
      d. Repeat step (B) until vector parameter estimate converges.

   C. Save best parameter estimate $\mathbf{g}$ and $SSE(F) = Q(\mathbf{g})$. This is the least squares estimate of the *full model.*

7. Calculate statistics for the nonlinear regression:

$$R^2 = 1 - \frac{SSE(F)}{SSE(R)}$$

$$F^{**} = \frac{MS(\text{Regression})}{MS(\text{Error})} = \frac{\dfrac{SSE(R) - SSE(F)}{df_R - df_F}}{\dfrac{SSE(F)}{df_F}}$$

$$t^{**}[k] = \frac{g[k]}{s(g[k])}$$

The above procedure is repeated for each voxel in the data set.

1.3 Usage

The syntax for execution of program 3dNLfim is as follows:

3dNLfim -input fname [-mask mset] -noise nlabel -signal slabel
[-ignore num] [-inTR] [-time fname]
[-nconstr 0 $\mathbf{a}_0$ $\mathbf{b}_0$] $\cdots$ **[-nconstr r-1 $\mathbf{a}_{r-1}$ $\mathbf{b}_{r-1}$]**
[-sconstr 0 $\mathbf{c}_0$ $\mathbf{d}_0$] $\cdots$ **[-sconstr p-1 $\mathbf{c}_{p-1}$ $\mathbf{d}_{p-1}$]**
[-nabs] [-nrand n] [-nbest b] [-rmsmin r] [-fdisp fval]
[-freg prefix] [-frsqr prefix] [-ftmax prefix] [-fsmax prefix]
[-fpsmax prefix] [-farea prefix] [-fparea prefix]
[-tncoef k prefix] [-tscoef k prefix] [-fncoef k prefix] [-fscoef k prefix]
[-bucket n prefix] [-brick $\mathbf{m}_1$ options] ... **[-brick $\mathbf{m}_n$ options]**
[-sfit prefix] [-snfit prefix]

The different command line options are explained below.

1.4 Option

-input *fname*

The mandatory -input command specifies that *fname* is the filename of the *AFNI* 3d + time data set to be used as input for the program.

-mask *mset*

The optional -mask command specifies that the 0 sub-brick of dataset *mset* is to be used as a mask (a sub-brick selector is allowed). This mask indicates which voxels are to be analyzed. That is, voxels having value 0 in the *mset* dataset are ignored. The default is to use all voxels.

-noise *nlabel*

The mandatory -noise command tells program 3dNLfim which noise model to use. During initialization, program 3dNLfim attempts to locate all noise models that have been compiled and stored on disk. Each noise model has an associated name or label. If program 3dNLfim finds *nlabel* among the noise model names, then that noise model will be used in the analysis. Otherwise, the program prints an error message and halts.

-signal *slabel*

The mandatory -signal command tells program 3dNLfim which signal model to use. During initialization, program 3dNLfim attempts to locate all signal models that have been compiled and stored on disk. Each signal model has an associated name or label. If program 3dNLfim finds *slabel* among the signal model names, then that signal model will be used in the analysis. Otherwise, the program prints an error message and halts.

-ignore *num*

The optional -ignore command specifies that the first *num* time series images are *not* to be used in the regression analysis. The default value is $num = 3$.

-inTR

The model functions are calculated on a time grid of: 0, *delt*, 2**delt*, 3**delt*, ...The optional -inTR command specifies that $delt$ = TR of the input 3d+time dataset. The default value is $delt = 1$ (i.e., the default time sequence is: 0, 1, ..., $n-1$).

-time *fname*

The optional -time command specifies that *fname* is the filename of the *ASCII* file containing the sequence of time coordinates for each volume in the time series. The default time sequence is: 0, 1, ..., $n-1$.

-scnstr k c_k d_k

The optional -scnstr command specifies the constraints on the signal model parameter values. Note that the constraints for the signal parameters are *absolute*, i.e., for the kth signal parameter, we require:

$$c_k <= g_s[k] <= d_k.$$

The default values for the signal parameter constraints are specified in the file which contains the signal model. The values for c_k and d_k must satisfy: $c_k <= d_k$.

-ncnstr k c_k d_k

The optional -ncnstr command specifies the constraints on the noise model parameter values. Note that the constraints for the noise parameters are *relative* to the linear regression estimates for these same parameters (but see option -nabs below), i.e., for the kth noise parameter, we require:

$$c_k + b[k] <= g_n[k] <= d_k + b[k],$$

where $b[k]$ is the linear regression coefficient for the kth noise parameter.

The default values for the noise parameter constraints are specified in the file which contains the noise model. The values for c_k and d_k must satisfy: $c_k <= d_k$.

-nabs

The optional -nabs command specifies that constraints for all noise parameters are *absolute*, i.e., for the kth noise parameter, we require:

$$c_k <= g_n[k] <= d_k; \quad 0 \leq k < r.$$

The default is that all noise constraints are *relative*.

-nrand n

The optional -nrand command specifies that n is the number of random test points (from the constrained parameter space) to be evaluated (prior to nonlinear optimization). The default value is $n = 100$.

-nbest *b*

The optional -nbest command indicates that the b best of the n random test points is to be used as the starting point (parameter vector) for nonlinear optimization. Of course, it is required that $b \leq n$. The default value is $b = 5$.

-rmsmin *r*

The optional -rmsmin command is used to set the minimum rms error r required in order to reject the reduced (noise) model. In other words, the full model will *not* be calculated for those voxels whose time series, when fitted with the reduced model, has error rms $< r$. This is used primarily to speed program execution by screening out voxels which lie outside the brain. The user should choose a value for r which is much smaller than the natural measurement error. The default value is $r = 0$.

-fdisp *f*

The optional -fdisp command is used to control output to the user's terminal during program execution. For each voxel in the data set, if the estimated F^{**} is greater than or equal to f, then the estimated noise and signal model parameters are written to the screen; otherwise, nothing is written to the screen for that particular voxel. Note that the -fdisp command effects screen output only, and has absolutely no effect upon the data file output generated by the program.

-freg *prefix*

The optional -freg command tells program 3dNLfim to calculate an F-test for significance of the nonlinear regression for each voxel. The output is written to the file with the user specified *prefix* filename. This output consists of a 2 sub-brick *AFNI* dataset of type "fift". The first sub-brick consists of the square root of the mean sum of squares due to error ($\sqrt{MS(\text{Error})}$) for each voxel, and the second sub-brick contains the corresponding F-statistics F^{**}. The F-statistics are calculated by dividing the mean sum of squares due to regression by the mean sum of squares due to error.

$$\textit{AFNI}\ \text{“fift” dataset} \quad \left\{ \begin{array}{l} \boxed{I = \sqrt{MS(\text{Error})}} \\ \\ \boxed{F^{**} = \dfrac{MS(\text{Reg})}{MS(\text{Error})} = \dfrac{\dfrac{SSE(R) - SSE(F)}{df_R - df_F}}{\dfrac{SSE(F)}{df_F}}} \end{array} \right.$$

When this data file is used as input to program *AFNI*, the second sub-brick can be used as a "threshold" for determining which voxels "light-up"; the intensity is then determined by the first sub-brick. The ".HEAD" file for this dataset tells *AFNI* that the second sub-brick contains F-statistics; also, the ".HEAD" file contains the numbers for the numerator degrees of freedom ($df_R - df_F = (n - r) - (n - (p + r)) = p$) and the denominator degrees of freedom ($df_F = n - (p + r)$). So, by setting the appropriate threshold, only those voxels having the user-specified p-values for significance of the nonlinear regression will "light-up".

-frsqr *prefix*
The optional -frsqr command tells program 3dNLfim to calculate the coefficient of multiple determination R^2. The output, consisting of a 2 sub-brick *AFNI* dataset of type "fift", is written to the file with the user specified *prefix* filename. The first sub-brick consists of the estimated R^2 for each voxel, and the second sub-brick contains the corresponding F-statistics F^{**}:

$$\textit{AFNI}\text{ ``fift'' dataset} \quad \left\{ \begin{array}{l} \boxed{R^2 = 1 - \dfrac{SSE(F)}{SSE(R)}} \\ \\ \boxed{F^{**} = \dfrac{MS(\text{Reg})}{MS(\text{Error})}} \end{array} \right.$$

-ftmax *prefix*
The optional -ftmax command tells program 3dNLfim to calculate the time at which the maximum absolute value of the estimated signal occurs. Suppose that the signal + noise model is given by:

$$Y_i = \gamma_{n,0} + \gamma_{n,1}t_i + \cdots + \gamma_{n,r-1}t_i^{r-1} + f(t_i, \gamma_{s,0}, \ldots, \gamma_{s,p-1}) + \varepsilon_i$$

The signal alone is estimated by:

$$s_i = h(t_i, g_{s,0}, \ldots, g_{s,p-1}), \text{ for } i = 1, \ldots, N,$$

where $g_{s,0}$, ..., $g_{s,p-1}$ are the nonlinear least squares estimates of $\gamma_{s,0}, \ldots, \gamma_{s,p-1}$.Then the time of the maximum absolute value is defined by:

$$\begin{aligned} T_{\max} &= j, \\ \text{where } |s_j| &= \max_{i=1,\ldots,N} |s_i| \end{aligned}$$

Thus, $T_{\max}$ is the time at which the signal is greatest in magnitude. The output, consisting of a 2 sub-brick *AFNI* dataset of type "fift", is written to the file with the user specified *prefix* filename. The first sub-brick consists of the estimated $T_{\max}$ for each voxel, and the second sub-brick contains the corresponding F-statistics F^{**}:

$$\textit{AFNI}\text{ ``fift'' dataset} \quad \left\{ \begin{array}{l} \boxed{T_{\max}} \\ \\ \boxed{F^{**} = \dfrac{MS(\text{Reg})}{MS(\text{Error})}} \end{array} \right.$$

-fsmax *prefix*
The optional -fsmax command tells program 3dNLfim to calculate the signed maximum absolute value of the estimated signal. As above, suppose the signal alone is estimated by:

$$s_i = f(t_i, g_{s,0}, \ldots, g_{s,p-1}), \text{ for } i = 1, \ldots, N.$$

Then the signed maximum absolute value is defined by:

$$\begin{aligned} S_{\max} &= s_j, \\ \text{where } |s_j| &= \max_{i=1,\ldots,N} |s_i| \end{aligned}$$

Roughly speaking, $S_{\max}$ is the "magnitude" of the signal. The output, consisting of a 2 sub-brick *AFNI* dataset of type "fift", is written to the file with the user specified *prefix* filename. The first sub-brick consists of the estimated $S_{\max}$ for each voxel, and the second sub-brick contains the corresponding F-statistics F^{**}:

$$\textit{AFNI} \text{ ``fift'' dataset} \quad \left\{ \begin{array}{l} \boxed{S_{\max}} \\ \\ \boxed{F^{**} = \dfrac{MS(\text{Reg})}{MS(\text{Error})}} \end{array} \right.$$

-fpsmax *prefixname*

The optional -fpsmax command tells program 3dNLfim to calculate the signed maximum absolute value of the estimated signal as a percentage of the baseline. Suppose that the signal + noise model is given by:

$$Y_i = \gamma_{n,0} + \gamma_{n,1} t_i + \cdots + \gamma_{n,r-1} t_i^{r-1} + f(t_i, \gamma_{s,0}, \ldots, \gamma_{s,p-1}) + \varepsilon_i$$

The signal alone is estimated by:

$$s_i = f(t_i, g_{s,0}, \ldots, g_{s,p-1}), \text{ for } i = 1, \ldots, N,$$

and the baseline is estimated by:

$$n_i = g_{n,0} + g_{n,1} t_i + \cdots + g_{n,r-1} t_i^{r-1}, \text{ for } i = 1, \ldots, N,$$

where $g_{s,0}$, …, $g_{s,p-1}$ are the nonlinear least squares estimates of $\gamma_{s,0}, \ldots, \gamma_{s,p-1}$, and $g_{n,0}$, …, $g_{n,r-1}$ are the nonlinear least squares estimates of $\gamma_{n,0}, \ldots, \gamma_{n,r-1}$. Then the percentage signed maximum absolute value is defined by:

$$\begin{aligned} PS_{\max} &= 100 \times \frac{s_j}{n_j}, \\ \text{where } j \text{ is s.t. } |s_j| &= \max_{i=1,\ldots,N} |s_i| \end{aligned}$$

The output, consisting of a 2 sub-brick *AFNI* dataset of type "fift", is written to the file with the user specified *prefix* filename. The first sub-brick consists of the estimated $PS_{\max}$ for each voxel, and the second sub-brick contains the corresponding F-statistics F^{**}:

$$\textit{AFNI} \text{ ``fift'' dataset} \quad \left\{ \begin{array}{l} \boxed{PS_{\max}} \\ \\ \boxed{F^{**} = \dfrac{MS(\text{Reg})}{MS(\text{Error})}} \end{array} \right.$$

-farea *prefix*

The optional -farea command tells program 3dNLfim to calculate the area under the signal. Suppose that the signal + noise model is given by:

$$Y_i = \gamma_{n,0} + \gamma_{n,1} t_i + \cdots + \gamma_{n,r-1} t_i^{r-1} + f(t_i, \gamma_{s,0}, \ldots, \gamma_{s,p-1}) + \varepsilon_i$$

The signal alone is estimated by:

$$s_i = f(t_i, g_{s,0}, \ldots, g_{s,p-1}), \text{ for } i = 1, \ldots, N,$$

where $g_{s,0}$, ..., $g_{s,p-1}$ are the nonlinear least squares estimates of $\gamma_{s,0}$, ..., $\gamma_{s,p-1}$. Then the area under the signal is defined by:

$$S_{area} = \int_1^N |s(t)|\, dt$$

where the integral is numerically approximated using trapezoidal integration. Note that the calculated area is always positive. The output, consisting of a 2 sub-brick *AFNI* dataset of type "fift", is written to the file with the user specified *prefix* filename. The first sub-brick consists of the estimated S_{area} for each voxel, and the second sub-brick contains the corresponding F-statistics F^{**}:

$$\textit{AFNI} \text{ ``fift'' dataset} \quad \left\{ \begin{array}{l} \boxed{S_{area}} \\ \\ \boxed{F^{**} = \dfrac{MS(\text{Reg})}{MS(\text{Error})}} \end{array} \right.$$

-fparea *prefix*

The optional -fparea command tells program 3dNLfim to calculate the signed area under the signal as a percentage of the baseline area. Suppose that the signal + noise model is given by:

$$Y_i = \gamma_{n,0} + \gamma_{n,1} t_i + \cdots + \gamma_{n,r-1} t_i^{r-1} + f(t_i, \gamma_{s,0}, \ldots, \gamma_{s,p-1}) + \varepsilon_i$$

The signal alone is estimated by:

$$s_i = f(t_i, g_{s,0}, \ldots, g_{s,p-1}), \text{ for } i = 1, \ldots, N,$$

and the baseline is estimated by:

$$n_i = g_{n,0} + g_{n,1} t_i + \cdots + g_{n,r-1} t_i^{r-1}, \text{ for } i = 1, \ldots, N,$$

where $g_{s,0}$, ..., $g_{s,p-1}$ are the nonlinear least squares estimates of $\gamma_{s,0}$, ..., $\gamma_{s,p-1}$, and $g_{n,0}$, ..., $g_{n,r-1}$ are the nonlinear least squares estimates of $\gamma_{n,0}$, ..., $\gamma_{n,r-1}$. The signed area under the signal is defined by:

$$S_{area} = \int_1^N s(t) dt$$

and the baseline area is defined by:

$$B_{area} = \int_1^N |n(t)|\, dt$$

where the integral is numerically approximated using trapezoidal integration. Note that S_{area} may be either positive or negative. Now the signed area under the signal as a percentage of the baseline area is defined by:

$$PS_{area} = 100 \times \frac{S_{area}}{B_{area}},$$

The output, consisting of a 2 sub-brick *AFNI* dataset of type "fift", is written to the file with the user specified *prefix* filename. The first sub-brick consists of the estimated PS_{area} for each voxel, and the second sub-brick contains the corresponding F-statistics F^{**}:

$$\textit{AFNI}\text{ “fift” dataset} \quad \left\{ \begin{array}{l} \boxed{PS_{area}} \\ \\ \boxed{F^{**} = \dfrac{MS(\text{Reg})}{MS(\text{Error})}} \end{array} \right.$$

-tncoef *k prefix*

The optional -tncoef command is used to save the estimated value of the kth noise parameter. Program 3dNLfim calculates the least squares estimate of the kth noise parameter and the corresponding t-statistic of the estimate for each voxel. The result is stored as a 2 sub-brick *AFNI* data set of type "fitt" in the file with the user specified *prefix* filename. The first sub-brick consists of the noise parameter estimate $g_n[k]$, and the second sub-brick contains the corresponding t-statistic t^{**} for each voxel.

$$\textit{AFNI}\text{ “fitt” dataset} \quad \left\{ \begin{array}{l} \boxed{g_n[k] = \text{non-linear L.S. est. of } \gamma_n[k]} \\ \\ \boxed{t^{**} = \dfrac{g_n[k]}{s(g_n[k])}} \end{array} \right.$$

When this is used as an input file to program *AFNI*, the second sub-brick can be used to set the threshold for determining which voxels have an estimated parameter value which is significantly different from zero. The ".HEAD" file informs *AFNI* that the second sub-brick contains t-statistics, and that $df = n - (r + p)$ should be used for the degrees of freedom.

It is the author's experience that the t^{**} statistic can be misleading, and its significance difficult to interpret, in the context of a nonlinear model. Therefore, unless the "nonlinear" model is actually a "linear" model, use of the t^{**} statistic is *not* recommended. A better choice for display of noise parameters is the -fncoef command discussed below.

-tscoef ***k prefix***

The optional -tscoef command is used to save the estimated value of the kth signal parameter. Program 3dNLfim then calculates the least squares estimate of the kth signal parameter and the corresponding t-statistic of the estimate for each voxel. The result is stored as a 2 sub-brick *AFNI* data set of type "fitt" in the file with the user specified *prefix* filename. The first sub-brick consists of the signal parameter estimate $g_s[k]$, and the second sub-brick contains the corresponding t-statistic t^{**} for each voxel.

$$\textit{AFNI}\ \text{“fitt” dataset} \quad \begin{cases} \boxed{g_s[k] = \text{non-linear L.S. est. of } \gamma_s[k]} \\[2ex] \boxed{t^{**} = \dfrac{g_s[k]}{s(g_s[k])}} \end{cases}$$

When this is used as an input file to program *AFNI*, the second sub-brick can be used to set the threshold for determining which voxels have an estimated parameter value which is significantly different from zero. The ".HEAD" file informs *AFNI* that the second sub-brick contains t-statistics, and that $df = n - (p + r)$ should be used for the degrees of freedom.

It is the author's experience that the t^{**} statistic can be misleading, and its significance difficult to interpret, in the context of a nonlinear model. Therefore, unless the "nonlinear" model is actually a "linear" model, use of the t^{**} statistic is *not* recommended. A better choice for display of signal parameters is the -fscoef command discussed below.

-fncoef k ***prefix***

The optional -fncoef command tells program 3dNLfim to save the non-linear least squares estimated value of the kth noise parameter in an *AFNI* "fift" data set, along with the F-statistics for significance of the regression. The output is then written to the file with the user specified *prefix* filename. This output consists of a 2 sub-brick *AFNI* dataset of type "fift". The first sub-brick consists of the noise parameter estimate $g_n[k]$, and the second sub-brick contains the corresponding F-statistics F^{**}. It must be emphasized that the F-statistics pertain to significance of the overall regression, and do not indicate the significance of the individual parameter.

$$\textit{AFNI}\ \text{“fift” dataset} \quad \begin{cases} \boxed{g_n[k] = \text{non-linear L.S. est. of } \gamma_n[k]} \\[2ex] \boxed{F^{**} = \dfrac{MS(\text{Reg})}{MS(\text{Error})}} \end{cases}$$

When this data file is used as input to program *AFNI*, the second sub-brick can be used as a "threshold" for determining which voxels "light-up"; the intensity is then determined by the first sub-brick. So, by setting the appropriate threshold, only those voxels having the user-specified p-values for significance of the nonlinear regression will "light-up". The color coding of the voxels which light-up indicates the sign and magnitude of the estimated noise parameters.

-fscoef k *prefix*

The optional -fscoef command tells program 3dNLfim to save the non-linear least squares estimated value of the kth signal parameter in an *AFNI* "fift" data set, along with the F-statistics for significance of the regression. The output is then written to the file with the user specified *prefix* filename. This output consists of a 2 sub-brick *AFNI* dataset of type "fift". The first sub-brick consists of the signal parameter estimate $g_s[k]$, and the second sub-brick contains the corresponding F-statistics F^{**}. It must be emphasized that the F-statistics pertain to significance of the overall regression, and do not indicate the significance of the individual parameter.

$$\textit{AFNI}\ \text{"fift" dataset} \quad \left\{ \begin{array}{l} \boxed{g_s[k] = \text{non-linear L.S. est. of } \gamma_s[k]} \\ \\ \boxed{F^{**} = \dfrac{MS(\text{Reg})}{MS(\text{Error})}} \end{array} \right.$$

When this data file is used as input to program *AFNI*, the second sub-brick can be used as a "threshold" for determining which voxels "light-up"; the intensity is then determined by the first sub-brick. So, by setting the appropriate threshold, only those voxels having the user-specified p-values for significance of the nonlinear regression will "light-up". The color coding of the voxels which light-up indicates the sign and magnitude of the estimated signal parameters.

-bucket n *prefix*

The -bucket command is used to create a single *AFNI* "bucket" type dataset having n sub-bricks. The output is written to the file with the user specified *prefix* filename. Each of the individual sub-bricks can then be accessed for display within program afni. The purpose of this command is to simplify file management, since all of the output results for a particular problem can now be contained within a single *AFNI* bucket dataset.

If $n = 0$, then the *default* output bucket dataset is created. (See Example 2.) The labels for the individual signal and noise parameters come from the model definition source code. The default dataset has $p + r + 8$ sub-bricks, as illustrated below.

If $n > 0$, then the contents and labels for the individual sub-bricks within the bucket dataset are specified by the user, by means of the -brick command described below. Note that the -bucket command *must* precede the -brick commands.

Structure of default bucket dataset:

Brick	Label	Contents
#0	(Label for noise par. #0)	$g_n[0]$ = non-linear L.S. est. of $\gamma_n[0]$
$\vdots$	$\vdots$	$\vdots$
#r-1	(Label for noise par. #r-1)	$g_n[r-1]$ = non-linear L.S. est. of $\gamma_n[r-1]$
#r	(Label for signal par. #0)	$g_s[0]$ = non-linear L.S. est. of $\gamma_s[0]$
$\vdots$	$\vdots$	$\vdots$
#p+r-1	(Label for signal par. #p-1)	$g_s[p-1]$ = non-linear L.S. est. of $\gamma_s[p-1]$
#p+r	Signal TMax	Time of signed maximum of signal
#p+r+1	Signal SMax	Signed maximum of signal
#p+r+2	Signal % SMax	Signed maximum of signal as a % of baseline
#p+r+3	Signal Area	Area under signal (always positive)
#p+r+4	Signal % Area	Signed area under signal as a % of baseline
#p+r+5	Sigma Resid	$\sigma = \sqrt{MS(\text{Error})}$
#p+r+6	R^2	$R^2 = 1 - \frac{SSE(F)}{SSE(R)}$
#p+r+7	F-stat Regression	$F^{**} = \frac{MS(\text{Reg})}{MS(\text{Error})}$

-brick m ***options***

The -brick command is used to specify the contents and labels for the mth sub-brick ($0 \leq m < n$) within the bucket dataset. There must be one -brick command for each of the n sub-bricks in the dataset (where n has been previously specified by the -bucket command). (See Example 3).

There are 12 versions of the -brick command:

-brick m ncoef k label	The mth sub-brick is to contain the non-linear L.S. est. of the kth parameter in the noise model
-brick m scoef k label	The mth sub-brick is to contain the non-linear L.S. est. of the kth parameter in the signal model.
-brick m tmax label	The mth sub-brick is to contain the time of the maximum absolute value of the signal.
-brick m smax label	The mth sub-brick is to contain the signed maximum value of the signal.
-brick m psmax label	The mth sub-brick is to contain the signed maximum value of the signal as a percentage relative to the baseline.
-brick m area label	The mth sub-brick is to contain the area between the signal and the baseline. Note: Area is always positive.
-brick m parea label	The mth sub-brick is to contain the signed area between the signal and the baseline as a percentage of the baseline area.

-brick m tncoef k label	The mth sub-brick is to contain the t-statistic for the kth parameter in the noise model.
-brick m tscoef k label	The mth sub-brick is to contain the t-statistic for the kth parameter in the signal model.
-brick m resid label	The mth sub-brick is to contain standard deviation of the error from fitting the full model.
-brick m rsqr label	The mth sub-brick is to contain the coefficient of multiple determination R^2.
-brick m fstat label	The mth sub-brick is to contain the F-statistic for significance of the non-linear regression.

In each case, the label for the sub-brick is specified by *label*.

The following commands write the time series fit for each voxel to an *AFNI* 3d+time dataset:

-sfit *prefix*

The optional -sfit command tells program 3dNLfim to write the signal model fit

$$\widehat{s}_i = f(t_i, g_{s,0}, \ldots, g_{s,p-1})$$

to the *AFNI* 3d+time dataset file with the user specified *prefix* filename.

-snfit *prefix*

The optional -snfit command tells program 3dNLfim to write the signal+noise model fit

$$\widehat{Y}_i = g_{n,0} + g_{n,1}t_i + \cdots + g_{n,r-1}t_i^{r-1} + f(t_i, g_{s,0}, \ldots, g_{s,p-1})$$

to the *AFNI* 3d+time dataset file with the user specified *prefix* filename.

1.5 Notes: Future Improvements

- Nonlinear Optimization Algorithm

  The nonlinear optimization algorithm used here (the “simplex” algorithm) is not very efficient. This algorithm requires a large number of function evaluations, and so consumes a great deal of time in searching for the least squares estimate of the model parameters. Other nonlinear optimization algorithms are available, and, if implemented, may decrease the time required for program execution.

- Additional Independent Variables

  In addition to the constant and linear trend, program 3dfim allows the operator to specify ‘ort’ time series files, to which the data is orthogonalized. Similarly, program 3dNLfim could be extended to incorporate additional independent variables in both the linear and the nonlinear models. For example, if X_i is a time series, then the above hypothesis test could be modified to include X_i as another independent variable:

$$\begin{aligned} H_o &: \; Y_i = \beta_0 + \beta_1 t_i + \beta_2 X_i + \varepsilon_i \\ H_a &: \; Y_i = \gamma_{n,0} + \gamma_{n,1} t_i + \gamma_{n,2} X_i + f(t_i, X_i, \gamma_{s,0}, \ldots, \gamma_{s,p-1}) + \varepsilon_i \end{aligned}$$

1.6 Examples

Example 1. Drug Response (Differential Exponential Model)

A researcher is using FMRI to study neural response to an injected drug. It is decided to use the difference-of-exponentials function to model the drug response as a function of time. Program 3dNLfim can be executed with the following batch commands:

Batch Command File for Example 1

```
3dNLfim  \
-input fred+orig \
-ignore 3  \
-noise Linear \
-signal DiffExp \
-nconstr 0 -100.0 100.0 \
-nconstr 1 -1.0 1.0  \
-sconstr 0 45.0 75.0 \
-sconstr 1 -500.0 500.0  \
-sconstr 2 0.00 0.15  \
-sconstr 3 0.15 0.50  \
-nrand 500  \
-nbest 5  \
-rmsmin 1.0  \
-fdisp 100.0  \
-fsmax fred.smax \
-ftmax fred.tmax \
-fncoef 0 fred.const  \
-fncoef 1 fred.linear  \
-fscoef 0 fred.t0  \
-fscoef 1 fred.k  \
-fscoef 2 fred.alpha1  \
-fscoef 3 fred.alpha2
```

■

The first batch command specifies that the input 3d+time dataset is to be read from file fred+orig (.HEAD and .BRIK). The command -ignore specifies that the 4th image is the first to be used in the analysis, i.e., the first three data points in each time series are to be discarded. The -noise command specifies that "Linear" noise model is to be used (i.e., the noise model includes a linear trend). The -signal command is used to specify signal model "DiffExp" (i.e., the differential - exponential drug response model). The full (signal + noise) model therefore has the form:

$$Y_i = g_n[0] + g_n[1] * t_i + g_s[1] * (\exp(-g_s[2] * (t_i - g_s[0])) - \exp(-g_s[3] * (t_i - g_s[0])))$$

for $t_i \geq g_s[0]$, and

$$Y_i = g_n[0] + g_n[1] * t_i$$

for $t_i < g_s[0]$.

The -nconstr and -sconstr commands are used to specify the noise and signal parameter constraints:

$$\begin{array}{ccccc}
-100.0 + b[0] & \leq & g_n[0] & \leq & 100.0 + b[0] \\
-1.0 + b[1] & \leq & g_n[1] & \leq & 1.0 + b[1] \\
45.0 & \leq & g_s[0] & \leq & 75.0 \\
-500.0 & \leq & g_s[1] & \leq & 500.0 \\
0.00 & \leq & g_s[2] & \leq & 0.15 \\
0.15 & \leq & g_s[3] & \leq & 0.50
\end{array}$$

where $b[0]$ and $b[1]$ are the linear regression estimates for the noise parameters.

The command -nrand indicates that 500 random parameter vectors are to be generated, and the command -nbest specifies that the 5 best of these random vectors are to be used as the initial values for the nonlinear optimization. Command -rmsmin indicates that the full model should be calculated only if the rms error from fitting the reduced model exceeds 1.0. The -fdisp command indicates that the reduced model and full model parameter estimates should be written to the screen only for those voxels whose F-statistic is equal to or greater than 100.

The remaining commands specify that 8 output *AFNI* "fift" data sets are to be created. Each of these commands writes the F-statistics for significance of the nonlinear regression to the second sub-brick of the *AFNI* "fift" dataset. The -fsmax and -ftmax commands write the signed maximum absolute value of the estimated signal, and the time of the signed maximum, to files fred.smax and fred.tmax, respectively. The two -fncoef commands write the estimates of the constant and linear trend noise parameters to files fred.const and fred.linear, respectively. Finally, the four -fscoef commands write estimates of signal parameters t_0, k, α_1, and α_2, to files fred.t0, fred.k, fred.alpha1, and fred.alpha2, respectively.

During execution of the program, screen output is generated for voxels whose F-statistic exceeds 100, such as:

Screen Output for Example 1

Program: 3dNLfim
Author: B. Douglas Ward
Date: 10 May 2000

Voxel #413

Reduced (Linear) Model:
b[0] = 766.668518 constant
b[1] = 0.380208 linear

Full (Linear + DiffExp) Model:

gn[0] =	739.128052	constant
gn[1] =	0.172320	linear
gs[0] =	54.229733	t0
gs[1] =	141.777496	k
gs[2] =	0.010696	alpha1
gs[3] =	0.153561	alpha2

Signal Tmax	=	73.000
Signal Smax	=	108.049
Signal PSmax	=	14.374
Signal Area	=	9422.469
Signal PArea	=	6.359

RMSE Rdcd	=	38.342
RMSE Full	=	16.360

R^2	=	0.822
F[4,191]	=	220.002
p-value	=	2.475459e-70

⋮

Voxel #476

etc.

■

After program 3dNLfim has finished execution, program afni can be used to view the 8 datasets (16 output files).

Example 2. Drug Response (continued)

In order to reduce the number of output files and simplify file management, the above example was repeated using the -bucket command.

Batch Command File for Example 2

```
3dNLfim  \
-input fred+orig \
-ignore 3  \
-noise Linear \
-signal DiffExp \
-nconstr 0 -100.0 100.0 \
-nconstr 1 -1.0 1.0  \
-sconstr 0 45.0 75.0 \
```

```
-sconstr 1 -500.0 500.0  \
-sconstr 2 0.00 0.15  \
-sconstr 3 0.15 0.50  \
-nrand 500  \
-nbest 5  \
-rmsmin 1.0  \
-fdisp 100.0  \
-bucket 0 fred.bucket
```

■

The batch command file is that same as for Example 1, except that the last 8 lines (8 output commands) have been replaced by the single -bucket command. The '0' following -bucket indicates that the default bucket dataset should be created, and that the output should be written to file fred.bucket+orig (.HEAD and .BRIK). The bucket dataset has the structure:

Brick	Label	Contents
#0	constant	$g_n[0]$ = non-linear L.S. est. of $\gamma_n[0]$
#1	linear	$g_n[1]$ = non-linear L.S. est. of $\gamma_n[1]$
#2	t0	$g_s[0]$ = non-linear L.S. est. of $\gamma_s[0]$
#3	k	$g_s[1]$ = non-linear L.S. est. of $\gamma_s[1]$
#4	alpha1	$g_s[2]$ = non-linear L.S. est. of $\gamma_s[2]$
#5	alpha2	$g_s[3]$ = non-linear L.S. est. of $\gamma_s[3]$
#6	Signal TMax	Time of signed maximum of signal
#7	Signal SMax	Signed maximum of signal
#8	Signal % SMax	Signed maximum of signal as a % of baseline
#9	Signal Area	Area under signal (always positive)
#10	Signal % Area	(Signed) area under signal as a % of baseline
#11	Sigma Resid	$\sigma = \sqrt{MS(\text{Error})}$
#12	R^2	$R^2 = 1 - \frac{SSE(F)}{SSE(R)}$
#13	F-stat Regression	$F^{**} = \frac{MS(\text{Reg})}{MS(\text{Error})}$

Note that the labels for the six parameters in the model are taken directly from the model definition shared-object files. These labels are used within afni to identify the individual sub-bricks.

Example 3. Drug Response (continued)

The previous example used the default specification for the bucket dataset output. If this is not acceptable, it is possible to explicitly define the labels and contents for each of the sub-bricks in the bucket dataset, as shown below.

Batch Command File for Example 3

```
3dNLfim  \
-input fred+orig \
-ignore 3  \
-noise Linear \
-signal DiffExp \
-nconstr 0 -100.0 100.0 \
-nconstr 1 -1.0 1.0  \
-sconstr 0 45.0 75.0 \
-sconstr 1 -500.0 500.0  \
-sconstr 2 0.00 0.15  \
-sconstr 3 0.15 0.50  \
-nrand 500  \
-nbest 5  \
-rmsmin 1.0  \
-fdisp 100.0  \
-bucket 8 fred.bucket \
-brick 0 scoef 0 ``Response Time'' \
-brick 1 scoef 1 ``Mult.  Constant'' \
-brick 2 scoef 2 ``Elimin.  Rate'' \
-brick 3 scoef 3 ``Absorp.  Rate'' \
-brick 4 tmax ``Signal TMax'' \
-brick 5 psmax ``Signal % SMax'' \
-brick 6 parea ``Signal % Area'' \
-brick 7 fstat ``F-stat Regression''
```

■

The '8' following -bucket indicates that the bucket dataset will contain 8 sub-bricks, and that the output should be written to file fred.bucket+orig (.HEAD and .BRIK). This is followed by 8 -brick commands, which specify the contents and labels for each of the 8 sub-bricks in the bucket dataset. The bucket dataset has the structure:

Brick	Label	Contents
#0	Response Time	$g_s[0]$ = non-linear L.S. est. of $\gamma_s[0]$
#1	Mult. Constant	$g_s[1]$ = non-linear L.S. est. of $\gamma_s[1]$
#2	Elimin. Rate	$g_s[2]$ = non-linear L.S. est. of $\gamma_s[2]$
#3	Absorp. Rate	$g_s[3]$ = non-linear L.S. est. of $\gamma_s[3]$
#4	Signal TMax	Time of signed maximum of signal
#5	Signal % SMax	Signed maximum of signal as a % of baseline
#6	Signal % Area	Signed area under signal as a % of baseline
#7	F-stat Regression	$F^{**} = \dfrac{MS(\text{Reg})}{MS(\text{Error})}$

In this way, the structure of the bucket dataset can be tailored to the needs of the user.

2 Program plug_nlfit

2.1 Purpose

Program plug_nlfit is an *AFNI* "plug-in" which displays the nonlinear least squares fit of the user specified signal (+ noise) waveform on top of the actual time series data for voxels of interest. Program plug_nlfit is the interactive version of the batch command program 3dNLfim. The reader is strongly advised to consult the documentation for program 3dNLfim first.

2.2 Usage

To use plug_nlfit, first one must be running afni. Display the Image and Graph for Axial, Sagittal, or Coronal views. Choose Define Datamode. This will popup the datamode menu. From the last line of the menu, choose Plugins. This presents a menu of the different *AFNI* plugins that are available. Choose NLfit & NLerr.

This displays the NLfit & NLerr popup control box. At the top are four control buttons: Quit, to close the popup without using the plugin; Run + Keep, to run the plugin and keep the popup window open; Run + Close, to run the plugin and close the popup window; and Help, to popup a help window. Below this, there are five option lines, labeled Control, Models, Noise, Signal, and Time Scale.

On the Control option line, there are three number choosers: Ignore, NRandom, and NBest. The Ignore box allows the user to specify how many of the initial time series data points to ignore when performing the non-linear least squares fit. The default value is to ignore the first 3 data points in each time series. The second number chooser on the Control option line is labeled NRandom. This option allows the user to specify the number of parameter vectors to be randomly chosen from the parameter space for evaluation. The default value for NRandom is 100. The next number chooser, NBest, specifies how many of these random vectors will be used as the initial point for the nonlinear optimization algorithm. The default value is 5.

Below the Control option line is the Models option line, which, as the name implies, allows the user to select the model to be used in performing the non-linear least squares fit of the time series data. The Noise Model and Signal Model options allow the user to choose among previously defined noise and signal models. During initialization, program plug_nlfit attempts to locate all signal and noise models that have been compiled and stored on disk. Each signal model and each noise model has an associated name or label (see documentation for Signal and Noise Models). The third option on this line is Noise Constr. The user can choose between Relative (default) and Absolute constraints for the noise parameters (see documentation for program 3dNLfim for a description of the difference between absolute and relative noise constraints).

The next option line is labeled Noise. This option line allows the user to manually set the minimum and maximum constraints for each noise parameter. The parameters are identified by number, from 0 to $r - 1$, where r is the number of parameters in the noise model. The default values for the noise parameter constraints are contained in the file which defines the specific noise model.

The fourth option line is labeled Signal. This option line allows the user to manually set the minimum and maximum constraints for each signal parameter. The parameters are identified by number, from 0 to $p-1$, where p is the number of parameters in the signal model. The default values for the signal parameter constraints are contained in the file which defines the specific signal model.

The final option line is labeled Time Scale. Here, the user can choose between Internal time reference (default), External time reference, and -inTR. For Internal time reference, the time increment (used in calculating the model time series) is set equal to 1.0. If External time reference is chosen, the user must enter the name of the file containing the sequence of time coordinates for successive volumes in the time series data. Selecting -inTR means that the time increment is equal to the value for TR which is stored with the input 3d+time dataset.

2.3 Examples

Example 1. Drug Response (Differential Exponential Model)

We will assume that the current subdirectory contains the *AFNI* 3d+time dataset of interest, which we will take to be fred+orig, plus statistical parameter output files, such as fred.k. To start the program, type afni. From the main menu, click on Axial (or Sagittal, or Coronal) Image. Once the image is displayed, it can be resized or moved to another (more convenient) location. Next, click on Switch Functions. From the pop-up menu, choose fred.k [fift], which contains the *AFNI* F-statistic map which was generated by program 3dNLfim. Click on Set to select this option. Next, click on Define Function. Adjust the F-statistic probability threshold using the vertical bar. Now, click on See Function. In the axial (or sagittal, or coronal) image view, those voxels whose nonlinear regression model is significant at the specified probability threshold will light up. The color coding for those voxels which light up indicates the sign and magnitude of the least squares estimate for parameter k (in this case). (Note that other parameters may be viewed by clicking the Switch Function box, and then choosing the appropriate file name corresponding to the parameter in question).

To view the observed time series at a particular location, use the mouse to change the placement of the crosshairs within the image. To see the corresponding time series for voxels indicated by the crosshairs, click on Axial (or Sagittal, or Coronal) Graph. The time series plots for a 3×3 grid of voxels pops up. The time series can be vertically rescaled by using the '+' and '-' keys.

To initialize the nonlinear regression analysis program, first click on Define Datamode. From the popup box, click on Plugins. This pops up a list of the different plugin programs that are available. From this list, choose NLfit & NLerr. This pops up the NLfit & NLerr Function control box. For the Noise Model, choose Linear, and for the Signal Model, choose DiffExp (for the differential - exponential drug response model). Then, press Run + Keep. The program will then write to the text window the following information:

Program:	plug_nlfit
Author:	B. Douglas Ward
Date:	10 May 2000

Controls:

Ignore	=	3
Num Random	=	500
Num Best	=	5
Noise Constr	=	Relative

Noise Model = Linear

gn[0] :	min = −100.000	max = 100.000	constant
gn[1] :	min = −1.000	min = 1.000	linear

Signal Model = DiffExp

gs[0] :	min = 45.000	max = 75.000	t0
gs[1] :	min = −500.000	max = 500.000	k
gs[2] :	min = 0.000	max = 0.150	alpha1
gs[3] :	min = 0.150	max = 0.500	alpha2

Internal Time Reference

■

To overlay a plot of the nonlinear estimate of the signal + noise time series on top of the observed time series, do the following: Click on Opt, and select Double Plot. Then, click on Opt again, and select Tran 1D, and select NLfit. For each voxel whose time series is displayed, the program writes relevant information into the text window, such as shown below:

Voxel Results:

Reduced (Linear) Model:

b[0] =	4250.861328	constant
b[1] =	−0.256537	linear

Full (Linear + DiffExp) Model:

gn[0] =	4203.534180	constant
gn[1] =	−0.194454	linear
gs[0] =	57.528519	t0
gs[1] =	230.609360	k
gs[2] =	0.026249	alpha1
gs[3] =	0.457229	alpha2

Signal Tmax	=	64.000
Signal Smax	=	182.620
Signal PSmax	=	4.357
Signal Area	=	8053.441
Signal PArea	=	0.982

RMSE Rdcd	=	60.225
RMSE Full	=	31.341

R^2	=	0.735
F[4,191]	=	132.268
p-value	=	6.479387e-54

■

To view the reduced and full model parameter estimates, rms error, and F-statistic corresponding to a particular voxel, move the cursor into the box containing the time series for that voxel, and press the right-most button on the mouse (i.e., Button 3). This pops-up a display window with the information corresponding to that voxel.

3 Program 3dTSgen

3.1 Purpose

Program 3dTSgen provides a means of generating artificial time series data, and storing such data into an *AFNI* 3d+time dataset. The time series data is generated using the operator specified signal and noise models. Such artificial time series data is useful in several ways:

1) Testing of statistical analysis programs for significance of the results. Artificial time series can be generated corresponding to the null hypothesis under consideration. Then, when input to a statistical analysis program, one can determine how often a false positive occurs; i.e., the probability of rejecting the null hypothesis, when it is, in fact, true.

2) Calculation of the statistical power of a test. Artificial time series can be generated corresponding to the alternative hypothesis under consideration. Then, when input to a statistical analysis program, one can determine how often the signal is detected; i.e., the probability of rejecting the null hypothesis, when it is, in fact, false. This enables one to estimate the power of the test.

3) Design of experiments. Various parameters of an experiment are under the researcher's control. Program 3dTSgen can be used to determine the importance of the parameters. For example, the researcher may wish to change the length of the time series data. Using previous experience to specify the signal and noise models, the researcher could use program 3dTSgen to determine the effect of time series length upon the statistical power of the test.

3.2 Usage

The syntax for execution of program 3dTSgen is as follows:

3dTSgen -input fname -noise nlabel -signal slabel \
[**-nconstr 0** $\mathbf{a}_0$ $\mathbf{b}_0$] $\cdots$ [**-nconstr r-1** $\mathbf{a}_{r-1}$ $\mathbf{b}_{r-1}$] \
[**-sconstr 0** $\mathbf{c}_0$ $\mathbf{d}_0$] $\cdots$ [**-sconstr p-1** $\mathbf{c}_{p-1}$ $\mathbf{d}_{p-1}$] \
-sigma s [**-voxel num**] **-output prefixname** \
[**-ncoef k prefixname**] [**-scoef k prefixname**]
[**-bucket n prefixname**] [**-brick** $\mathbf{m}_1$ **options**] ... [**-brick** $\mathbf{m}_n$ **options**]

The different command line options are explained below.

3.3 Options

-input *fname*

The mandatory -input command specifies that *fname* is the filename of the *AFNI* 3d + time data set to be used as the "prototype" for the output file. The prototype defines the dimensions of the output dataset, as well as the length of the time series.

-noise nlabel

The mandatory -noise command tells program 3dTSgen which noise model to use. During initialization, program 3dTSgen attempts to locate all noise models that have been compiled and stored on disk. Each noise model has an associated name or label. If program 3dTSgen finds *nlabel* among the noise model names, then that noise model will be used for generating the artificial time series. Otherwise, the program prints an error message and halts.

-signal slabel

The mandatory -signal command tells program 3dTSgen which signal model to use. During initialization, program 3dTSgen attempts to locate all signal models that have been compiled and stored on disk. Each signal model has an associated name or label. If program 3dTSgen finds *slabel* among the signal model names, then that signal model will be used for generating the artificial time series. Otherwise, the program prints an error message and halts.

-ncnstr k $\mathbf{c}_k$ $\mathbf{d}_k$

The optional -ncnstr command specifies the constraints on the noise model parameter values. The kth noise parameter is selected at random, with a uniform density, over the specified interval:

$$c_k <= g_n[k] <= d_k,$$

To fix the value for a specific noise parameter, the user can set $c_k = d_k$. The default values for the noise parameter constraints are specified in the file which defines the noise model.

-scnstr k $\mathbf{c}_k$ $\mathbf{d}_k$

The optional -scnstr command specifies the constraints on the signal model parameter values. The kth signal parameter is selected at random, with a uniform density, over the specified interval:

$$c_k <= g_s[k] <= d_k.$$

To fix the value for a specific signal parameter, the user can set $c_k = d_k$. The default values for the signal parameter constraints are specified in the file which defines the signal model.

-sigma s

The mandatory -sigma command is used to set the standard deviation s of the additive Gaussian noise. This random noise will be independent at each voxel and at each time point.

-voxel num

The optional -voxel command is used to control output to the user's terminal during program execution. If this command is used, the individual time step values of the time series, as well as the signal and noise parameters, are written to the screen for voxel number num. Note that the -voxel command effects screen output only, and has absolutely no effect upon the data file output generated by the program.

-output *prefixname*

The mandatory output command tells program 3dTSgen to write the 3d+time dataset to the file have the specified *prefixname*. The output dataset has n sub-bricks, where n is the length of the time series.

$$\textit{AFNI}\ \text{3d+time dataset} \quad \left\{ \begin{array}{l} \boxed{Y[0]} \\ \boxed{Y[1]} \\ \vdots \\ \boxed{Y[n-1]} \end{array} \right.$$

-ncoef *k prefixname*

The optional -ncoef command is used to save the "true" randomly generated values of the kth noise parameter. The result is stored as an *AFNI* "fim" dataset.

$$\textit{AFNI}\ \text{“fim” dataset} \quad \left\{ \boxed{\gamma_n[k] = \text{“true” } k\text{th noise parameter}} \right.$$

-scoef *k prefixname*

The optional -scoef command is used to save the "true" randomly generated values of the kth signal parameter. The result is stored as an *AFNI* "fim" dataset.

$$\textit{AFNI}\ \text{“fim” dataset} \quad \left\{ \boxed{\gamma_s[k] = \text{“true” } k\text{th signal parameter}} \right.$$

-bucket n *prefixname*

The -bucket command is used to create a single *AFNI* "bucket" type dataset having n sub-bricks. The output is written to the file with the user specified prefix filename. Each of the individual sub-bricks can then be accessed for display within program afni. The purpose of this command is to simplify file management, since all of the output parameter datasets (but *not* the 3d+time dataset itself) for a particular problem can now be contained within the single *AFNI* bucket dataset.

If $n = 0$, then the *default* output bucket dataset is created. (See Example 3.) The labels for the individual signal and noise parameters come from the model definition source code. The default dataset has $p + r$ sub-bricks, as illustrated below.

Brick	Label	Contents
#0	(Label for noise par. #0)	$g_n[0]$ = non-linear L.S. est. of $\gamma_n[0]$
$\vdots$	$\vdots$	$\vdots$
#r-1	(Label for noise par. #r-1)	$g_n[r-1]$ = non-linear L.S. est. of $\gamma_n[r-1]$
#r	(Label for signal par. #0)	$g_s[0]$ = non-linear L.S. est. of $\gamma_s[0]$
$\vdots$	$\vdots$	$\vdots$
#p+r-1	(Label for signal par. #p-1)	$g_s[p-1]$ = non-linear L.S. est. of $\gamma_s[p-1]$

If $n > 0$, then the contents and labels for the individual sub-bricks within the bucket dataset are specified by the user, by means of the -brick command described below. Note that the -bucket command *must* precede the -brick commands.

-brick m ***options***

The -brick command is used to specify the contents and labels for the mth sub-brick ($0 \leq m < n$) within the bucket dataset. There must be one -brick command for each of the n sub-bricks in the dataset (where n has been previously specified by the -bucket command).

There are 2 versions of the -brick command:

-brick m ncoef k label	The mth sub-brick is to contain the *true* (randomly generated) value for the kth parameter in the noise model.
-brick m scoef k label	The mth sub-brick is to contain the *true* (randomly generated) value for the kth parameter in the signal model.

In each case, the label for the sub-brick is specified by *label*.

3.4 Examples

Example 1. Statistical Significance Calculation

A researcher wishes to assess the probability of obtaining false positives for a set of data. From previous experience, a time series under the null condition can be represented by a linear trend plus independent Gaussian noise.

Batch Command File for Example 1

```
3dTSgen  \
-input fred+orig \
-noise Linear \
-signal Null \
-sigma 25.0 \
-nconstr 0 -100.0 100.0 \
-nconstr 1 -1.0 1.0  \
-voxel 12345 \
-output noise.ts
```

■

The -input command indicates that the *AFNI* 3d+time dataset fred+orig should be used as the prototype for the output dataset; i.e., the number of voxels and the time series length are taken from dataset fred+orig. The -noise command indicates that the Linear noise model (linear trend+Gaussian noise) should be used. The -signal command indicates that the program should use the Null signal model (no signal model, i.e., just noise is present). The noise parameter constraints are specified by the -nconstr commands. The -voxel command results in the time series and parameter values being printed to the screen for voxel #12345. Finally, the -output command writes the *AFNI* 3d+time dataset to the output file with prefix noise.ts.

Now, the output file noise.ts can be used as input to other 3d+time statistical analysis programs, e.g., 3dfim, 3dNLfim, etc. For example, if noise.ts is used as input to program

3dNLfim, the resulting F-statistic output file for significance of the nonlinear regression can be used to assess the probability of a false detection.

Example 2. Statistical Power Calculation

To estimate the probability of detecting a signal when the signal is present (i.e., the statistical power), the following batch command file was used to execute program 3dTSgen.

Batch Command File for Example 2

```
3dTSgen  \
-input fred+orig \
-noise Linear \
-signal DiffExp \
-sigma 25.0 \
-nconstr 0 -100.0 100.0 \
-nconstr 1 -1.0 1.0  \
-sconstr 0 45.0 75.0 \
-sconstr 1 -500.0 500.0  \
-sconstr 2 0.00 0.15  \
-sconstr 3 0.15 0.50  \
-voxel 12345 \
-output signal.ts  \
-ncoef 0 signal.b0  \
-ncoef 1 signal.b1  \
-scoef 0 signal.t0  \
-scoef 1 signal.k  \
-scoef 2 signal.alpha1  \
-scoef 3 signal.alpha2
```

■

As in Example 1, the -input command indicates that the *AFNI* 3d+time dataset fred+orig should be used as the prototype for the output dataset. The -noise command indicates that the Linear noise model (linear trend+Gaussian noise) should be used. The -signal command indicates that the program should use the DiffExp signal model (differential - exponential drug response function). The noise model parameter constraints are specified by the -nconstr commands, and the signal model parameter constraints are specified by the -sconstr commands. The -voxel command results in the time series and parameter values being printed to the screen for voxel #12345. The -output command writes the *AFNI* 3d+time dataset to the output file with prefix signal.ts. The -ncoef and -scoef commands store the noise and signal model parameters that were used to generate the time series at each voxel. These values are stored in separate AFNI datasets for each parameter.

Now, if the output file signal.ts is used as input to program 3dNLfim, the resulting F-statistic output files can be used to assess the statistical power of the test in detecting signal buried in noise.

Example 3. Statistical Power Calculation (continued)
In the previous example, the noise and signal model parameters that were used to generate the time series at each voxel were written to separate datasets. In order to simplify file management, all of these parameters can be written to a single 'bucket' dataset, as shown below.

Batch Command File for Example 3

```
-input fred+orig \
-noise Linear \
-signal DiffExp \
-sigma 25.0 \
-nconstr 0 -100.0 100.0 \
-nconstr 1 -1.0 1.0  \
-sconstr 0 45.0 75.0 \
-sconstr 1 -500.0 500.0  \
-sconstr 2 0.00 0.15  \
-sconstr 3 0.15 0.50  \
-voxel 12345 \
-output signal.ts  \
-bucket 0 signal.parameters
```

■

The batch command file is that same as for Example 2, except that the last 6 lines (6 output commands) have been replaced by the single -bucket command. The '0' following -bucket indicates that the default bucket dataset should be created, and that the output should be written to file signal.parameters+orig (.HEAD and .BRIK). The bucket dataset has the structure:

Brick	Label	Contents
#0	constant	noise parameter $g_n[0]$ used to generate time series
#1	linear	noise parameter $g_n[1]$ used to generate time series
#2	t0	signal parameter $g_s[0]$ used to generate time series
#3	k	signal parameter $g_s[1]$ used to generate time series
#4	alpha1	signal parameter $g_s[2]$ used to generate time series
#5	alpha2	signal parameter $g_s[3]$ used to generate time series

Note that the labels for the six parameters in the model are taken directly from the model definition shared-object files. These labels are used within afni to identify the individual sub-bricks.

4 Signal and Noise Models

4.1 Signal Models

Presently, nine different signal models are included with the *AFNI* source code. These are listed below.

Label	Description	Parameters
Null	No Signal	—
SineWave_AP	Sinusoidal Response	Amplitude, Phase
SquareWave_AP	Square Wave Response	Amplitude, Phase
TrnglWave_AP	Triangular Wave Response	Amplitude, Phase
SineWave_APF	Sinusoidal Response	Amplitude, Phase, Frequency
SquareWave_APF	Square Wave Response	Amplitude, Phase, Frequency
TrnglWave_APF	Triangular Wave Response	Amplitude, Phase, Frequency
DiffExp	Differential-Exponential Drug Response	t0, k, alpha1, alpha2
GammaVar	Gamma-Variate Function Drug Response	t0, k, r, b

The actual signal models are defined in the C source code files model_null.c, model_sinewave_ap.c, . . ., model_gammavar.c. These source code files have been compiled into the "shared object" files model_null.so, model_sinewave_ap.so, . . ., model_gammavar.so.

If a new signal model is to be added, or if a pre-existing model is modified, then only the model itself has to be compiled. Programs 3dNLfim, plug_nlfit, and 3dTSgen will then automatically have access to the new or modified model.

Each signal model contains two separate function routines. The function initialize_model is called first (by program 3dNLfim, or plug_nlfit, or 3dTSgen) to initialize the signal model prior to execution. The second function, signal_model, does the actual calculation of the time series values corresponding to the specified model with the given parameter values.

4.2 Initializing the Signal Model

The function initialize_model is used to establish the name, number of parameters, labels for the parameters, and default values for the parameter constraints. The function returns a data structure of type MODEL_interface, as defined below:

```
typedef struct {
   char label[MAX_NAME_LENGTH];                      /* name of the model */
   int model_type;                            /* noise or signal model?  */
   int params;                                   /* number of parameters */
   char plabel[MAX_PARAMETERS][MAX_NAME_LENGTH];    /* parameter labels */
   float min_constr[MAX_PARAMETERS];  /* minimum parameter constraints */
   float max_constr[MAX_PARAMETERS];  /* maximum parameter constraints */
   void_func * call_func;      /* function which implements the model */
} MODEL_interface;
```

The initialize_model function must declare a variable to be of type MODEL_interface, and each member of the MODEL_interface structure must then be initialized. In the following, we will assume that variable mi has been declared to be of type MODEL_interface.

The user must assign a unique name to each model (this name is used by the programs to identify the model itself). If, for example, the user wants to assign the name "Drug Response" to the model, then the corresponding c-language statement would be:

```
strcpy (mi->label, ''DrugResponse'');
```

Note that no space is left between "Drug" and "Response"; this is for reasons connected with execution of the batch command file.

The type of model must be specified. Each model is either a MODEL_NOISE_TYPE or a MODEL_SIGNAL_TYPE. Since the user is specifying a signal model, the function must include the statement:

```
mi->model_type = MODEL_SIGNAL_TYPE;
```

The number of parameters in the model must be specified. At present, there is a limit of 10 parameters for the signal model. If the user's model has 3 parameters, then the following statement must appear:

```
mi->params = 3;
```

Each of the parameters must be given a name. The parameters are numbered consecutively, starting with 0. Therefore, the third parameter would be given the name "beta" by the statement:

```
strcpy (mi->plabel[2], ''beta'');
```

Minimum and maximum (default) constraints must be specified for *each* parameter in the model. Again, it must be emphasized that the parameters are numbered consecutively, starting with 0. So, the constraints for the 2nd parameter could be specified by:

```
mi->min_constr[1]= -100.0; mi->max_constr[1]= 250.0;
```

The last member of the mi structure is used to indicate the name of the function to be called which implements the model itself. If the function name is signal_model, then the following statement must appear:

```
mi->call_func = &signal_model;
```

A listing of function intialize_model for the differential-exponential drug response model appears below.

Listing of function initialize_model from file model_diffexp.c

```
/*
Routine to initialize the signal model by defining
the number of parameters in the signal model, the name of
the signal model, and the default values for the minimum
and maximum parameter constraints.
*/
MODEL_interface * initialize_model ( )
{
   MODEL_interface * mi = NULL;

   /*----- allocate memory space for model interface -----*/
   mi = (MODEL_interface *) XtMalloc (sizeof(MODEL_interface));

   /*----- define interface for the differential - exponential
   model -----*/

   /*----- name of this model -----*/
   strcpy (mi->label, ''DiffExp'');

   /*----- this is a signal model -----*/
   mi->model_type = MODEL_SIGNAL_TYPE;

   /*----- number of parameters in the model -----*/
   mi->params = 4;

   /*----- parameter labels -----*/
   strcpy (mi->plabel[0], ''t0'');
   strcpy (mi->plabel[1], ''k'');
   strcpy (mi->plabel[2], ''alpha1'');
   strcpy (mi->plabel[3], ''alpha2'');

   /*----- minimum and maximum parameter constraints -----*/
   mi->min_constr[0]= 45.0; mi->max_constr[0]= 75.0;
   mi->min_constr[1]= -500.0; mi->max_constr[1]= 500.0;
   mi->min_constr[2]= 0.00; mi->max_constr[2]= 0.15;
   mi->min_constr[3]= 0.15; mi->max_constr[3]= 0.50;

   /*----- function which implements the model -----*/
   mi->call_func = &signal_model;

   /*----- return pointer to the model interface -----*/
   return (mi);
}
```

The C code for the differential-exponential drug response model indicates: the output label for this model is "DiffExp"; there are 4 parameters in the model; and the signal model parameter names and constraints are as follows:

$$
\begin{array}{lclcl}
45.0 & \leq & \text{gs}[0] \equiv \text{"t0"} & \leq & 75.0 \\
-500.0 & \leq & \text{gs}[1] \equiv \text{"k"} & \leq & 500.0 \\
0.00 & \leq & \text{gs}[2] \equiv \text{"alpha1"} & \leq & 0.15 \\
0.15 & \leq & \text{gs}[3] \equiv \text{"alpha2"} & \leq & 0.50
\end{array}
$$

Note that these are the *default* values for the signal parameter constraints, and they may be overridden by the user in either program 3dNLfim, or program plug_nlfit, or program 3dTSgen.

4.3 Defining the Signal Model

A listing of function signal_model from model_diffexp.c appears below. There are 3 inputs to the function: gs, an array containing the estimated parameter vector; ts_length, the number of time series data points; and x_array, which contains the independent variable(s) (currently, this is just the time index). As output, the function calculates the time series array ts_array.

As indicated below, the signal model is the "DiffExp" drug response model. Recall that the differential-exponential drug response model is:

$$Y_i = k \left[e^{-\alpha_1 (t_i - t_0)} - e^{-\alpha_2 (t_i - t_0)} \right]$$

This is translated into C code as:

```
fval = gs[1]*(exp(-gs[2]*(t-gs[0]))- exp(-gs[3]*(t-gs[0])));
```

where

gs[0] $= t_0$,
gs[1] $= k$,
gs[2] $= \alpha_1$,
gs[3] $= \alpha_2$,
t = x_array[it][1] $= t_i$,
fval = ts_array[it] $= Y_i$.

The correspondence between the model parameters and the elements of the gs array is arbitrary, but the user must be consistent. The x_array is actually a matrix; column 0 consists of all 1's (this represents the constant term), column 1 consists of the integers from 0 to ts_length-1 (this represents time t, where time is measured in units corresponding to the interval between successive images), and column 2 consists of the squares of column 1 (representing t^2). The complete signal_model function is listed below.

Listing of function signal_model from file model_diffexp.c

```
/*
Routine to calculate the time series which results from
using the differential exponential drug response signal
model with the specified model parameters.

Definition of model parameters:
   gs[0] = time delay of response (t0)
   gs[1] = multiplicative constant (k)
   gs[2] = elimination rate constant (alpha1)
   gs[3] = absorption rate constant (alpha2)
*/
void signal_model
(
   float * gs,                       /* parameters for signal model */
   int ts_length,                      /* length of time series data */
   float ** x_array,                  /* independent variable matrix */
   float * ts_array             /*estimated signal model time series*/
)

{
   int it;                                          /* time index */
   float t;                                               /* time */
   float fval;                       /* time series value at time t */
   for (it = 0; it < ts_length; it++)
      {
         t = x_array[it][1];
         if (t < gs[0])
            fval = 0.0;
         else
            fval = gs[1] * (exp(-gs[2]*(t-gs[0])) -
                        exp(-gs[3]*(t-gs[0])));
         ts_array[it] = fval;
      }
}
```

■

Particularly note in the above code that if $t < gs[0]$, then $fval$ is set to 0. This is required so that the "signal" does not occur prior to $t_0 = gs[0]$. Any model where the signal "switches on" at a particular time must incorporate similar logic.

4.4 Compilation

Once the initialize_model and the signal_model functions have been created, the C source code should be stored in one file, under a file name beginning with "model", and ending with ".c", such as model_new.c. This file should be stored in the same directory with the *AFNI* source code.

To facilitate compilation of the source code to a shared object, file Makefile.INCLUDE (distributed with MCW *AFNI*) contains the necessary commands. In Makefile.INCLUDE, locate the line beginning with "models:". Following "models:" there is a list of the file names for the different pre-defined models. The new model file prefix name should be appended to this list, e.g.,

```
models:  model_constant.$(SO) model_linear.$(SO) model_quadratic.$(SO) \
            ⋮
         model_diffexp.$(SO) model_gammavar.$(SO) model_new.$(SO)
```

Now, after Makefile.INCLUDE has been updated, the new model shared object file is created through the command:

make models

The computer responds by compiling all model source code files which have been added or modified since the last compilation. The shared object file model_new.so should now appear in the directory listing.

At initialization, Programs 3dNLfim, plug_nlfit, and 3dTSgen will search for the model libraries. These programs will look in the directories specified in the shell environment variable AFNI_MODELPATH. If this variable does not exist, then these programs will use the PATH variable instead. For example, if the following command appears in the .cshrc file:

```
setenv AFNI_MODELPATH /user/fred/AFNI96
```

then the programs will search directory /user/fred/AFNI96 for model libraries. All shared object libraries, in the specified directories, whose name begins with "model", and which contain the initialize_model function, will be loaded into the program. Note that storing plugins and models in the same directory is perfectly acceptable. However, duplicate model names should be avoided at all cost.

4.5 Noise Models

Presently, there are 3 different noise models included with the source code. These are listed below.

Label	Description	Parameters
Constant	Constant + Gaussian Noise	Constant
Linear	Linear Trend + Gaussian Noise	Constant, Linear
Quadratic	Quadratic + Gaussian Noise	Constant, Linear, Quadratic

The actual noise models are defined in source code files model_constant.c, model_linear.c, and model_quadratic.c. The noise models, once they have been compiled into dynamic library "shared object" files, are used by programs 3dNLfim, plug_nlfit, and 3dTSgen.

Two separate function routines are involved in the definition of the noise models. The function intialize_model is called first to establish the name, number of parameters, parameter labels, and default values for the noise parameter constraints. The second function, noise_model, does the actual calculation of the time series values corresponding to the specified noise model with the given parameter values.

At the present time, the set of noise models is limited to those listed above. However, as mentioned in the documentation for Program 3dNLfim, the capability to include additional independent variables (such as the "ort" time series of Program 3dfim) may be added in the future. This would allow the user to extend the list of noise models.

It should be noted that under the current paradigm, the "noise" model is always a "linear" model. This was done for simplicity, since this makes calculation of the reduced (noise) model a linear regression problem. It is not absolutely necessary that the noise model be linear. However, allowing nonlinear noise models would require altering the program(s) so that both the reduced and the full models would use nonlinear parameter estimation (thus increasing run time). The author invites comments on the utility of nonlinear noise models.

5 References

1. P. A. Bandettini, A. Jesmanowicz, E. C. Wong, J. S. Hyde, Processing strategies for time-course data sets in functional MRI of the human brain. *Magn. Reson. Med.* 30, 161-173 (1993).

2. R. W. Cox, A. Jesmanowicz, J.S. Hyde, Real-time functional magnetic resonance imaging. *Magn. Reson. Med.* 33, 230-236 (1995).

3. C. R. Craig, R. E. Stitzel, *Modern Pharmacology*, 2nd edition. Boston: Little, Brown and Company (1982).

4. W. W. Orrison Jr., J. D. Lewine, J. A. Sanders, M. F. Hartshorne, *Functional Brain Imaging*. St. Louis: Mosby (1995).

5. S. S. Rao, *Optimization Theory and Applications*, 2nd edition. New Delhi: Wiley (1984).

6. J. W. Cooper, R. B. Lam, *A Jump Start Course in C++ Programming*. New York: Wiley (1994).

7. J. Neter, W. Wasserman, M. H. Kutner, *Applied Linear Statistical Models*, 2nd edition. Homewood, Illinois: Irwin (1985).