# Deconvolution Analysis of FMRI Time Series Data

B. Douglas Ward
Biophysics Research Institute
Medical College of Wisconsin
email: ward@mcw.edu
Initial Release: September 8, 1998
Latest Revision:

May 3, 2002

**Abstract**

Program 3dDeconvolve was developed to provide deconvolution analysis of FMRI time series data. This has two primary applications: (1) estimation of the system impulse response function, and (2) multiple linear regression analysis of time series data. Given the input stimulus function(s), and the measured FMRI signal data, program 3dDeconvolve first estimates the impulse response function(s); the impulse response function(s) is then convolved with the stimulus time series to yield the estimated response. Various statistics are calculated to indicate the "goodness" of the fit.

The capability of fitting *multiple* stimulus (or reference) waveforms differentiates program 3dDeconvolve from the cross-correlation analysis programs (such as AFNI fim and 3dfim). Another way that program 3dDeconvolve differs from the cross-correlation analysis programs is in the model for the output waveform. The cross-correlation programs model the system response as a scaled version of a fixed waveform (such as a square wave or a sine wave). Program 3dDeconvolve uses a sum of scaled and time-delayed versions of the stimulus time series. The data itself determines (within limits) the functional form of the estimated response. In fact, the shape of the fitted waveform can vary from voxel to voxel. Program 3dDeconvolve is described in Section 1.

Since program 3dDeconvolve was developed for use in a "batch" processing mode, it should be used in conjunction with the interactive program plug_deconvolve. See the documentation for program plug_deconvolve for a description of this interactive version. The details are contained in Section 2.

Program RSFgen is a simple program for generating random stimulus functions. This capability may be useful for experimental design, and the evaluation of experimental designs. This program is described in Section 3.

Program 3dConvolve, as the name implies, performs the inverse operation of program 3dDeconvolve. That is, program 3dConvolve convolves the input stimulus functions with the given system impulse response functions, in order to predict the output measured response. This program is described in Section 4.

# Contents

# 1 Program 3dDeconvolve

## 1.1 Purpose

Program 3dDeconvolve was developed to provide deconvolution analysis of FMRI time series data. This has two primary applications: (1) estimation of the system impulse response function, and (2) multiple linear regression analysis of time series data. Given the input stimulus function(s), and the measured FMRI signal data, program 3dDeconvolve first estimates the impulse response function(s); the impulse response function(s) is then convolved with the stimulus time series to yield the estimated response. Various statistics are calculated to indicate the "goodness" of the fit.

The capability of fitting *multiple* stimulus (or reference) waveforms differentiates program 3dDeconvolve from the cross-correlation analysis programs (such as AFNI fim and 3dfim). Another way that program 3dDeconvolve differs from the cross-correlation analysis programs is in the model for the output waveform. The cross-correlation programs model the system response as a scaled version of a fixed waveform (such as a square wave or a sine wave). Program 3dDeconvolve uses a sum of scaled and time-delayed versions of the stimulus time series. The data itself determines (within limits) the functional form of the estimated response. In fact, the shape of the fitted waveform can vary from voxel to voxel.

The input to program 3dDeconvolve consists of an *AFNI* 3d+time data set, along with one or more input stimulus time series. Output consists of the estimated system impulse response function, along with the statistical significance of the fit of this impulse response function to the original FMRI data, for each voxel in the dataset. In addition to the regression coefficients, the program calculates the F-statistic and $R^2$ for significance of the multiple regression, as well as t-statistics for each of the impulse response function parameters. In the case of multiple input stimuli, the program calculates the partial F-statistics and partial $R^2$ for significance of each individual stimulus.

The user has the option of performing one or more general linear tests on the model parameters; e.g., a within-run test for differences in response to different stimuli. The general linear test is defined by the user in the form of a matrix, whose rows represent the multiple linear constraints on the model parameters. The program calculates the specified linear combinations, as well as the F-statistic for significance of the general linear test.

An alternative input option is the -input1D command, which can be used for performing the deconvolution analysis on a single measured FMRI time series. Also, see the -nodata option, which allows the user to evaluate an experimental design prior to collecting data. Specifically, the experimental design can be tested for multicollinearity. Also, the relative accuracies of different hypothetical experimental designs can be evaluated.

Other output options include -fitts, which writes the full model time series fit for each voxel to a 3d+time dataset. Also, see option -errts, which writes the residual errors to a 3d+time dataset. Analysis of the residuals may indicate where the full model is inadequate, or needs to be improved. Also, this option can be used to detrend data by removing polynomial functions, linear combinations of stimulus functions, etc.

## 1.2 Theory

### 1.2.1 Impulse Response Function

The paradigm for almost all FMRI experiments consists of measuring the hemodynamic consequences of the neural response to a specific time-varying input stimulus condition. Letting $f(t)$ represent the time course of the stimulus condition, and $y(t)$ represent the measured FMRI signal for a particular voxel, the experiment can be represented diagrammatically by:

$$f(t) \longrightarrow \boxed{\text{System}} \longrightarrow y(t)$$

The system itself may be quite complex; in some cases it may be useful to think of the system as decomposed into simpler subsystems:

$$f(t) \longrightarrow \boxed{\text{Subsystem 1}} \longrightarrow \boxed{\text{Subsystem 2}} \longrightarrow \boxed{\text{Subsystem 3}} \longrightarrow y(t)$$

where the output from one subsystem is the input to the next subsystem. For example, suppose that $f(t)$ represents a drug injection. Then subsystem 1 could represent the effect of the circulatory system upon drug concentration in the blood. The neuronal response to drug concentration is then represented by subsystem 2. The effect of neuron activation upon blood oxygen level, and hence upon the FMRI signal, is represented by subsystem 3. Each of these subsystems could be further decomposed into sub-subsystems, etc.

We cannot observe directly the internal workings of the system. However, we are able to observe the response of the system to specific inputs. Under certain conditions, this is sufficient to characterize the response of the system to an arbitrary input.

In the following, we will make certain assumptions about the nature of the system. First, we will assume that the system is *linear*. That is, if input $f(t)$ produces response $y(t)$,

$$f(t) \longrightarrow \boxed{\text{System}} \longrightarrow y(t)$$

and input $g(t)$ produces response $z(t)$,

$$g(t) \longrightarrow \boxed{\text{System}} \longrightarrow z(t)$$

then for arbitrary constants $a$ and $b$, we have:

$$af(t) + bg(t) \longrightarrow \boxed{\text{System}} \longrightarrow ay(t) + bz(t)$$

In other words, the response to a linear combination of the inputs is the same linear combination of the responses to the respective inputs.

Another assumption that we will make about the nature of the system is that it is *time invariant*. That is, if the input is delayed by some time $t_0$, then the response is simply delayed by $t_0$:

$$\begin{aligned} f(t) &\longrightarrow \boxed{\text{System}} \longrightarrow y(t) \\ f(t - t_0) &\longrightarrow \boxed{\text{System}} \longrightarrow y(t - t_0) \end{aligned}$$

If the system under consideration has both these properties, then we say that the system is *linear time invariant.*

The response of a linear time invariant system to an arbitrary input can be easily determined from its response to an *impulse function* (or *Dirac delta function*). An impulse function, $\delta(t)$, is a theoretical construct, having infinite height and zero width,

$$\delta(t) = \begin{cases} +\infty, & t = 0 \\ 0, & t \neq 0 \end{cases}$$

such that the area under its curve is unity:

$$1 = \int_{-\varepsilon}^{+\varepsilon} \delta(\tau) d\tau, \ \varepsilon > 0.$$

The impulse function is defined mathematically by the following integral equation:

$$f(t) = \int_{-\infty}^{+\infty} f(\tau)\delta(t-\tau)d\tau$$

where it is assumed that $f(t)$ is continuous at $t = \tau$.

Now, consider using the impulse function as the input to a system. Of course, such a function cannot be physically applied to a system. But we can hypothesize that the system output will be a function $h(t)$, called the *impulse response function.*

$$\delta(t) \longrightarrow \boxed{\text{System}} \longrightarrow h(t)$$

In the following, we will use the operator $\mathbf{T}$ to represent the mapping between the input to the system, and the output response of the system, e.g.,

$$h(t) = \mathbf{T}\{\delta(t)\}$$

### 1.2.2 Convolution Integral

(This section is based upon Ref. [1]). Since we seldom (actually, never) apply an impulse function to a system, why bother trying to estimate the impulse response function? The answer is that the impulse response function $h(t)$ can be used to describe the system response to an arbitrary input $f(t)$. Let $y(t)$ be the system output corresponding to input $f(t)$:

$$y(t) = \mathbf{T}\{f(t)\}$$

Substituting the defining property for the impulse function, we have:

$$y(t) = \mathbf{T}\left\{\int_{-\infty}^{+\infty} f(\tau)\delta(t-\tau)d\tau\right\}$$

Replacing the integral by summation:

$$y(t) = \mathbf{T}\left\{\lim_{\Delta t \to 0} \sum_{n=-\infty}^{+\infty} f(n\Delta t)\delta(t - n\Delta t)\Delta t\right\}$$

Since the system is linear, the operator $\mathbf{T}$ can be moved inside the summation:

$$y(t) = \lim_{\Delta t \to 0} \sum_{n=-\infty}^{+\infty} f(n\Delta t)\mathbf{T}\left\{\delta(t - n\Delta t)\right\}\Delta t$$

Using the definition of the impulse response function, and since the system is time invariant, we have:

$$y(t) = \lim_{\Delta t \to 0} \sum_{n=-\infty}^{+\infty} f(n\Delta t)h(t - n\Delta t)\Delta t$$

Finally, replacing summation by integration, we get:

$$y(t) = \int_{-\infty}^{+\infty} f(\tau)h(t-\tau)d\tau$$

This is the *convolution integral*, and is denoted:

$$\int_{-\infty}^{+\infty} f(\tau)h(t-\tau)d\tau \equiv f(t) \otimes h(t)$$

If $h(t)$ is the impulse response for a system which is composed of a cascade of subsystems, as illustrated above, then the system impulse response is obtained by convolving the impulse responses of the subsystems, i.e.,

$$h(t) = h_1(t) \otimes h_2(t) \otimes h_3(t)$$

so the output can be represented by:

$$y(t) = f(t) \otimes h_1(t) \otimes h_2(t) \otimes h_3(t)$$

Physical systems are causal, i.e., the output at time $t_0$ is determined by the inputs at times $t \leq t_0$, and not by future inputs (i.e., inputs at times $t > t_0$). Therefore, for causal systems, $h(t) = 0$ for $t < 0$; hence, the convolution integral can be written:

$$y(t) = \int_{-\infty}^{t} f(\tau)h(t-\tau)d\tau$$

If we further assume that the input is zero prior to time $t = 0$, i.e., $f(t) = 0$ for $t < 0$, then the convolution integral becomes:

$$y(t) = \int_{0}^{t} f(\tau)h(t-\tau)d\tau$$

### 1.2.3 Numerical Solution

The output is measured at discrete times. In the following, we will approximate the convolution integral by summation over discrete time:

$$y(n\Delta t) = \sum_{m=0}^{n} f(m\Delta t)h(n\Delta t - m\Delta t)\Delta t.$$

We will let $\Delta t \equiv 1$, and switch to the more convenient subscript notation:

$$\begin{aligned} y_n &= \sum_{m=0}^{n} f_m h_{n-m} \\ &= \sum_{m=0}^{n} f_{n-m} h_m. \end{aligned}$$

Usually it is the case that the impulse response function decreases to zero with increasing time. We will assume that the impulse response function is essentially zero for time lags greater than $p$. In this case, the above summation becomes:

$$y_n = \sum_{m=0}^{p} f_{n-m} h_m, \quad n \geq p.$$

Note that "lag" refers to times into the past; that is, $h_m$ represents the influence of the stimulus at time point $n - m$ on the data at time point $n$.

Of course, the measurement process is not perfect. If we assume that the measurements include additive, uncorrelated, Gaussian noise, then we get the sequence of random variables:

$$Z_n = \sum_{m=0}^{p} h_m f_{n-m} + \varepsilon_n, \quad n \geq p$$

where $\varepsilon_n \overset{iid}{\sim} N(0, \sigma^2)$. For FMRI data, it is often the case that the measurement can be modeled by a constant plus linear trend plus noise, in addition to the signal:

$$\begin{aligned} Z_n &= y_n + \beta_0 + \beta_1 n + \varepsilon_n \\ &= \beta_0 + \beta_1 n + h_0 f_n + h_1 f_{n-1} \cdots + h_p f_{n-p} + \varepsilon_n, \\ \text{for } n &= p, p+1, \ldots, N-1. \end{aligned}$$

Using the matrix notation

$$\mathbf{Z} = \begin{bmatrix} Z_p \\ Z_{p+1} \\ \vdots \\ Z_{N-1} \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & p & f_p & \cdots & f_0 \\ 1 & p+1 & f_{p+1} & \cdots & f_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & N-1 & f_{N-1} & \cdots & f_{N-p-1} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ h_0 \\ \vdots \\ h_p \end{bmatrix}, \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_p \\ \varepsilon_{p+1} \\ \vdots \\ \varepsilon_{N-1} \end{bmatrix}.$$

the above equation can be written:

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

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. In this fit to the model, the time series data is then estimated by:

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

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

$$\begin{aligned} SSE &= Q(\mathbf{b}) = \sum_{i=0}^{N-1} \left( Z_i - \hat{Z}_i \right)^2 \\ &= \left( \mathbf{Z} - \hat{\mathbf{Z}} \right)^t \left( \mathbf{Z} - \hat{\mathbf{Z}} \right) \end{aligned}$$

It is easy to show that

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

is the least squares estimate of $\boldsymbol{\beta}$.

Since $\mathbf{b}$ contains the estimated impulse response parameters,

$$\mathbf{b} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \widehat{h}_0 \\ \vdots \\ \widehat{h}_p \end{bmatrix},$$

we see that solving for the estimated impulse response function reduces to simple linear algebra.

Program 3dDeconvolve calculates the estimated impulse response function at each voxel location, and appends these values as sub-bricks of an *AFNI* "bucket" dataset. Also, if requested by the user, program 3dDeconvolve saves the estimated impulse response function for each voxel into an *AFNI* 3d+time dataset. In this case, the dataset time series has length $p + 1$. See the Examples in Section 1.4.2.

### 1.2.4 Multicollinearity

Note that the above formula for calculating the estimated parameter vector $\mathbf{b}$ requires calculation of the matrix inverse: $\left( \mathbf{X}^t \mathbf{X} \right)^{-1}$. This might not be possible, due to the structure of the $\mathbf{X}$ matrix. This problem is referred to as the multicollinearity problem. The multicollinearity problem is NOT a limitation of program 3dDeconvolve; rather, it is a mathematical limitation.

Consider this example: Let $f(k)$ = time series for a single stimulus function. Suppose that $f(k)$ is periodic; say that $f(k)$ is "on" for one TR, and "off" for 4 TR. Since $f(k)$ has period 5 TR, it is obvious that we cannot estimate the impulse response function (IRF) for time lags 0 through 5. This is because the input stimulus, with a time lag of 0, is identical to the input stimulus with a time lag of 5, $f(k) \equiv f(k-5)$. Therefore, it is not possible to determine the individual contributions of these factors in explaining the output.

(An analogy is this: Suppose that a researcher is doing multiple regression analysis. The dependent variable is subject's weight. There are 2 independent variables: subject's height, measured in feet, and subject's height, measured in meters. Since the independent variables are perfectly correlated, the problem cannot be solved. More accurately, there are infinitely many solutions.)

Multicollinearity can arise more subtly. Suppose, in the above example, that we reduce the maximum time lag for the estimated IRF from 5 TR to 4 TR. Then,

$$
\begin{array}{lll}
f(k) & = \{ & 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ \ldots\} \\
f(k-1) & = \{ & 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ \ldots\} \\
f(k-2) & = \{ & 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ \ldots\} \\
f(k-3) & = \{ & 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ \ldots\} \\
f(k-4) & = \{ & 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ \ldots\} \\
\hline
Sum & = \{ & 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \ldots\}
\end{array}
$$

i.e.,

$$f(k) + f(k-1) + f(k-2) + f(k-3) + f(k-4) \equiv 1.$$

We see that the sum of $f$ at time lags 0, 1, 2, 3, and 4 is equal to a constant. This means that there is ambiguity as to the source of an observed response: is it due to a constant offset, or is it due to the combined response to stimulus $f$ at time lags 0 through 4?

As described above, time delayed versions of the input stimulus function become columns of the $\mathbf{X}$ matrix. For the above example, these columns are linearly dependent, and thus the problem is not solvable. Mathematically, the $\mathbf{X}^t\mathbf{X}$ matrix is not invertible.

Some general comments: Periodicity of the stimulus function is often responsible for multicollinearity. If the data has already been collected, the only solution is to reduce the length of the estimated IRF, until multicollinearity is no longer a problem. However, if the experiment is in the design phase, it should be possible to avoid multicollinearity from the outset. When practical, randomization of the stimulus function is recommended. Unless the "random" stimulus function is quite unusual, this should help avoid multicollinearity. This has the additional benefit of reducing the likelihood of confounding of the stimulus function with natural biological processes (e.g., respiration). Also, an unpredictable stimulus function helps to avoid "anticipatory" effects. The sample program RSFgen may be used to generate random stimulus functions (see Section 3).

Therefore, before conducting an experiment, it is recommended that you first check whether the columns of the experimental design matrix are linearly dependent (or almost linearly dependent). This can be done using the `-nodata` option, described in Section 1.3. Also, see the Examples in Section 1.4.1.

### 1.2.5 F-test for Significance of the 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 constant plus linear trend, plus independent Gaussian random variates. And suppose the "signal" waveform is represented by the *convolution* of the impulse response function $h(t)$ with the input stimulus function $f(t)$. The observed "signal" is then represented by the sum of the signal and the noise. The hypotheses can then be expressed as:

$$\begin{aligned} H_o &: Z_n = \gamma_0 + \gamma_1 n + \varepsilon_n \\ H_a &: Z_n = \beta_0 + \beta_1 n + h_0 f_n + h_1 f_{n-1} + \cdots + h_p f_{n-p} + \varepsilon_n \end{aligned}$$

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., $\beta_0$ and $\gamma_0$, $\beta_1$ and $\gamma_1$. This is to emphasize the fact that it is not (in general) true that $\beta_0 = \gamma_0$ and $\beta_1 = \gamma_1$.

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 "baseline" model) and the parameters which yield the least squares fit for the signal plus noise model (also called the "full" model). Let $SSE(B)$ be the residual sum of squares obtained from fitting the baseline model:

$$SSE(B) = \sum_n \left(Z_n - (\hat{\gamma}_0 + \hat{\gamma}_1 n)\right)^2$$

and let $SSE(F)$ be the residual sum of squares obtained from fitting the full model:

$$SSE(F) = \sum_n \left(Z_n - \left(\widehat{\beta}_0 + \widehat{\beta}_1 n + \widehat{h}_0 f_n + \widehat{h}_1 f_{n-1} + \cdots + \widehat{h}_p f_{n-p}\right)\right)^2$$

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 baseline model ($SSE(B)$). However, if $SSE(F)$ is only slightly smaller than $SSE(B)$, then we do not have reason to reject the null hypothesis. Consider the test statistic $F^*$ :

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

where $df_B$ is the number of degrees of freedom for the baseline model, and $df_F$ is the number of degrees of freedom for the full model. Specifically, we have (assuming that the noise model contains the two parameters for constant plus linear trend):

$$\begin{aligned} df_B &= N^{'} - 2, \\ df_F &= N^{'} - 2 - (p+1), \\ \text{so } df_B - df_F &= p + 1. \end{aligned}$$

where $N' = N - p$ is the number of usable data points.

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. The statistic $F^*$ has the $F(df_B - df_F, df_F)$ distribution under the null hypothesis (Ref. [2]).

Program 3dDeconvolve calculates the $F^*$ statistic for each voxel, and (if the -fout option is used) appends these values as one of the sub-bricks of an *AFNI* "bucket" dataset.

### 1.2.6 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 $R^2$:

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

where $SSE(F)$ and $SSE(B)$ are defined above. Roughly speaking, $R^2$ is the proportion of the variation in the data (about the baseline) that is explained by the full regression model. Note that, for every voxel, $0 \leq R^2 \leq 1$. ($R^2$ is a generalization of the square of the correlation coefficient computed in the fim programs).

Program 3dDeconvolve calculates $R^2$ for each voxel, and (if the -rout option is used) appends these values as one of the sub-bricks of an *AFNI* "bucket" dataset.

### 1.2.7 t-test for Significance of Individual Parameters

When comparing different impulse response functions, it is useful to know the significance of the individual terms that constitute the impulse response function. This may help in deciding whether different impulse response functions are truly different, or merely reflect the influence of measurement noise. Therefore, program 3dDeconvolve provides the $t$-statistics for the individual terms in the impulse response function.

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

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

Then, for large sample size $N$, define the statistic $t^*$ :

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

where $s(h_k)$ is the square root of the corresponding diagonal element of the $\mathbf{s}^2(\mathbf{b})$ matrix. Under the null hypothesis, $t^*$ has the $t(N' - p - 3) = t(N - 2p - 3)$ distribution (Ref. [2]).

Program 3dDeconvolve calculates the $t^*$ statistic for each point in the impulse response function, at each voxel location, and (if the -tout option is used) appends these values as sub-bricks of an *AFNI* "bucket" dataset for each parameter. Also, if requested by the user (see option -sresp), program 3dDeconvolve saves the array of standard deviations $\{s(h_k), k = 0, ..., p\}$ for each voxel into an *AFNI* 3d+time dataset. In this case, the dataset time series has length $p + 1$.

As mentioned above, the sample standard deviation $s(h_k)$ of the estimate for parameter $h_k$ is obtained as the square root of the corresponding diagonal element of the $\mathbf{s}^2(\mathbf{b})$ matrix. The $\mathbf{s}^2(\mathbf{b})$ matrix is calculated by multiplying the scalar $MSE$ (the mean square error, which estimates the measurement error variance) by the matrix $(\mathbf{X}^t\mathbf{X})^{-1}$. Now, assuming that the measurement error variance is a constant (for a given voxel), we see that $\mathbf{s}^2(\mathbf{b})$ is a function of $\mathbf{X}$ alone. In other words, the accuracy of the estimate for parameter $h_k$ is determined by the structure of the experimental design matrix $\mathbf{X}$. But the structure of $\mathbf{X}$ is determined by the input stimulus function(s) $f(t)$. Therefore, we see that the experimental design directly influences the accuracy of the parameter estimates.

The significance of this is: we can numerically evaluate the relative accuracy of different hypothetical experimental designs *prior* to collecting data. Knowing the input stimulus function(s) $f(t)$, the matrix $\mathbf{X}$ can be assembled, and $(\mathbf{X}^t\mathbf{X})^{-1}$ computed. Setting the unknown scalar $MSE$ to 1, the *normalized* standard deviation $s(h_k)$ can be calculated, as described above. This procedure can be repeated for different hypothetical stimulus functions, thus allowing a comparison of alternative experimental designs prior to data collection. See the Examples in Section 1.4.1.

### 1.2.8 Multiple Linear Regression

Thus far, we have considered only a single input stimulus function. However, the extension to multiple input stimuli is straightforward. Suppose that the system response $y(t)$ is a linear combination of the responses to input stimuli $u(t)$, $v(t)$, and $w(t)$:

$$y(t) = \gamma_u u(t) + \gamma_v v(t) + \gamma_w w(t)$$

Then the measured response, including linear drift and additive noise, and with the measurements occurring at discrete times $n$, is given by:

$$Z_n = \beta_0 + \beta_1 n + \gamma_u u_n + \gamma_v v_n + \gamma_w w_n + \varepsilon_n$$

where $\varepsilon(t) \overset{iid}{\sim} N(0, \sigma^2)$. This assumes zero time delay between the input stimulus and the measured response. However, there may be some time delay, and the amount of time delay may be different for different voxels. We will use the same approach as was used in estimating the impulse response function, i.e., the output will be modeled by the *sum* of lagged versions of the input functions. For example, if we consider time lags of 0, 1, and 2 time units for each of the input stimuli, then the measured response can be modeled thus:

$$\begin{aligned} Z_n \;=\; & \beta_0 + \beta_1 n + \gamma_{u,0} u_n + \gamma_{u,1} u_{n-1} + \gamma_{u,2} u_{n-2} \\ & + \gamma_{v,0} v_n + \gamma_{v,1} v_{n-1} + \gamma_{v,2} v_{n-2} \\ & + \gamma_{w,0} w_n + \gamma_{w,1} w_{n-1} + \gamma_{w,2} w_{n-2} + \varepsilon_n \end{aligned}$$

This multiple linear regression problem can be written in matrix form as:

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

where

$$\mathbf{Z} = \begin{bmatrix} Z_2 \\ Z_3 \\ \vdots \\ Z_N \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & 2 & u_2 & u_1 & u_0 & v_2 & v_1 & v_0 & w_2 & w_1 & w_0 \\ 1 & 3 & u_3 & u_2 & u_1 & v_3 & v_2 & v_1 & w_3 & w_2 & w_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & N & u_N & u_{N-1} & u_{N-2} & v_N & v_{N-1} & v_{N-2} & w_N & w_{N-1} & w_{N-2} \end{bmatrix},$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \gamma_{u,0} \\ \gamma_{u,1} \\ \gamma_{u,2} \\ \gamma_{v,0} \\ \gamma_{v,1} \\ \gamma_{v,2} \\ \gamma_{w,0} \\ \gamma_{w,1} \\ \gamma_{w,2} \end{bmatrix}, \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_N \end{bmatrix}.$$

Calculation of the least squares estimate for the vector of regression coefficients, the F-test for significance of the regression, and the t-tests for significance of the individual parameters, proceed just as described above for the impulse response function. See the Examples in Section 1.4.3.

### 1.2.9 Partial F-statistics

It will usually be the case that, when multiple input stimuli are being studied, the investigator will wish to determine the significance of each individual stimulus in explaining the variation in the data. The F-statistic for the regression, described above, can only be used for testing if all of the stimuli combined are statistically significant, but does not indicate whether an individual stimulus is significant. The t-test, described above, tests for the significance of individual parameters. If several time lags are used for modeling the response to an individual stimulus, then more than one parameter is associated with that stimulus; therefore, the t-test does not indicate the significance of that stimulus.

In order to test for the significance of an individual stimulus, the partial F-statistic is calculated:

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

Here, $SSE(F)$ is the error sum of squares from fitting the *full model* (i.e., the model with all parameters included), and $SSE(R)$ is the error sum of squares after fitting the *reduced model* (i.e., the same model except that those parameters corresponding to the stimulus in question are *excluded*). For example, suppose we wish to calculate the partial $F^*$ for the input stimulus $v(t)$. Then the error sum of squares for the full model and the reduced model are calculated as follows:

$$\begin{aligned} SSE(F) &= SSE(\beta_0, \beta_1, \gamma_{u,0}, \gamma_{u,1}, \gamma_{u,2}, \gamma_{v,0}, \gamma_{v,1}, \gamma_{v,2}, \gamma_{w,0}, \gamma_{w,1}, \gamma_{w,2}) \\ SSE(R) &= SSE(\beta_0, \beta_1, \gamma_{u,0}, \gamma_{u,1}, \gamma_{u,2}, \gamma_{w,0}, \gamma_{w,1}, \gamma_{w,2}) \end{aligned}$$

In this particular case, letting $N' \equiv$ the number of usable data points, we have: $df_F = N' - 11$ and $df_R = N' - 8$, hence $df_R - df_F = 3$ (which is the number of time lags, or parameters, used for modeling the response to input stimulus $v(t)$). The statistic $F^*$ has the $F(df_R - df_F, df_F)$ distribution under the null hypothesis (Ref. [2]).

Program 3dDeconvolve calculates the partial $F^*$ statistic for each input stimulus for each voxel, and (if the -fout option is used) appends these values as sub-bricks of an *AFNI* "bucket" dataset.

#### 1.2.10 Coefficient of Partial Determination

Recall that the coefficient of multiple determination, $R^2$, indicates how well the full model fits the data. The coefficient of partial determination, or partial $R^2$, measures the marginal contribution of a single stimulus function, or of a single GLT (see General Linear Test, described below). We therefore define the partial $R^2$ thus:

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

where $SSE(F)$ is the error sum of squares from fitting the *full model* (i.e., the model with all parameters included), and $SSE(R)$ is the error sum of squares after fitting the *reduced model* (i.e., the same model except that those parameters corresponding to the stimulus in question are *excluded*). Or, for a GLT, $SSE(R)$ represents the error sum of squares after fitting the model, subject to the multiple linear constraints imposed on the parameters by the GLT.

Roughly speaking, the partial $R^2$ is the proportional reduction of the variation in the data that is explained by adding the stimulus function, given that all other stimulus functions are already included in the model. Note that, for every voxel, $0 \leq$ partial $R^2 \leq 1$.

Program 3dDeconvolve calculates the partial $R^2$ for each stimulus function and for each general linear test, for every voxel, and (if the -rout option is used) appends these values as sub-bricks of an *AFNI* "bucket" dataset.

#### 1.2.11 General Linear Test

As a generalization of the above partial $F$-test, we will consider the test of a null hypothesis which involves $s$ linear constraints on the model parameters.

$$H_o : \mathbf{C}\boldsymbol{\beta} = \mathbf{0}$$

where $\mathbf{C}$ is an $s \times P$ matrix, and $P =$ total number of parameters in the full model. It can be shown (see Ref. [2]) that the least squares estimate for $\boldsymbol{\beta}$ which satisfies the above set of linear constraints is given by:

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

where $\mathbf{b}_F$ = least squares estimate of $\boldsymbol{\beta}$ under the full model, and $\mathbf{b}_R$ = least squares estimate of $\boldsymbol{\beta}$ under the reduced model. The error sum of squares for this reduced model is then:

$$SSE(R) = (\mathbf{Z} - \mathbf{X}\mathbf{b}_R)^t(\mathbf{Z} - \mathbf{X}\mathbf{b}_R)$$

Finally, the test statistic for the general linear test (GLT) is:

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

which has the $F(df_R - df_F, df_F)$ distribution under the null hypothesis (Ref. [2]), where $df_R - df_F = s$.

Writing the $s \times P$ matrix $\mathbf{C}$ in terms of row vectors:

$$\mathbf{C} = \begin{bmatrix} \mathbf{c}_1^t \\ \mathbf{c}_2^t \\ \vdots \\ \mathbf{c}_s^t \end{bmatrix}$$

we can estimate $\mathbf{C}\boldsymbol{\beta}$ by:

$$\mathbf{C}\mathbf{b}_F = \begin{bmatrix} \mathbf{c}_1^t\mathbf{b}_F \\ \mathbf{c}_2^t\mathbf{b}_F \\ \vdots \\ \mathbf{c}_s^t\mathbf{b}_F \end{bmatrix} \equiv \begin{bmatrix} L_1 \\ L_2 \\ \vdots \\ L_s \end{bmatrix}$$

where $L_1$, $L_2$, $\ldots$, $L_s$ are the $s$ linear combinations of the parameter vector $\mathbf{b}_F$ specified by the GLT.

Program 3dDeconvolve calculates the $s$ linear combinations $L_1$, $L_2$, $\ldots$, $L_s$, and the GLT $F^*$ statistic, for each operator specified GLT, and for each voxel, and appends these values as sub-bricks of an *AFNI* "bucket" dataset. (See the Examples in Section 1.4.4.)

### 1.2.12 Concatenation of Runs

Many users choose to concatenate runs prior to time series analysis. This is a very delicate operation, due to the implicit time dependence across runs. That is, consecutive image volumes are assumed to have been acquired at consecutive points in time; but for concatenated runs, this is not the case. In order to facilitate analysis of concatenated runs, program 3dDeconvolve has been modified to allow the user to indicate the volume indices, of a concatenated dataset, at which the different runs begin. Internally, the program makes allowance for the fact that an input stimulus from one run should not effect the measured response for the following run.

Another change that is required for processing of concatenated runs is the addition of separate baseline parameters for each run. That is, if the baseline is modeled by a constant plus linear trend, then a separate constant and slope are estimated for each run. This would be necessary even if program 3dTcat had been used to remove the linear trend from each run prior to concatenation. The reason is that the stimulus functions are seldom orthogonal to a constant plus linear trend. Hence, removing the linear trends, in isolation, does not yield the same result as the complete least squares solution.

As a consequence, if the baseline model is to represent a constant offset plus linear trend (i.e., 2 parameters) for each run, and if the concatenated dataset contains $r$ runs, then the baseline model contains a total of $2r$ parameters.

Suppose that the input dataset is a concatenation of 3 runs; each run has length $N$; the single input stimulus function $f$ is present at time lags 0 through $p$. Since the individual runs are modeled by:

$$\begin{aligned}
Z_n &= \beta_{1,0} + \beta_{1,1} n + h_0 f_n + h_1 f_{n-1} \cdots + h_p f_{n-p} + \varepsilon_n, \quad n = p, p+1, \ldots, N-1, \ \text{for Run \#1} \\
Z_n &= \beta_{2,0} + \beta_{2,1} n + h_0 f_n + h_1 f_{n-1} \cdots + h_p f_{n-p} + \varepsilon_n, \quad n = p, p+1, \ldots, N-1, \ \text{for Run \#2} \\
Z_n &= \beta_{3,0} + \beta_{3,1} n + h_0 f_n + h_1 f_{n-1} \cdots + h_p f_{n-p} + \varepsilon_n, \quad n = p, p+1, \ldots, N-1, \ \text{for Run \#3}
\end{aligned}$$

the single concatenated run can be modeled by:

$$\begin{aligned}
Z_n &= \left(\beta_{1,0} + \beta_{1,1} n\right) X_1 + \left(\beta_{2,0} + \beta_{2,1}(n-N)\right) X_2 + \left(\beta_{3,0} + \beta_{3,1}(n-2N)\right) X_3 \\
&\quad + h_0 f_n + h_1 f_{n-1} \cdots + h_p f_{n-p} + \varepsilon_n, \\
n &= p, p+1, \ldots, N-1, N+p, N+p+1, \ldots, 2N-1, 2N+p, 2N+p+1, \ldots, 3N-1,
\end{aligned}$$

where

$$\begin{aligned}
X_1 &= \begin{cases} 1 & \text{if time point is from Run \#1} \\ 0 & \text{otherwise} \end{cases} \\
X_2 &= \begin{cases} 1 & \text{if time point is from Run \#2} \\ 0 & \text{otherwise} \end{cases} \\
X_3 &= \begin{cases} 1 & \text{if time point is from Run \#3} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

In this notation, $\beta_{i,j}$ is the coefficient of $t^j$ for the $i$th run.

Therefore, the matrix representation of the full model is:

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

where

$$\mathbf{Z} = \begin{bmatrix} Z_p \\ Z_{p+1} \\ \vdots \\ Z_{N-1} \\ Z_{N+p} \\ Z_{N+p+1} \\ \vdots \\ Z_{2N-1} \\ Z_{2N+p} \\ Z_{2N+p+1} \\ \vdots \\ Z_{3N-1} \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & p & 0 & 0 & 0 & 0 & f_p & \cdots & f_0 \\ 1 & p+1 & 0 & 0 & 0 & 0 & f_{p+1} & \cdots & f_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & N-1 & 0 & 0 & 0 & 0 & f_{N-1} & \cdots & f_{N-p-1} \\ 0 & 0 & 1 & p & 0 & 0 & f_{N+p} & \cdots & f_N \\ 0 & 0 & 1 & p+1 & 0 & 0 & f_{N+p+1} & \cdots & f_{N+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & N-1 & 0 & 0 & f_{2N-1} & \cdots & f_{2N-p-1} \\ 0 & 0 & 0 & 0 & 1 & p & f_{2N+p} & \cdots & f_{2N} \\ 0 & 0 & 0 & 0 & 1 & p+1 & f_{2N+p+1} & \vdots & f_{2N+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 1 & N-1 & f_{3N-1} & \cdots & f_{3N-p-1} \end{bmatrix},$$

$$\boldsymbol{\beta}^T = \left[\ \beta_{1,0} \quad \beta_{1,1} \quad \beta_{2,0} \quad \beta_{2,1} \quad \beta_{3,0} \quad \beta_{3,1} \quad h_0 \quad \cdots \quad h_p\ \right].$$

Implementation: Program 3dDeconvolve allows the user to enter the name of a file which simply contains a short (column) listing of the volume indices for the starting point of each concatenated run. All other details are handled internally. (See the description of the -concat option in Section 1.3, and the Examples in Section 1.4.6.)

### 1.2.13 Censoring of Individual Time Points

Occasionally, a user may wish to remove one or more time points from the analysis (e.g., wild points due to large measurement noise). In *AFNI* fim / 3dfim+, this is accomplished by placing "99999" at each time point in the ideal time series that is to be excluded. A similar approach would not work for program 3dDeconvolve, due to the time-lagged dependence on the stimulus functions. Therefore, a separate "censor" file can now be used to indicate which data points are to be excluded from the analysis, without effecting the temporal representation of the input stim functions. The censor file consists of a column of all 1's, except for 0's at those time points to be censored. (See the description of the -censor option in Section 1.3, and the Examples in Section 1.4.7)

### 1.2.14 Time Related Issues

As described in Section 1.2.3, for the deconvolution analysis, the time axis is quantized in steps of TR. This is reasonable, since the measured data is acquired at intervals of 1 TR for each voxel. However, different slices in the imaging volume are usually acquired at different relative times. In order to accurately compare IRF's across slices, either graphically, or by the use of program 3dStatClust, it is necessary to take account of the different slice acquisition times.

Consider again the discrete time convolution equation:

$$y(n\Delta t) = \sum_{m=0}^{n} f(m\Delta t)h(n\Delta t - m\Delta t)\Delta t.$$

This equation relates the output $y$ at the current time $n\Delta t$ to the input $f$ at current and past times $m\Delta t$ through the impulse response function $h$. Now, suppose that $y$ is measured not at the times $n\Delta t$ ($n = 0, 1, 2, ...$), but at $n\Delta t + r\Delta t$, where $r$ is a constant ($-\frac{1}{2} \leq r \leq \frac{1}{2}$). In this case, the above equation becomes:

$$y(n\Delta t + r\Delta t) = \sum_{m=0}^{n} f(m\Delta t)h(n\Delta t - m\Delta t + r\Delta t)\Delta t.$$

Letting $k \equiv n - m$, the estimated impulse response function coefficients $\widetilde{h}$ are actually:

$$\widetilde{h}(k\Delta t) = h(k\Delta t + r\Delta t).$$

Hence, the desired (non-time-shifted) $h$ coefficients are obtained by:

$$h(k\Delta t) = \widetilde{h}(k\Delta t - r\Delta t).$$

Therefore, if we take a slice acquired at time offset $\frac{1}{2}TR$ as the zero reference point, then for slices acquired earlier (i.e., $-\frac{1}{2} \leq r < 0$), the estimated IRF should be shifted to the left by $r \cdot TR$; whereas for slices acquired later (i.e., $0 \leq r < \frac{1}{2}$), the estimated IRF should be shifted to the right by $r \cdot TR$.

The -tshift option has been added to program 3dDeconvolve, in order to compensate for relative differences in slice acquisition times. The program uses cubic spline interpolation to time shift (interpolate) the estimated impulse response functions. The new time scale will be shifted relative to the old time scale by an amount depending on the relative slice acquisition times. Note that this effects *only* the 3d+time output dataset generated by using the -iresp option. In particular, the -tshift option does *not* change the estimated multiple regression coefficients and corresponding statistics stored in the bucket dataset.

## 1.3 Usage

### 1.3.1 Syntax

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

**3dDeconvolve** [**-input** *fname* | **-input1D** *dname* | **-nodata** ] [**-mask** *mname*] [**-censor** *cname*] [**-concat** *rname*] [**-nfirst** *fnum*] [**-nlast** *lnum*] [**-polort** *pnum*] [**-rmsmin** *r*] [**-xout**] [**-fdisp** *fval*] [**-progress** *n*] **-num_stimts** *num* **-stim_file** *k sname* [**-stim_file** *k "sname[j]"*] [**-stim_base** *k*] [**-stim_label** *k slabel*] [**-stim_minlag** *k m*] [**-stim_maxlag** *k n*] [**-stim_nptr** *k p*] **-num_glt** *g* [**-glt** *s gltname*] [**-glt_label** *k glabel*] [**-iresp** *k iprefix*] [**-tshift**] [**-sresp** *k sprefix*] [**-fitts** *fprefix*] [**-errts** *eprefix*] [**-fout**] [**-rout**] [**-tout**] [**-vout**] [**-nobout**] [**-nocout**] [**-full_first**] [**-bucket** *bprefix*]

The different command line options are explained below.

### 1.3.2 Options

**-input *fname***
The -input command specifies that $fname$ is the filename of the *AFNI* 3d+time data set to be used as input for the deconvolution program. The -input command is mandatory *except* when either the -nodata command or the -input1D command is used in its place.

**-input1D *dname***
The -input1D command specifies that *dname* is the filename of the *AFNI* .1D time series data file to be used as input for the deconvolution program. That is, instead of a 3d+time dataset, the input consists of only a single FMRI time series as the measured data. This option allows analysis of, e.g., individual time series obtained from selected voxels, or time series obtained as the average over an ROI.

If this option is used, most output is directed to the screen. Commands which would otherwise generate 3d+time datasets, such as -iresp, -sresp, -fitts, and -errts, will instead create individual .1D time series output files. With -input1D, the -bucket command is ignored.

**-nodata**
The optional -nodata command allows the user to evaluate the experimental design without entering measurement data. This replaces the -input command. The user *must* specify the input stimulus functions. Also, the length of the 3d+time data set *must* be specified using the -nlast command. When the -nodata option is used, all output will go to the screen.

As explained in Section 1.2.3, the input stimulus functions are used to set up the experimental design matrix $\mathbf{X}$. The program will then attempt to calculate the $(\mathbf{X}^t\mathbf{X})^{-1}$ matrix. If the program is unable to calculate the inverse matrix, the program prints the following error message:

3dDeconvolve Error: Improper X matrix (cannot invert X'X)

This error message results if there is multicollinearity in the experimental design (see Section 1.2.4). Otherwise, the program will print out the "normalized" standard deviations for each of the stimulus function coefficients, where the normalized $s(h_k)$ = square root of the corresponding diagonal element of the $(\mathbf{X}^t\mathbf{X})^{-1}$ matrix (see Section 1.2.7).

See Section 1.4.1 for illustrations of the use of this command.

**-mask *mname***

The optional -mask command specifies that *mname* is the filename of the *AFNI* 3d dataset to be used for "masking " the input data. That is, if a voxel in the mask dataset has value zero, then the corresponding voxel in the 3d+time input dataset will be ignored for computational purposes. All output corresponding to that particular voxel will be set to zero. If the mask dataset represents the brain, i.e., if the mask contains 1's only at locations inside the brain, and 0's at locations outside the brain, this will greatly improve the program execution speed. The -mask command provides an alternative to the -rmsmin command described below.

Of course, the mask dataset must have the same voxel dimensions as the input 3d+time dataset.

**-censor *cname***

The optional -censor command is used to specify that *cname* is a .1D file, equal in length to that of the input 3d+time dataset, consisting of 1's and 0's. Data points corresponding to 0's in file *cname* are to be excluded from the analysis. See Section 1.4.7 for illustrations of the use of this command.

**-concat *rname***

The optional -concat command specifies that *rname* is the filename of the *AFNI* .1D time series data file containing a list of the starting points for each individual run within a concatenated dataset. See Section 1.4.6 for illustrations of the use of this command.

**-nfirst *fnum***

The optional -nfirst command specifies that $fnum$ is the number of the first image to be used in the analysis. (The first image in the dataset is numbered 0.) The default value is $fnum$ = maximum of the maximum time lags, as specified below. Note: For concatenated datasets, $fnum$ applies to each individual run within the dataset.

**-nlast *lnum***

The optional -nlast command specifies that $lnum$ is the number of the last image to be used in the analysis. (The first image in the dataset is numbered 0). The default value is $lnum$ = number of the last image in the dataset. Note: For concatenated datasets, $lnum$ applies to each individual run within the dataset.

**-polort *pnum***

The optional -polort command specifies that $pnum$ ($pnum = 0, 1, 2, ...$) is the degree of the polynomial in the baseline model (corresponding to the null hypothesis). The default value is $pnum = 1$ (corresponding to the baseline model: $Z_n = \gamma_0 + \gamma_1 n + \varepsilon_n$; i.e., the signal is a constant plus linear trend plus noise).

Note: The command "-polort -1" can be used to specify *no* baseline model parameters; i.e., the baseline model is: $Z_n = 0 + \varepsilon_n$.

**-rmsmin *r***

The optional -rmsmin command is used to set the minimum rms error $r$ required in order to reject the baseline (noise) model. In other words, the full model will *not* be calculated for those voxels whose time series, when fitted with the baseline 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 smaller than the measurement error for voxels inside the brain. The default value is $r = 0$.

Note: The -mask command, described above, might be a better alternative for speeding program execution.

**-xout**

The optional -xout command is used to write the experimental design matrix $\mathbf{X}$ and $(\mathbf{X}^t\mathbf{X})^{-1}$ to the screen This may help the user to understand what the program is doing. Note that this option effects screen output only, and does not alter the content of the output dataset(s).

**-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 regression $F-$statistic is greater than or equal to $f$, then the estimated baseline and impulse response 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 on the data file output generated by the program. This option is disabled by default.

**-progress *n***

The optional -progress command is similar to the above -fdisp command, except that the screen output is generated every $n$ voxels, regardless of the value of the $F-$statistic. Note that the -progress command effects screen output only, and has absolutely no effect on the data file output generated by the program. This option is disabled by default.

**-num_stimts *num***

The mandatory -num_stimts is used to indicate that *num* input stimulus time series will be used. Note: the -num_stimts command *must precede* the following commands.

**-stim_file *k sname***

The mandatory -stim_file command specifies that *sname* is the filename of the .1D time series representing the $k$th input stimulus function. For multi-column .1D files, this command has the alternative format:

**-stim_file *k "sname[j]"***

In this case, the -stim_file command specifies that the $k$th input stimulus function is contained in column $j$ of file *sname*. Note: The column indexing begins with 0; i.e., the first column corresponds to $j = 0$, etc. Also note that the square brackets around the column index must be enclosed within quotation marks.

**-stim_base *k***

The optional -stim_base command is used to indicate that the $k$th input stimulus function is to be included in the baseline model (corresponding to the null hypothesis). This option is disabled by default; i.e., the $k$th input stimulus function is *not* included in the baseline model.

**-stim_label *k slabel***

The optional -stim_label command is used to specify the string *slabel* for labeling the output corresponding to the $k$th input stimulus function.

**-stim_minlag *k m***

The optional -stim_minlag command specifies the minimum time lag $m$ for estimation of the impulse response function corresponding to the $k$th input stimulus. The default value is $m = 0$.

**-stim_maxlag *k n***

The optional -stim_maxlag command specifies the maximum time lag $n$ for estimation of the impulse response function corresponding to the $k$th input stimulus. The default value is $n = 0$. Note that for each input stimulus, it is required that: $\min lag \leq \max lag$.

**-stim_nptr *k p***

The optional -stim_nptr command specifies that there are $p$ ($p = 1, 2, 3, ...$) time points in the $k$th input stimulus for each TR. The default value is $p = 1$. If the input 3d+time dataset contains $N$ points, then the $k$th input stimulus must contain at least $p \times N$ points. If $p > 1$, then the user *must* align all input slices to 0 time offset beforehand (see program 3dTshift). Note: The IRF time limits specified with -stim_minlag and -stim_maxlag are in multiples of $TR/p$. Therefore, the commands:

-stim_minlag $k$ $m$
-stim_maxlag $k$ $n$
-stim_nptr $k$ $p$

indicate that the $k$th IRF extends from $m \times TR/p$ to $n \times TR/p$.

**-num_glt *g***

The -num_glt command is used to indicate that $g$ general linear tests will be used. The -num_glt command is optional except when $g > 10$. Note: The -num_glt command *must precede* the following -glt commands.

**-glt *s gltname***

The optional -glt command is used to conduct a general linear test. The parameter $s$ specifies the number of linear combinations (or linear constraints) in the general linear test. The matrix which determines the general linear test must be contained in file *gltname*. Note that the matrix must contain $s$ rows, each row specifying a linear constraint on the model parameters, and $P$ columns, each column corresponding to one of the $P$ parameters in the full model.

See the Examples in Section 1.4.4 for illustrations of the use of this command.

**-glt_label *k glabel***

The optional -glt_label command is used to specify the string *glabel* for labeling the output corresponding to the $k$th general linear test.

**-iresp *k iprefix***

The optional -iresp command instructs program 3dDeconvolve to save the estimated impulse response function corresponding to the $k$th input stimulus. That is, each impulse response function for each voxel is saved as a time series of length $p + 1$, and stored into an *AFNI* 3d+time dataset. The output dataset has prefix filename iprefix.

If -iresp is used in conjunction with the -input1D command, then the single estimated impulse response function time series will be written to file iprefix.1D. Note: This will automatically overwrite a pre-existing iprefix.1D file.

**-tshift**

Use cubic spline interpolation to time shift the estimated impulse response function, in order to correct for differences in slice acquisition times. (See Section 1.2.14 for further discussion.) Note that this option effects *only* the 3d+time output dataset generated by the -iresp option.

Note: It is *very* important that the slice acquisition time offsets, which are stored in the .HEAD files, be correct. The -tshift option uses these values to determine how much to shift the impulse response functions for each slice. The time offsets are listed under the label TAXIS_OFFSETS in the header files, and should be inspected by the user.

**-sresp *k sprefix***

The optional -sresp command instructs program 3dDeconvolve to save the array of standard deviations for the impulse response function corresponding to the $k$th input stimulus. That is, the standard deviations for the time points of the impulse response function are made into a time series of length $p + 1$, and stored into an *AFNI* 3d+time dataset. The output dataset has prefix filename sprefix.

If -sresp is used in conjunction with the -input1D command, then the single array of standard deviations will be written to file sprefix.1D. Note: This will automatically overwrite a pre-existing sprefix.1D file.

**-fitts *fprefix***

The optional -fitts command instructs program 3dDeconvolve to save the full model fit to the input time series data for each voxel. The fitted time series are stored into an *AFNI* 3d+time dataset. The output dataset has prefix filename fprefix.

If -fitts is used in conjunction with the -input1D command, then the single fitted time series will be written to file fprefix.1D. Note: This will automatically overwrite a pre-existing fprefix.1D file.

**-errts** ***eprefix***

The optional -errts command instructs program 3dDeconvolve to save the residual errors (i.e., error = data - full model fit) into an *AFNI* 3d+time dataset. The output dataset has prefix filename eprefix.

If -errts is used in conjunction with the -input1D command, then the single residual error time series will be written to file eprefix.1D. Note: This will automatically overwrite a pre-existing eprefix.1D file.

**-bucket** ***bprefix***

The -bucket command is used to create a single *AFNI* "bucket" type dataset having multiple sub-bricks. The output is written to the file with the user specified prefix filename bprefix. 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 most of the output results for a particular problem can now be contained within a single *AFNI* bucket dataset. See Examples 1.4.2.5, 1.4.3.6, 1.4.4.6, and 1.4.6.5 for illustrations of the format of the bucket dataset.

The following commands control the contents of the output bucket dataset:

| | |
|---|---|
| **-fout** | Flag to output the partial model, full model, and GLT $F$-statistics |
| **-rout** | Flag to output the partial model, full model, and GLT $R^2$ |
| **-tout** | Flag to output the $t$-statistic for each regression parameter and for each GLT linear combination |
| **-vout** | Flag to output the sample variance ($MSE$) map |
| **-nobout** | Flag to suppress output of the baseline coefficients (and associated statistics) |
| **-nocout** | Flag to suppress output of the regression coefficients (and associated statistics) |
| **-full_first** | Flag to specify that the full model statistics will appear first in the bucket dataset output |

Note: The -nobout option differs from the -nocout option in that the -nobout option only eliminnates output corresponding to the baseline model; the -nocout option eliminates output corresponding to all regression parameters (including the baseline). Therefore, the -nocout option should be used if the user is only interested in output pertaining to the GLT linear combinations. The -nobout option may be useful when analyzing datasets formed by concatenation of many runs, in which case there would be many sub-bricks for the baseline parameters, which may not be of interest to the user.

## 1.4 Examples

### 1.4.1 Evaluation of the Experimental Design

In the following Examples, we consider evaluation of the experimental design *prior* to collecting data. This is done by using the -nodata option in place of the -input command.

**Example 1.4.1.1 Evaluation of a Block Design**

Suppose that we wish to evaluate a "block" type design, with a repeating stimulus pattern of 4 "Off" TR, followed by 4 "On" TR, as indicated below.

$$
\begin{array}{rcl}
f(t) & = & \{ \begin{array}{cccccccccccccccccccc} 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0 \\ 1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0 \end{array} \}
\end{array}
$$

Save the above $f(t)$, one row at a time, as a single column of 0's and 1's into file Block.1D. Suppose that we wish to estimate the system IRF for 0-4 TR's. Consider the following script:

Program 3dDeconvolve Command Line for Example 1.4.1.1

```
3dDeconvolve \
-nodata \
-nlast 59 \
-polort 0 \
-num_stimts 1 \
-stim_file 1 Block.1D \
-stim_label 1 ''Block'' \
-stim_maxlag 1 4
```

■

The -nodata command is used to specify that there is no input data (i.e., no measurement data). This indicates that the program is only to evaluate the experimental design. There is one input stimulus function (-num_stimts 1), which is specified by the contents of file Block.1D (-stim_file 1 Block.1D). The maximum duration for the estimated IRF is 4 TR (-stim_maxlag 1 4). The program creates the experimental design matrix $\mathbf{X}$ from time lagged versions of the input stimulus functions, as explained in Section 1.2.3.

Try executing the above script. The program will attempt to calculate the $(\mathbf{X}^t\mathbf{X})^{-1}$ matrix. In this particular case, the program is unable to calculate the inverse matrix; therefore, the program prints the following error message:

```
3dDeconvolve Error:  Improper X matrix (cannot invert X'X)
```

This error message results from multicollinearity in the experimental design. See Section 1.2.4 for a discussion of multicollinearity and how to avoid it. Can you explain why this design suffers from multicollinearity? Since the stimulus function has a period of 8 TR, does it surprise you that the problem is not solvable for a maximum time lag of only 4 TR? If you cannot explain the result, add the -xout command option to the above script, and repeat the execution. This time, the program prints out the experimental design matrix $\mathbf{X}$. The $\mathbf{X}$ matrix has 6 columns: column #0 contains all 1's; columns #1 through #5 contain time-lagged versions of the stimulus function (corresponding to 0-4 TR). What is the result if you add column #1 and column #5 (one row at a time)?

If you can write a linear combination (with some coefficients non-zero) of the columns of $\mathbf{X}$ that is equal to a column of zeros, then the columns of $\mathbf{X}$ are said to be linearly dependent. In the current example, we have:

$$1 * \mathbf{C}_0 - 1 * \mathbf{C}_1 - 1 * \mathbf{C}_5 = \mathbf{0}$$

where $\mathbf{C}_i = i$th column of $\mathbf{X}$.

Now, try executing the above script, but reduce maxlag to 3 (i.e., use the command -stim_maxlag 1 3 in place of the -stim_maxlag 1 4 command). Does this resolve the multicollinearity problem? You should see the following printout on the screen:

Program 3dDeconvolve Screen Output for Example 1.4.1.1

```
Program:              3dDeconvolve
Author:               B. Douglas Ward
Initial Release:      02 Sept 1998
Latest Revision:      29 Jan 2002

(X'X) inverse matrix:
  0.0820  -0.0656   0.0000  -0.0000  -0.0656
 -0.0656   0.1382  -0.0714  -0.0000   0.0667
  0.0000  -0.0714   0.1429  -0.0714  -0.0000
 -0.0000  -0.0000  -0.0714   0.1429  -0.0714
 -0.0656   0.0667   0.0000  -0.0714   0.1382

Stimulus:  Block
 h[0]  norm.  std.  dev.  =  0.3717
 h[1]  norm.  std.  dev.  =  0.3780
 h[2]  norm.  std.  dev.  =  0.3780
 h[3]  norm.  std.  dev.  =  0.3717
```

■

If the program is able to evaluate the $(\mathbf{X}^t\mathbf{X})^{-1}$ matrix, this matrix is printed to the screen. As mentioned in Section 1.2.7, the variance-covariance matrix for the regression coefficients is given by:

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

where $MSE$ = sample variance. Since there is no input measurement data, $MSE$ is unknown. However, if $MSE$ is normalized to a value of 1, we see that $(\mathbf{X}^t\mathbf{X})^{-1}$ is the (normalized) variance-covariance matrix. By taking the square roots of the corresponding diagonal elements, we obtain the (normalized) standard deviations of the estimated impulse response coefficients. These values are also printed to the screen (labeled norm. std.dev.). Note that the above procedure can be repeated using different stimulus functions. This allows a comparative analysis of the IRF coefficient estimation accuracy for different experimental designs, under the assumption that $MSE$ will be constant for a given voxel.

We have seen that, using the above experimental design, it is not possible to estimate the system IRF for a maxlag of 4; however, it is possible to estimate the IRF for a maxlag of 3 (or less). Question: If it is required that the "On" period and the "Off" period be equal, can you devise a block type design that allows estimation of the IRF for a maxlag of 4 (or greater)? Verify your answer using 3dDeconvolve with the -nodata option.

**Example 1.4.1.2 Evaluation of a Random Design**

In place of the above "block" type design, we will now investigate the use of a "random" design. Toss a coin 60 times; for each coin toss, record a 1 if "heads" occurs, and record a 0 if "tails" occurs. The record of the coin tosses might look something like this:

$$f(t) = \{ \begin{array}{cccccccccccccccccccc} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{array} \}$$

Save your record of coin tosses $f(t)$ as a single column of 0's and 1's into file Random.1D. Now, execute the following script:

Program 3dDeconvolve Command Line for Example 1.4.1.2

```
3dDeconvolve \
-nodata \
-nlast 59 \
-polort 0 \
-num_stimts 1 \
-stim_file 1 Random.1D \
-stim_label 1 ''Random'' \
-stim_maxlag 1 4
```

■

Do you obtain a result similar to the following?

Program 3dDeconvolve Screen Output for Example 1.4.1.2

```
Program:           3dDeconvolve
Author:            B. Douglas Ward
Initial Release:   02 Sept 1998
Latest Revision:   29 Jan 2002
```

```
(X'X) inverse matrix:
  0.1451  -0.0378  -0.0481  -0.0544  -0.0518  -0.0497
 -0.0378   0.0722   0.0026   0.0047   0.0003  -0.0044
 -0.0481   0.0026   0.0729   0.0042   0.0087   0.0039
 -0.0544   0.0047   0.0042   0.0745   0.0071   0.0115
 -0.0518   0.0003   0.0087   0.0071   0.0738   0.0070
 -0.0497  -0.0044   0.0039   0.0115   0.0070   0.0745

Stimulus:  Random
 h[0]  norm.  std.  dev.  =  0.2686
 h[1]  norm.  std.  dev.  =  0.2700
 h[2]  norm.  std.  dev.  =  0.2730
 h[3]  norm.  std.  dev.  =  0.2717
 h[4]  norm.  std.  dev.  =  0.2730
```

■

Repeat the analysis using maxlag = 8, 12, 16, ... . At what point does multicollinearity occur? What can you say about the norm. std. dev. values as the maxlag is increased?

Repeat the above analysis using a different sequence of coin tosses to generate the random binary sequence stimulus function. How do the parameter estimation accuracies compare with the first random binary sequence?

Extra Credit: Set maxlag = 0, and execute the script. Does the output from 3dDeconvolve indicate that the norm. std. dev. is approximately 0.2582? Can you provide a mathematical justification for this result? Of course, this result applies only for a random binary sequence of length N=60. What result would you expect for N = 32? For N = 128? Use 3dDeconvolve to verify your predictions. What is the functional relationship between parameter estimation accuracy and N for a binary random process, with maxlag = 0?

Extra Extra Credit: So far, we have only considered random binary sequences with Prob(1) = Prob(0) = $\frac{1}{2}$. Letting $p$ = Prob(1), investigate how the parameter estimation accuracy varies when you change $p$. For example, using a single die, or some other randomization method, generate a random binary sequence having $p = \frac{1}{3}$. For maxlag = 0, use 3dDeconvolve to calculate the norm. std. dev. for your random binary sequence. Derive the theoretical value for the accuracy of the parameter estimate as a function of $p$ and $N$.

For a continuation of this discussion concerning experimental design, and consideration of statistical power estimation, see Sections 3.3 and 3.4.

### 1.4.2 Estimation of the System Impulse Response Function

In this set of Examples, we consider application of program 3dDeconvolve for estimation of the system impulse response function (IRF).

**Example 1.4.2.1 Estimation of IRF from Non-Overlapping Responses; No Measurement Noise**

First, we will consider the case where the input stimuli are spaced sufficiently far apart in time that the hemodynamic responses to the individual stimuli do no overlap. To further simplify things, we will assume that there is no measurement noise.

Let the 0-4 point IRF be:

$$h(t) = \{\ 0 \quad 5 \quad 10 \quad 5 \quad 2\ \}$$

Let the 20 point stimulus function be:

$$f(t) = \{\ 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ \}$$

Save f(t) as a single column of numbers in file f.1D. To represent the system response, convolve the stimulus function with the IRF. The result is:

$$\begin{aligned} y(t) &= f(t) \otimes h(t) \\ &= \{\ 0 \quad 0 \quad 0 \quad 5 \quad 10 \quad 5 \quad 2 \quad 0 \quad 0 \quad 5 \quad 10 \quad 5 \quad 2 \quad 0 \quad 0 \quad 5 \quad 10 \quad 5 \quad 2 \quad 0\ \} \end{aligned}$$

If we model the measured response by adding a constant (100) plus slope (1), we get:

$$\begin{aligned} z(t) &= b_0 + b_1 t + y(t) \\ &= 100 + 1 \cdot t + y(t) \\ &= \begin{array}{lllllllllll} \{ & 100 & 101 & 102 & 108 & 114 & 110 & 108 & 107 & 108 & 114 \\ & 120 & 116 & 114 & 113 & 114 & 120 & 126 & 122 & 120 & 119 \quad \} \end{array} \end{aligned}$$

Save the above, as a single column of numbers, into file z.1D. Now, to estimate the IRF, execute the following script:

Program 3dDeconvolve Command Line for Example 1.4.2.1

```
3dDeconvolve \
-input1D z.1D   -num_stimts 1  \
-stim_file 1 f.1D   -stim_label 1 ''f''   -stim_maxlag 1 4
```

■

The program output is as shown below:

Program 3dDeconvolve Screen Output for Example 1.4.2.1

```
Program:              3dDeconvolve
Author:               B. Douglas Ward
Initial Release:      02 Sept 1998
Latest Revision:      29 Jan 2002

Baseline:
 t^0 coef =  100.0000    t^0 t-st =  1000.0000    p-value =  0.0000e+00
 t^1 coef =    1.0000    t^1 t-st =  1000.0000    p-value =  0.0000e+00
```

```
Stimulus:  f
 h[0] coef =   0.0000    h[0] t-st =     0.0000    p-value =  1.0000e+00
 h[1] coef =   5.0000    h[1] t-st =  1000.0000    p-value =  0.0000e+00
 h[2] coef =  10.0000    h[2] t-st =  1000.0000    p-value =  0.0000e+00
 h[3] coef =   5.0000    h[3] t-st =  1000.0000    p-value =  0.0000e+00
 h[4] coef =   2.0000    h[4] t-st =  1000.0000    p-value =  0.0000e+00
       R^2 =   1.0000       F[5,9] =  1000.0000    p-value =  0.0000e+00

Full Model:
 MSE =  0.0000
 R^2 =  1.0000    F[5,9] =  1000.0000    p-value =  0.0000e+00
```
■

Note that the t-stats and F-stats are limited to a maximum value of 1000.0. Also, note that the estimated full model parameter vector is exactly correct:

$$\begin{aligned}\mathbf{b}^t &= \{\ b_0 \quad b_1 \quad h[0] \quad h[1] \quad h[2] \quad h[3] \quad h[4]\ \} \\ &= \{\ 100.00 \quad 1.00 \quad 0.00 \quad 5.00 \quad 10.00 \quad 5.00 \quad 2.00\ \}\end{aligned}$$

which is not surprising, since no noise was present in the "measurement".

**Example 1.4.2.2 Estimation of IRF from Non-Overlapping Responses; Additive Noise**

To simulate the effect of measurement error, add white Gaussian noise to the above data. For example, adding values from a table of $N(\mu, \sigma^2) = N(0, 4)$ random variates (from Ref.[3]) to the above $z(t)$, we obtain:

$$\begin{aligned}&zn(t) = z(t) + \varepsilon(t) \\ &= \{\ 99.78 \quad 100.46 \quad 101.30 \quad 113.51 \quad 111.60 \quad 109.01 \quad 107.84 \quad 106.42 \quad 106.11 \quad 114.85 \\ &\quad 117.55 \quad 113.18 \quad 114.58 \quad 111.93 \quad 115.01 \quad 121.21 \quad 128.23 \quad 125.22 \quad 123.75 \quad 120.28\ \}\end{aligned}$$

Save your "response+noise" data into file zn.1D. You can plot the "data" using the command: 1dplot zn.1D. Looking at the plot, can you visualize the shape of the IRF?

Now, execute the script from the previous Example, except replace -input1D z.1D with -input1D zn.1D. The program output is as shown below:

Program 3dDeconvolve Screen Output for Example 1.4.2.2

```
Program:            3dDeconvolve
Author:             B. Douglas Ward
Initial Release:    02 Sept 1998
Latest Revision:    29 Jan 2002

Baseline:
 t^0 coef =  95.9670    t^0 t-st =  69.1079    p-value =  1.4053e-13
 t^1 coef =   1.3007    t^1 t-st =  15.5897    p-value =  8.0672e-08
```

```
Stimulus:  f
 h[0] coef =   0.2848    h[0] t-st =   0.2062    p-value =  8.4121e-01
 h[1] coef =   6.4541    h[1] t-st =   4.6989    p-value =  1.1219e-03
 h[2] coef =  10.1522    h[2] t-st =   8.1118    p-value =  1.9809e-05
 h[3] coef =   5.5282    h[3] t-st =   4.4670    p-value =  1.5614e-03
 h[4] coef =   3.8141    h[4] t-st =   3.1032    p-value =  1.2658e-02
       R^2 =   0.9075       F[5,9] =  17.6576    p-value =  2.0485e-04

Full Model:
 MSE =  2.2556
 R^2 =  0.9075    F[5,9] =  17.6576    p-value =  2.0485e-04
```

■

Note that, since there is only one input stimulus function, the Partial R^2 and Partial F values (listed under Stimulus: f) are identical to the Full Model R^2 and F-stat, respectively. Also, note that the estimated full model parameter vector is now:

$$\mathbf{b}^t = \left\{\ 95.97 \quad 1.30 \quad 0.28 \quad 6.45 \quad 10.15 \quad 5.53 \quad 3.81\ \right\}$$

Due to the measurement noise, the estimated full model parameter vector is not exactly equal to the "true" parameter vector.

**Example 1.4.2.3 Estimation of IRF from Overlapping Responses; No Measurement Noise**

In the previous Examples, the input impulses were spaced sufficiently far apart in time that the system responses to the individual stimuli did not overlap. What happens when the system responses do overlap? Consider the following stimulus function:

$$g(t) = \left\{\ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \right\}$$

Save $g(t)$ as a single column of numbers in file g.1D. The stimulus function $g(t)$ contains 10 impulses. Assuming that the system is linear, the response to the sequence of impulses should be equal to the sum of the responses to the individual impulses. Thus, the system response $y(t) = g(t) \otimes h(t)$ can be calculated by adding up the individual impulse response functions:

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 5 | 10 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 5 | 10 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 5 | 10 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 10 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 10 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 10 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 10 | 5 | 2 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 10 | 5 | 2 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 10 | 5 | 2 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 10 | 5 |
| | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – | – |
| y(t) = | 0 | 5 | 15 | 15 | 7 | 7 | 15 | 20 | 17 | 7 | 7 | 10 | 5 | 7 | 10 | 10 | 17 | 20 | 17 | 7 |

Now, model the measured data as a constant (100) plus slope (1) plus the system response, e.g.:

$$\begin{aligned} w(t) &= b_0 + b_1 t + y(t) = 100 + 1 \cdot t + y(t) \\ &= \begin{Bmatrix} 100 & 106 & 117 & 118 & 111 & 112 & 121 & 127 & 125 & 116 \\ 117 & 121 & 117 & 120 & 124 & 125 & 133 & 137 & 135 & 126 \end{Bmatrix} \end{aligned}$$

and store this as a single column of numbers into file w.1D. You can plot the "data" using the command: 1dplot w.1D. Looking at the plot, can you visualize the shape of the IRF?

Execute the following script, which uses the $g(t)$ stimulus function, and the $w(t)$ "data".

Program 3dDeconvolve Command Line for Example 1.4.2.3

```
3dDeconvolve \
-input1D w.1D   -num_stimts 1  \
-stim_file 1 g.1D   -stim_label 1 ''g''   -stim_maxlag 1 4
```

■

Since no noise is present, the program output should agree with the "true" parameter vector:

$$\mathbf{b}^t = \{ \; 100.00 \quad 1.00 \quad 0.00 \quad 5.00 \quad 10.00 \quad 5.00 \quad 2.00 \; \}$$

**Example 1.4.2.4 Estimation of IRF from Overlapping Responses; Additive Noise**

Again simulate the effect of measurement error, by adding white Gaussian noise to the above data. So, adding random values (say from a table of $N(\mu, \sigma^2) = N(0, 4)$ random variates (see Ref.[3])) to the above $w(t)$, you obtain something that looks like:

$$\begin{aligned} &wn(t) = w(t) + \varepsilon(t) \\ &= \{ \; \begin{matrix} 99.78 & 105.46 & 116.30 & 123.51 & 108.60 & 111.01 & 120.84 & 126.42 & 123.11 & 116.85 \\ 114.55 & 118.18 & 117.58 & 118.93 & 125.01 & 126.21 & 135.23 & 140.22 & 138.75 & 127.28 \end{matrix} \; \} \end{aligned}$$

Save your "response+noise" data into file wn.1D. You can plot the "data" using the command: 1dplot wn.1D. Looking at the plot, can you visualize the shape of the IRF?

Now, execute the script from the previous Example, except replace -input1D w.1D with -input1D wn.1D. The program output, for the above data, is as shown below:

Program 3dDeconvolve Screen Output for Example 1.4.2.4

```
Program:            3dDeconvolve
Author:             B. Douglas Ward
Initial Release:    02 Sept 1998
Latest Revision:    29 Jan 2002
```

```
Baseline:
 t^0 coef =  92.6567    t^0 t-st =  77.2499    p-value =  5.1655e-14
 t^1 coef =   1.3345    t^1 t-st =  23.6341    p-value =  2.0731e-09

Stimulus:  g
 h[0] coef =   1.9530    h[0] t-st =    3.5183    p-value =  6.5325e-03
 h[1] coef =   6.0968    h[1] t-st =   11.2205    p-value =  1.3615e-06
 h[2] coef =  11.5062    h[2] t-st =   19.8937    p-value =  9.5163e-09
 h[3] coef =   6.6768    h[3] t-st =   11.9295    p-value =  8.0960e-07
 h[4] coef =   2.6870    h[4] t-st =    4.7401    p-value =  1.0587e-03
       R^2 =   0.9835       F[5,9] =  107.3899    p-value =  9.6139e-08

Full Model:
 MSE =  0.9618
 R^2 =  0.9835    F[5,9] =  107.3899    p-value =  9.6139e-08
```

■

Note that, since there is only one input stimulus function, the Partial R^2 and Partial F values (listed under Stimulus: g) are identical to the Full Model R^2 and F-stat, respectively.

Also, note that the estimated full model parameter vector is now:

$$\mathbf{b}^t = \{\ 92.66 \quad 1.33 \quad 1.95 \quad 6.10 \quad 11.51 \quad 6.68 \quad 2.69\ \}$$

Due to the measurement noise, the estimated full model parameter vector is not exactly equal to the "true" parameter vector.

**Example 1.4.2.5 Estimation of IRF; 3d+time Dataset**

A researcher wants to estimate the system impulse response function for a particular type of visual stimulus. The magnitude of the input stimulus function, as a function of time, is recorded in file Visual.1D. The measured FMRI data is contained in the 3d+time dataset Paula+orig (.HEAD and .BRIK). In order to estimate the impulse response function, at each voxel location, Program 3dDeconvolve can be executed with the following batch commands:

Program 3dDeconvolve Batch Command File for Example 1.4.2.5

```
3dDeconvolve \
-input Paula+orig \
-polort 1   -rmsmin 1.0   -progress 1000 \
-num_stimts 1 \
-stim_file 1 Visual.1D   -stim_label 1 Visual \
-stim_minlag 1 0   -stim_maxlag 1 10 \
-iresp 1 Paula.irf \
-fitts Paula.fit \
-fout -rout -tout   -bucket Paula.bucket
```

■

The first batch command specifies that the input 3d+time dataset is to be read from file Paula+orig (.HEAD and .BRIK). The command -polort 1 indicates that the baseline model should include a constant offset plus linear drift terms (this is the default condition). The command -rmsmin 1.0 is used to screen out voxels showing little variation (this can be used to exclude voxels located outside the brain). The command -progress 1000 is used to specify that, during execution of the program, screen output is generated for every 1000th voxel.

The number of input stimulus files is specified with the command -num_stimts 1. The name of the stimulus time series file is entered with the command -stim_file 1 Visual.1D. The command -stim_label 1 Visual indicates that the stimulus function will be identified with the label "Visual" in the program output.

The next two commands, -stim_minlag 1 0 and -stim_maxlag 1 10 specify that the impulse response function is to be estimated for time lags between 0 and 10 (i.e., for $\tau = 0$ TR, 1 TR, 2 TR, ..., 10 TR).

The remaining commands specify the program output. The command -iresp 1 Paula.irf generates the 3d+time dataset Paula.irf+orig (.HEAD and .BRIK), which contains the estimated impulse response function for each voxel. The command -fitts Paula.fit generates the 3d+time dataset Paula.fit+orig (.HEAD and .BRIK), which contains the full model fitted time series for each voxel. The -fout, -rout, and -tout commands indicate that the F-statistics, $R^2$, and t-statistics are to be included in the bucket dataset output. Finally, the command -bucket Paula.bucket is used to generate the "bucket" type dataset Paula.bucket+orig (.HEAD and .BRIK) containing the parameter estimates, corresponding t-statistics, and the F-statistic and $R^2$ for significance of the regression.

After program 3dDeconvolve has finished execution, program afni can be used to view the output files. The format for the bucket dataset Paula.bucket is illustrated below.

| Brick # | Label | Contents |
|---|---|---|
| 0 | Base t^0 Coef | least squares est. of $b_0$(constant term) |
| 1 | Base t^0 t-st | t-statistic for $b_0 = b_0/s(b_0)$ |
| 2 | Base t^1 Coef | least squares est. of $b_1$(linear trend) |
| 3 | Base t^1 t-st | t-statistic for $b_1 = b_1/s(b_1)$ |
| 4 | Visual[0] Coef | least squares est. of $h_0$ (impulse response at time lag 0) |
| 5 | Visual[0] t-st | t-statistic for $h_0 = h_0/s(h_0)$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| 24 | Visual[10] Coef | least squares est of $h_{10}$ (impulse response at time lag 10) |
| 25 | Visual[10] t-st | t-statistic for $h_{10}=h_{10}/s(h_{10})$ |
| 26 | Visual R^2 | $R$^2 for significance of the Visual stimulus |
| 27 | Visual F-stat | F-statistic for significance of Visual stimulus |
| 28 | Full R^2 | Coefficient of multiple determination $R^2$ |
| 29 | Full F-stat | F-statistic for significance of the overall regression |

Note that the output of this program can be used as input to other programs, such as 3dANOVA, for comparing results across subjects, runs, experimental conditions, etc. For this purpose, it would be desirable to reduce the above output to a single number per voxel. One possible way to accomplish this is to use the -glt options, as illustrated in Section 1.4.4.

### 1.4.3 Multiple Linear Regression

In this section, we consider the multiple linear regression problem. Actually, we have already performed multiple linear regression when calculating the impulse response function for multiple time lags. However, we will use "multiple linear regression" to refer more specifically to those experiments involving "multiple stimulus functions".

Another point about the notation: In *AFNI* fim / 3dfim+, a distinction is made between "ort" functions and "ideal" functions. That is, the "ideal" function is the input for which we are interested in determining the system response; the "ort" functions represent inputs that are not of inherent interest, and which only make it more difficult to determine the true system response. However, in 3dDeconvolve, no such distinction is made; in fact, both types of functions are treated equally and are referred to as "stimulus" functions. Statistics are calculated for all stimulus functions, and it is only the user's interpretation of the results which serves to distinguish between "ort" and "ideal".

**Example 1.4.3.1 Analysis of Extracted time series**

In this and the following Example, we analyze a time series extracted from a 3d+time dataset. We will use the v2:time+orig sample dataset which is included with the *AFNI* distribution. Of course, feel free to use your own 3d+time dataset.

First, use afni to display the v2:time+orig dataset. Display the time series graphs, and use the Write Center option from the graphing menu to write out the time series corresponding to the voxel with coordinates (29,29,8). The time series corresponding to this particular voxel is then written to file 029_029_008.1D. You can examine the contents of this file with a text editor, or use

```
1dplot 029_029_008.1D
```

to display a graph of this particular time series.

Also included with the *AFNI* distribution is file cos7.00.1D. This contains regularly-spaced values of the cosine function. We will use this as the input stimulus function.

Program 3dDeconvolve Command Line for Example 1.4.3.1

```
3dDeconvolve \
-input1D 029_029_008.1D \
-nfirst 6 \
-polort 1
-num_stimts 1 \
-stim_file 1 cos7.00.1D   -stim_label 1 ''Cosine''   -stim_maxlag 1 1  \
-fitts myFit
```

■

A couple of observations about the above script. The -nfirst 6 command indicates that the first 6 data points are to be discarded. This is necessitated by the nature of the cos7.00.1D file. If you inspect this file, you will see that the first 4 values are 100000. This code is used to tell programs afni fim / 3dfim+ to discard the first 4 data points. As explained earlier, program 3dDeconvolve does not use this code for discarding data points (instead, it uses as separate "censor" file). However, since 3dDeconvolve uses time-lagged versions of the input stimulus function, it is necessary to discard not only the first 4 points, but some additional points as well, to keep the "100000" values from contaminating the results.

Note that the -stim_maxlag 1 1 command indicates that only two time lags, 0 and 1, are considered. However, this is sufficient to accurately model all responses of the form:

$$y(t) = A\cos(\omega(t-\tau))$$

(Can you prove this claim?)

The command -fitts myFit causes the fitted time series to be written to file myFit.1D. You can plot the input time series and the fitted time series in separate graph windows with the following commands (suppressing the first 6 points):

```
1dplot  -ignore 6  029_029_008.1D
1dplot  -ignore 6  myFit.1D
```

However, if you wish to plot the fitted time series on top of the FMRI time series data, the following commands can be used to combine the two time series into a single file (e.g., Overlay.1D) for plotting:

Script for Plotting Fitted Data on top of Actual Data

```
1dcat 029_029_008.1D  myFit.1D  >  Overlay.1D
1dplot  -ignore 6  -one Overlay.1D
```

■

You should see the fitted time series (in red) plotted on top of the original time series (in black). This result is similar to that which you will obtain in Example 2.3.1 (q.v.) using the Deconvolution plugin.

**Example 1.4.3.2 Multiple Linear Regression using Estimated Motion Parameters**

In this Example, we repeat the above analysis, but will include in the baseline model the estimated motion parameters. The reason for doing so is to account for variation in the measured response due to subject motion. This should help both to reduce false positives (by subtracting out the effect of stimulus correlated motion) and to increase true positives (by reducing the measurement noise estimate).

The first step is to estimate the subject motion. This can be done using program 3dvolreg, e.g.:

```
3dvolreg -base 34 -1Dfile v2.motion.1D v2:time+orig
```

This will write to file v2.motion.1D the estimated motion parameters: roll, pitch, yaw, dS. dL, and dP, as 6 columns of numbers. Now, these separate columns can be entered as individual stimulus functions, as indicated below:

Program 3dDeconvolve Command Line for Example 1.4.3.2

```
3dDeconvolve \
-input1D 029_029_008.1D \
-nfirst 6 \
-polort 1
-num_stimts 7 \
-stim_file 1 cos7.00.1D   -stim_label 1 Cosine   -stim_maxlag 1 1 \
-stim_file 2 'v2.motion.1D[0]'   -stim_base 2   -stim_label 2 Roll \
-stim_file 3 'v2.motion.1D[1]'   -stim_base 3   -stim_label 3 Pitch \
-stim_file 4 'v2.motion.1D[2]'   -stim_base 4   -stim_label 4 Yaw \
-stim_file 5 'v2.motion.1D[3]'   -stim_base 5   -stim_label 5 dS \
-stim_file 6 'v2.motion.1D[4]'   -stim_base 6   -stim_label 6 dL \
-stim_file 7 'v2.motion.1D[5]'   -stim_base 7   -stim_label 7 dP
```

■

If you execute the above script, you should see a result similar to that of: Deconvolution Plugin Screen Output for Example 2.3.2 (q.v.).

From the output, we see that the p-value for the Cosine stimulus is $9.2*10^{-8}$, which is the same as the p-value for the Full Model. This is because the -stim_base command was used to indicate that each of the motion parameter stimulus functions belongs to the Baseline Model. Therefore, the p-value for the Full Model does *not* include the contributions of the estimated motion parameters in explaining the response, since this is not relevant to the matter being investigated.

From the previous Example, the p-value for the Cosine stimulus, without motion parameters in the model, was $3.1*10^{-6}$. Therefore, at least in this example, inclusion of the motion parameters has greatly increased the statistical significance of the result (i.e., reduced the p-value). This can be explained by looking at the sample error variance ($MSE$). Without the motion parameters, $MSE = 76.60$; with the motion parameters, $MSE = 46.99$. That is, by including the motion parameters in the baseline model, variation in the measured response due to motion is subtracted out; this reduces the estimate for the error variance, and hence increases the statistical significance of the Cosine stimulus in explaining the response.

**Exercise:** Add the -fitts command to the above command line in order to generate the fitted time series. Plot the fitted time series on top of the actual data, as was done in Example 1.4.3.1.

**Exercise 1.4.3.3 Comparison of Multiple Regression and Cross Correlation**

Repeat Examples 1.4.3.1 and 1.4.3.2, but change the script files to process the entire 3d+time dataset. Using afni, compare the distribution of "active" voxels found by program 3dDeconvolve, with and without the estimated motion parameter inputs. Be sure to threshold using the partial F-statistic for the Cosine stimulus function. (Question: Why not use the full F-statistic for thresholding?)

Repeat the above analysis using program 3dfim+. In this case, use the cosall.1D file as the "ideal" function input. Process the 3d+time dataset, with and without using v2.motion.1D as the "ort" function input. Use afni to compare the 3dfim+ output obtained with and without the motion parameters. Also, compare the 3dDeconvolve output parametric maps with the 3dfim+ parametric maps.

What general conclusions do you reach?

**Example 1.4.3.4 Multiple Experimental Input Stimuli; No Noise**

A researcher wants to perform a multiple linear regression on FMRI time series data to distinguish different regions of neural activation based upon differences in linguistic processing. In this experiment, the subject is shown a sequence of "words" that fall into one of three categories: 1) "Random": the letters which make up the "word" are selected at random, but with the same marginal probabilities as occur in English language text; 2) "Markov": the letters which make up the "word" are chosen with the same 1st order Markov probabilities that are found in English; and 3) "English": actual English words. Letting

$$\begin{aligned} \text{X} &= \text{No input} \\ \text{R} &= \text{Random input} \\ \text{M} &= \text{Markov input} \\ \text{E} &= \text{English input} \end{aligned}$$

the sequence of experimental inputs is given by:

X R M E X M X E R X R E M X X X M E R X

In this example, there are three separate stimulus time series functions, corresponding to the times of presentation for the three different word categories.

$$\begin{aligned} r(t) &= \{\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ \} \\ m(t) &= \{\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ \} \\ e(t) &= \{\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ \} \end{aligned}$$

Store these rows as single columns into files Random.1D, Markov.1D, and English.1D, respectively.

Since the researcher is not interested in the impulse response function per se, but is interested in decomposing the output as a linear function of the different input stimuli, only a few time lags will be used for estimating each individual input stimulus response. These time lags are used to allow for differing rates of response to the stimulus in different voxels.

Let the 0-2 point IRF's be:

$$\begin{aligned} h_R(t) &= \{ \ 2 \ \ 7 \ \ 5 \ \} \\ h_M(t) &= \{ \ 1 \ \ 4 \ \ 6 \ \} \\ h_E(t) &= \{ \ 3 \ \ 9 \ \ 2 \ \} \end{aligned}$$

Verify that the result of convolving these IRF's with the individual stimulus functions, and adding the results, is the following:

$$\begin{aligned} y(t) &= r(t) \otimes h_R(t) + m(t) \otimes h_M(t) + e(t) \otimes h_E(t) \\ &= \{ \ 0 \ \ 2 \ \ 8 \ \ 12 \ \ 15 \ \ 3 \ \ 4 \ \ 9 \ \ 11 \ \ 9 \ \ 7 \ \ 10 \ \ 15 \ \ 6 \ \ 6 \ \ 0 \ \ 1 \ \ 7 \ \ 17 \ \ 9 \ \} \end{aligned}$$

Now, model the measured data as a constant (100) plus slope (1) plus the system response, e.g.:

$$\begin{aligned} z(t) &= b_0 + b_1 t + y(t) = 100 + 1 \cdot t + y(t) \\ &= \begin{matrix} \{ & 100 & 103 & 110 & 115 & 119 & 108 & 110 & 116 & 119 & 118 & \\ & 117 & 121 & 127 & 119 & 120 & 115 & 117 & 124 & 135 & 128 & \} \end{matrix} \end{aligned}$$

and store this as a single column of numbers into file Ling.1D.

The parameter vector to be estimated has the following form:

$$\mathbf{b}^t = \{ \ b_0 \ \ b_1 \ \ h_R[0] \ \ h_R[1] \ \ h_R[2] \ \ h_M[0] \ \ h_M[1] \ \ h_M[2] \ \ h_E[0] \ \ h_E[1] \ \ h_E[2] \ \}$$

A sample batch command file for executing Program 3dDeconvolve for this example is presented below.

Program 3dDeconvolve Command Line for Example 1.4.3.4

```
3dDeconvolve \
-input1D Ling.1D \
-num_stimts 3 \
-stim_file 1 Random.1D   -stim_label 1 ''Random''   -stim_maxlag 1 2 \
-stim_file 2 Markov.1D   -stim_label 2 ''Markov''   -stim_maxlag 2 2 \
-stim_file 3 English.1D  -stim_label 3 ''English''  -stim_maxlag 3 2
```

■

Since there is no "noise", the estimated full model parameter vector is exactly correct:

$$\mathbf{b}^t = \{ \ 100.0 \ \ 1.0 \ \ 2.0 \ \ 7.0 \ \ 5.0 \ \ 1.0 \ \ 4.0 \ \ 6.0 \ \ 3.0 \ \ 9.0 \ \ 2.0 \ \}$$

**Example 1.4.3.5 Multiple Experimental Input Stimuli; Additive Noise**

Again simulate the effect of measurement error, by adding white Gaussian noise to the above data. So, adding random values (say from a table of $N(\mu, \sigma^2) = N(0, 1)$ random variates (see Ref.[3])) to the above $z(t)$, you obtain something that looks like:

$zn(t) = z(t) + \varepsilon(t)$

$$= \{ \begin{array}{cccccccccc} 100.46 & 103.14 & 112.46 & 114.68 & 118.93 & 108.30 & 109.71 & 117.30 & 119.24 & 117.04 \\ 117.06 & 118.47 & 126.47 & 118.81 & 120.54 & 113.44 & 117.19 & 122.81 & 135.02 & 128.52 \end{array} \}$$

Save your "response+noise" data into file LingNoise.1D.

Now, execute the script from the previous example, except replace -input1D Ling.1D with -input1D LingNoise.1D. The program output, for the above data, is as shown below:

Program 3dDeconvolve Screen Output for Example 1.4.3.5

```
Program:            3dDeconvolve
Author:             B. Douglas Ward
Initial Release:    02 Sept 1998
Latest Revision:    29 Jan 2002

Baseline:
 t^0 coef =  99.3593    t^0 t-st =  95.0398    p-value =  3.7617e-12
 t^1 coef =   0.9435    t^1 t-st =  18.5667    p-value =  3.2618e-07

Stimulus:  Random
 h[0] coef =  3.4230    h[0] t-st =   3.6685    p-value =  7.9804e-03
 h[1] coef =  7.7680    h[1] t-st =   9.1181    p-value =  3.9187e-05
 h[2] coef =  5.0313    h[2] t-st =   6.3798    p-value =  3.7442e-04
       R^2 =  0.9392       F[3,7] =  36.0613    p-value =  1.2574e-04

Stimulus:  Markov
 h[0] coef =  2.7658    h[0] t-st =   3.2833    p-value =  1.3427e-02
 h[1] coef =  5.0166    h[1] t-st =   5.4020    p-value =  1.0064e-03
 h[2] coef =  8.0361    h[2] t-st =   8.8991    p-value =  4.5900e-05
       R^2 =  0.9214       F[3,7] =  27.3355    p-value =  3.0773e-04

Stimulus:  English
 h[0] coef =  2.2758    h[0] t-st =   2.9019    p-value =  2.2925e-02
 h[1] coef =  7.9706    h[1] t-st =  10.2192    p-value =  1.8541e-05
 h[2] coef =  2.1289    h[2] t-st =   2.8398    p-value =  2.5051e-02
       R^2 =  0.9383       F[3,7] =  35.4904    p-value =  1.3246e-04

Full Model:
 MSE =  1.0943
 R^2 =  0.9802    F[9,7] =  38.4744    p-value =  3.8639e-05
```

■

**Example 1.4.3.6 Multiple Experimental Input Stimuli; 3d+time Dataset**

We complete the previous example by displaying the batch command file necessary for processing an entire 3d+time dataset. Here, it is assumed the the input data is contained in file Monica+orig.

Program 3dDeconvolve Batch Command File for Example 1.4.3.6

```
3dDeconvolve \
-input Monica+orig \
-polort 1   -rmsmin 1.0   -progress 1000 \
-num_stimts 3 \
-stim_file 1 Random.1D  -stim_file 2 Markov.1D  -stim_file 3 English.1D \
-stim_label 1 Random    -stim_label 2 Markov    -stim_label 3 English \
-stim_minlag 1 2        -stim_minlag 2 2        -stim_minlag 3 2 \
-stim_maxlag 1 5        -stim_maxlag 2 5        -stim_maxlag 3 5 \
-fout   -bucket Monica.bucket \
-errts Monica.err
```

■

The first batch command specifies that the input 3d+time dataset is to be read from file Monica+orig (.HEAD and .BRIK). The command -polort 1 indicates that the baseline model should include a constant offset plus linear drift terms (this is the default condition). The command -rmsmin 1.0 is used to screen out voxels showing little variation (this can be used to exclude voxels located outside the brain). Alternatively, one could use the -mask mname command. The command -progress 1000 is used to specify that, during execution of the program, screen output is generated for every 1000th voxel.

The number of input stimulus files is specified with the command -num_stimts 3. The names of the three stimulus time series files are specified with the commands -stim_file 1 Random.1D, -stim_file 2 Markov.1D, and -stim_file 3 English.1D. The following three -stim_label commands indicate that the stimulus functions will be identified with the labels Random, Markov, and English, respectively, in the program output.

The following -stim_minlag and -stim_maxlag commands specify that the system response to the different input stimuli is to be modeled by time delayed versions of the input stimulus functions, for time lags 2, 3, 4, and 5 (i.e., for $\tau = 2$ TR, 3 TR, 4 TR, and 5 TR).

The command -bucket Monica.bucket is used to generate the "bucket" type dataset Monica.bucket+orig (.HEAD and .BRIK) containing the parameter estimates, partial F-statistics, and the F-statistic for significance of the regression. Note that the content of the bucket dataset is specified by the -fout command Finally, the command -errts Monica.err indicates that the residual errors from fitting the full model to the time series data should be written to the 3d+time dataset Monica.err+orig.

After program 3dDeconvolve has finished execution, program afni can be used to view the output files. The format for the bucket dataset is illustrated below.

| Brick | Label | | Brick | Label |
|---|---|---|---|---|
| 0 | Base t^0 Coef | | 9 | Markov[4] Coef |
| 1 | Base t^1 Coef | | 10 | Markov[5] Coef |
| 2 | Random[2] Coef | | 11 | Markov F-stat |
| 3 | Random[3] Coef | | 12 | English[2] Coef |
| 4 | Random[4] Coef | | 13 | English[3] Coef |
| 5 | Random[5] Coef | | 14 | English[4] Coef |
| 6 | Random F-stat | | 15 | English[5] Coef |
| 7 | Markov[2] Coef | | 16 | English F-stat |
| 8 | Markov[3] Coef | | 17 | Full F-stat |

### 1.4.4 General Linear Tests

We continue consideration of Example 1.4.3.5. Recall that the full model parameter vector was represented by:

$$\mathbf{b}^t = \left\{ \begin{array}{ccccccccccc} b_0 & b_1 & h_R[0] & h_R[1] & h_R[2] & h_M[0] & h_M[1] & h_M[2] & h_E[0] & h_E[1] & h_E[2] \end{array} \right\}$$

**Example 1.4.4.1 Test of a single regression parameter**
Suppose that we wish to test whether there is a significant response to input stimulus $m(t)$ at time lag 1 (TR). As a test of hypotheses, this can be expressed as:

$$\begin{array}{rrl} & H_o : & h_M[1] = 0 \\ \text{vs.} & H_a : & h_M[1] \neq 0 \end{array}$$

In matrix terms, the null hypothesis is specified by $H_o : \mathbf{C}\boldsymbol{\beta} = \mathbf{0}$, where

$$\mathbf{C} = \left[ \begin{array}{ccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \right].$$

Since the null hypothesis involves only $s = 1$ linear constraint, matrix $\mathbf{C}$ is actually a row vector. Parameter $h_M[1]$ is the 7th parameter in the list of parameters for the full model, so matrix $\mathbf{C}$ consists of all 0's, except for a single 1 in the 7th position.

Store the above matrix (as a row) into file `glt1.mat`, and execute the following script:

Program 3dDeconvolve Command Line for Example 1.4.4.1

```
3dDeconvolve \
-input1D LingNoise.1D \
-num_stimts 3 \
```

```
-stim_file 1 Random.1D   -stim_label 1 ''Random''   -stim_maxlag 1 2 \
-stim_file 2 Markov.1D   -stim_label 2 ''Markov''   -stim_maxlag 2 2 \
-stim_file 3 English.1D  -stim_label 3 ''English''  -stim_maxlag 3 2 \
-glt 1 glt1.mat   -glt_label 1 ''h[1] Markov''
```

■

The output is the same as for Example 1.4.3.5, but with the additional GLT output:

```
 General Linear Test:  h[1] Markov
  LC[0] coef =  5.0166    LC[0] t-st =   5.4020    p-value =  1.0064e-03
         R^2 =  0.8065        F[1,7] =  29.1811    p-value =  1.0064e-03
```

■

It is important to note that $LC[0] = 5.0166$ is equal to $h_M[1]$, the estimated lag-1 coefficient for the Markov stimulus. Also, note that $F[1,7] = 29.1811$ is the square of the calculated $t-$stat for $h_M[1]$. (Can you explain why?)

Note: This example is for illustrative purposes only. Since the above test is equivalent to the $t-$test for significance of an individual parameter, this GLT is unnecessary.

**Example 1.4.4.2 Test of a single stimulus function**

The previous example showed how the GLT can be used to test for significance of a single stimulus at a single time lag. To test for significance of a single stimulus at multiple time lags, e.g., significance of $h_M(t)$, one would perform the test:

$$\begin{array}{rl} H_o: & h_M[0] = h_M[1] = h_M[2] = 0 \\ \text{vs.} \quad H_a: & \text{not all } h_M[i] = 0, \quad i = 0, 1, 2. \end{array}$$

In matrix terms, the null hypothesis is specified by $H_o : \mathbf{C}\boldsymbol{\beta} = \mathbf{0}$, where

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The $\mathbf{C}$ matrix contains a single 1 in each column corresponding to the $m(t)$ input stimulus. Note that the $\mathbf{C}$ matrix has 3 rows, since the GLT involves $s = 3$ linear constraints on the parameters.

Store the above matrix as 3 rows into file glt2.mat, and execute the script from the previous example, except replace the -glt command line with:

$$-\texttt{glt 3 glt2.mat} \quad -\texttt{glt\_label 1 "Markov"}$$

The output is the same as for Example 1.4.3.5, but with the additional GLT output:

```
General Linear Test:  Markov
 LC[0] coef =  2.7658    LC[0] t-st =   3.2833    p-value =  1.3427e-02
 LC[1] coef =  5.0166    LC[1] t-st =   5.4020    p-value =  1.0064e-03
 LC[2] coef =  8.0361    LC[2] t-st =   8.8991    p-value =  4.5900e-05
        R^2 =  0.9214        F[3,7] =  27.3355    p-value =  3.0773e-04
```

■

Compare the above output for the GLT with the output for the "Markov" stimulus.

Note: This example is for illustrative purposes only. Since the above test is equivalent to the partial $F-$test for significance of an individual stimulus, this GLT is unnecessary.

**Example 1.4.4.3 Comparison of two different regression parameters**
Suppose we wish to test whether the response to stimulus $r(t)$ at time lag 1 is equal to the response to stimulus $e(t)$ at time lag 1. As a test of hypotheses, this can be stated:

$$\begin{array}{rl} & H_o: \;\; h_R[1] = h_E[1] \\ \text{vs.} & H_a: \;\; h_R[1] = h_E[1] \end{array} \qquad \text{or} \qquad \begin{array}{rl} & H_o: \;\; h_R[1] - h_E[1] = 0 \\ \text{vs.} & H_a: \;\; h_R[1] - h_E[1] \neq 0 \end{array}$$

In matrix terms, the null hypothesis is specified by $H_o : \mathbf{C}\boldsymbol{\beta} = \mathbf{0}$, where

$$\mathbf{C} = \left[ \begin{array}{ccccccccccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \end{array} \right]$$

Note how the elements of the $\mathbf{C}$ matrix correspond to the coefficients of the linear constraint which corresponds to the null hypothesis.

Store the above matrix as 1 row into file glt3.mat, and execute the script for the previous example, except replace the -glt command line with:

```
-glt 1 glt3.mat   - glt_label 1 "Difference"
```

The output is the same as for Example 1.4.3.5, but with the additional GLT output:

```
General Linear Test:  Difference
 LC[0] coef =  -0.2026    LC[0] t-st =  -0.1775    p-value =  8.6417e-01
        R^2 =   0.0045        F[1,7] =   0.0315    p-value =  8.6417e-01
```

■

Verify that $LC[0] = h_R[1] - h_E[1]$.

**Example 1.4.4.4 Comparison of two different impulse response functions**
To test whether the response to stimulus $r(t)$ and to stimulus $e(t)$ are equal at all time lags, i.e., to test

$$\begin{array}{rl} & H_o: \;\; h_R[i] = h_E[i], \quad i = 0, 1, 2 \\ \text{vs.} & H_a: \;\; \text{not all } h_R[i] = h_E[i]. \end{array}$$

The null hypothesis can also be written:

$$H_o : \begin{cases} h_R[0] - h_E[0] = 0, \text{ and} \\ h_R[1] - h_E[1] = 0, \text{ and} \\ h_R[2] - h_E[2] = 0 \end{cases}$$

so we would use the following GLT matrix:

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Store the above matrix as 3 rows into file glt4.mat, and execute the script for the previous example, except replace the -glt command line with:

```
-glt 3 glt4.mat   - glt_label 1 "Random - English"
```

The output is the same as for Example 1.4.3.5, but with the additional GLT output:

```
General Linear Test:  Random-English
 LC[0] coef =   1.1473    LC[0] t-st =   1.0466    p-value =  3.3008e-01
 LC[1] coef =  -0.2026    LC[1] t-st =  -0.1775    p-value =  8.6417e-01
 LC[2] coef =   2.9024    LC[2] t-st =   2.8088    p-value =  2.6191e-02
        R^2 =   0.6514        F[3,7] =   4.3598    p-value =  4.9681e-02
```

■

Verify that $LC[0] = h_R[0] - h_E[0]$, $LC[1] = h_R[1] - h_E[1]$, and $LC[2] = h_R[2] - h_E[2]$.

**Example 1.4.4.5 Comparison of two different impulse response magnitudes**
The previous example shows how to test whether the responses to two different stimuli differ at any time lag. Of more direct interest may be whether the responses differ in "magnitude". Here, we will use "area under the curve" as an indication of response magnitude, and will approximate "area under the curve" by summing the impulse response function parameters over all time lags. In this case, the hypotheses under consideration are:

$$\begin{array}{rl} H_o : & h_R[0] + h_R[1] + h_R[2] = h_E[0] + h_E[1] + h_E[2] \\ \text{vs. } H_a : & h_R[0] + h_R[1] + h_R[2] \neq h_E[0] + h_E[1] + h_E[2] \end{array}$$

In matrix terms, the null hypothesis is specified by $H_o : \mathbf{C}\boldsymbol{\beta} = \mathbf{0}$, where

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & -1 \end{bmatrix}$$

Note that this GLT involves only 1 linear constraint, whereas the GLT in the previous example used 3 linear constraints.

Store the above matrix as 1 row into file glt5.mat, and execute the script for the previous example, except replace the -glt command line with:

$$-\texttt{glt 1 glt5.mat} \quad -\texttt{glt\_label 1 "Random} - \texttt{English"}$$

The output is the same as for Example 1.4.3.5, but with the additional GLT output:

```
General Linear Test:  Random-English
 LC[0] coef =  3.8471    LC[0] t-st =  1.5420    p-value =  1.6697e-01
       R^2 =  0.2536         F[1,7] =  2.3779    p-value =  1.6697e-01
```

■

Verify that $LC[0] = h_R[0] + h_R[1] + h_R[2] - h_E[0] - h_E[1] - h_E[2]$.

**Example 1.4.4.6 Calculation of GLT for a 3d+time dataset**

In this example, we consider 3 different experimental conditions (A, B, and C). For each of the 3 conditions, the IRF is to be estimated at 5 time points (0 TR through 4 TR). The individual impulse response functions are then to be compared with each other (A - B and A - C).

The full parameter vector has 17 elements:

$$\boldsymbol{\beta}^t = (\beta_0, \beta_1, \gamma_{a,0}, \gamma_{a,1}, \gamma_{a,2}, \gamma_{a,3}, \gamma_{a,4}, \gamma_{b,0}, \gamma_{b,1}, \gamma_{b,2}, \gamma_{b,3}, \gamma_{b,4}, \gamma_{c,0}, \gamma_{c,1}, \gamma_{c,2}, \gamma_{c,3}, \gamma_{c,4})$$

and the null hypotheses are specified by:

$$H_{o,1} : \begin{cases} \gamma_{a,0} = \gamma_{b,0}, \\ \gamma_{a,1} = \gamma_{b,1}, \\ \gamma_{a,2} = \gamma_{b,2}, \\ \gamma_{a,3} = \gamma_{b,3}, \\ \gamma_{a,4} = \gamma_{b,4}, \end{cases} \quad \text{and} \quad H_{o,2} : \begin{cases} \gamma_{a,0} = \gamma_{c,0}, \\ \gamma_{a,1} = \gamma_{c,1}, \\ \gamma_{a,2} = \gamma_{c,2}, \\ \gamma_{a,3} = \gamma_{c,3}, \\ \gamma_{a,4} = \gamma_{c,4}, \end{cases}$$

In this case, there are 2 GLT's. Each GLT is determined by a matrix (with $s = 5$ rows and $P = 17$ columns), which must be stored in the user specified files:

Contents of file DiffAB.mat:

$$\begin{array}{ccccccccccccccccc}
0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Contents of file DiffAC.mat:

$$\begin{array}{ccccccccccccccccc}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1
\end{array}$$

The following batch command file can be used to conduct the general linear tests (GLT's) for testing these two hypotheses.

Program 3dDeconvolve Batch Command File for Example 1.4.4.6

```
3dDeconvolve \
-input Subj307+orig \
-polort 1   -rmsmin 1.0   -progress 1000 \
-num_stimts 3 \
-stim_file 1 CondA.1D   -stim_label 1 CondA   -stim_maxlag 1 4 \
-stim_file 2 CondB.1D   -stim_label 2 CondB   -stim_maxlag 2 4 \
-stim_file 3 CondC.1D   -stim_label 3 CondC   -stim_maxlag 3 4 \
-glt 5 DiffAB.mat   -glt_label 1 ''A-B'' \
-glt 5 DiffAC.mat   -glt_label 2 ''A-C'' \
-fout   -bucket Subj307.buck
```

■

After program 3dDeconvolve has finished execution, program afni can be used to view the output files. The format for the bucket dataset Subj307.buck+orig is illustrated below.

| Brick | Label | | Brick | Label |
|---|---|---|---|---|
| 0 | Base t^0 Coef | | 17 | CondC[3] Coef |
| 1 | Base t^1 Coef | | 18 | CondC[4] Coef |
| 2 | CondA[0] Coef | | 19 | CondC F-stat |
| 3 | CondA[1] Coef | | 20 | A-B LC[0] |
| 4 | CondA[2] Coef | | 21 | A-B LC[1] |
| 5 | CondA[3] Coef | | 22 | A-B LC[2] |
| 6 | CondA[4] Coef | | 23 | A-B LC[3] |
| 7 | CondA F-stat | | 24 | A-B LC[4] |
| 8 | CondB[0] Coef | | 25 | A-B F-stat |
| 9 | CondB[1] Coef | | 26 | A-C LC[0] |
| 10 | CondB[2] Coef | | 27 | A-C LC[1] |
| 11 | CondB[3] Coef | | 28 | A-C LC[2] |
| 12 | CondB[4] Coef | | 29 | A-C LC[3] |
| 13 | CondB F-stat | | 30 | A-C LC[4] |
| 14 | CondC[0] Coef | | 31 | A-C F-stat |
| 15 | CondC[1] Coef | | 32 | Full F-stat |
| 16 | CondC[2] Coef | | | |

Note that the above GLT's test whether the IRF's differ at *any* time lag $s$, $0 \leq s \leq 4$. To test whether the "area under the curve" is different, one could use the following modified test:

Program 3dDeconvolve Batch Command File for Example 1.4.4.6

```
3dDeconvolve \
    ⋮
(same as above)
    ⋮
-glt 1 DiffAB2.mat \
-glt 1 DiffAC2.mat
```

■

where the matrices which define these GLT's each contain a single row ($s = 1$):

Contents of file DiffAB2.mat:
0 0 1 1 1 1 1 −1 −1 −1 −1 −1 0 0 0 0 0
Contents of file DiffAC2.mat:
0 0 1 1 1 1 1 0 0 0 0 0 −1 −1 −1 −1 −1

One advantage of this approach is that the measure of the magnitude of each difference has been reduced to a single sub-brick, either A-B LC[0], or A-C LC[0]. This simplifies subsequent statistical analysis.

### 1.4.5 Multiple Linear Regression Approach to ANOVA

The following examples illustrate how multiple linear regression can be used to perform an ANOVA type analysis.

Previous versions of program 3dDeconvolve would always include an intercept ($b0$) (i.e., constant) term. However, the current version of the program allows the user to specify that *no* baseline parameters are to be included in the model (i.e., not even a constant term). This simplifies implementation of an ANOVA cell means model, as indicated by the following example..

**Example 1.4.5.1 Cell means ANOVA model**

As an illustration of using 3dDeconvolve to solve a pure ANOVA type problem, we will consider the "Castle Bakery" example from Ref.[2]. In that example, there are 2 factors: Factor A, at 3 levels; and Factor B, at 2 levels. Letting

$$A_iB_j = \begin{cases} 1 & \text{if Factor A is at level i and Factor B is at level j} \\ 0 & \text{otherwise} \end{cases}$$

then the cell means model would be given by:

$$Y = aA_1B_1 + bA_1B_2 + cA_2B_1 + dA_2B_2 + eA_3B_1 + fA_3B_2$$

The data for this experiment are presented in the following table:

| $Y$ | $A_1B_1$ | $A_1B_2$ | $A_2B_1$ | $A_2B_2$ | $A_3B_1$ | $A_3B_2$ |
|---|---|---|---|---|---|---|
| 47 | 1 | 0 | 0 | 0 | 0 | 0 |
| 43 | 1 | 0 | 0 | 0 | 0 | 0 |
| 46 | 0 | 1 | 0 | 0 | 0 | 0 |
| 40 | 0 | 1 | 0 | 0 | 0 | 0 |
| 62 | 0 | 0 | 1 | 0 | 0 | 0 |
| 68 | 0 | 0 | 1 | 0 | 0 | 0 |
| 67 | 0 | 0 | 0 | 1 | 0 | 0 |
| 71 | 0 | 0 | 0 | 1 | 0 | 0 |
| 41 | 0 | 0 | 0 | 0 | 1 | 0 |
| 39 | 0 | 0 | 0 | 0 | 1 | 0 |
| 42 | 0 | 0 | 0 | 0 | 0 | 1 |
| 46 | 0 | 0 | 0 | 0 | 0 | 1 |

The data from the above table, *without the column headings*, should be entered into file Castle.data.1D. Note that the first column contains the measured data, and the following 6 columns contain the indicator variables $A_1B_1$,...,$A_3B_2$.

To perform the ANOVA analysis, the indicator variables are entered as stim functions, as in the above equation. Do this, by executing the following script:

Program 3dDeconvolve Command Line for Example 1.4.5.1

```
3dDeconvolve \
-input1D 'Castle.data.1D[0]' \
-nfirst 0 \
-polort -1 \
-num_stimts 6 \
-stim_file 1 'Castle.data.1D[1]'   -stim_label 1 A1B1 \
-stim_file 2 'Castle.data.1D[2]'   -stim_label 2 A1B2 \
-stim_file 3 'Castle.data.1D[3]'   -stim_label 3 A2B1 \
-stim_file 4 'Castle.data.1D[4]'   -stim_label 4 A2B2 \
-stim_file 5 'Castle.data.1D[5]'   -stim_label 5 A3B1 \
-stim_file 6 'Castle.data.1D[6]'   -stim_label 6 A3B2
```

■

This generates the following output:

Program 3dDeconvolve Screen Output for Example 1.4.5.2

```
Program:           3dDeconvolve
Author:            B. Douglas Ward
Initial Release:   02 Sep 1998
Latest Revision:   27 Feb 2002
```

```
Results for Voxel #0:

  Baseline:

 Stimulus:  A1B1
  h[0] coef =  45.0000    h[0] t-st =   19.7974    p-value =  1.0773e-06
        R^2 =   0.9849       F[1,6] =  391.9355    p-value =  1.0773e-06

 Stimulus:  A1B2
  h[0] coef =  43.0000    h[0] t-st =   18.9175    p-value =  1.4098e-06
        R^2 =   0.9835       F[1,6] =  357.8710    p-value =  1.4098e-06

 Stimulus:  A2B1
  h[0] coef =  65.0000    h[0] t-st =   28.5962    p-value =  1.2109e-07
        R^2 =   0.9927       F[1,6] =  817.7419    p-value =  1.2109e-07

 Stimulus:  A2B2
  h[0] coef =  69.0000    h[0] t-st =   30.3560    p-value =  8.4809e-08
        R^2 =   0.9935       F[1,6] =  921.4839    p-value =  8.4809e-08

 Stimulus:  A3B1
  h[0] coef =  40.0000    h[0] t-st =   17.5977    p-value =  2.1612e-06
        R^2 =   0.9810       F[1,6] =  309.6774    p-value =  2.1612e-06

 Stimulus:  A3B2
  h[0] coef =  44.0000    h[0] t-st =   19.3574    p-value =  1.2306e-06
        R^2 =   0.9842       F[1,6] =  374.7097    p-value =  1.2306e-06

 Full Model
  MSE =  10.3333
  R^2 =   0.9981    F[6,6] =  528.9032    p-value =  6.7016e-08
```

■

Compare these results with the textbook values for the individual cell means.

**Example 1.4.5.2 Tests for Main Effects and Interactions**

We continue the previous example by testing for the presence of main effects and interactions.

Recall that the cell means model for this example is given by:

$$Y = aA_1B_1 + bA_1B_2 + cA_2B_1 + dA_2B_2 + eA_3B_1 + fA_3B_2$$

Therefore, the null hypothesis of the test for Factor A main effect is:

$$\begin{array}{rcl} a+b=c+d & \iff & a+b-c-d=0 \\ a+b=e+f & \iff & a+b-e-f=0 \end{array}$$

Since the parameter vector $\boldsymbol{\beta}$ is:

$$\boldsymbol{\beta}^t = \left[ \begin{array}{cccccc} a & b & c & d & e & f \end{array} \right]$$

the test for Factor A main effect can be represented by the following GLT matrix:

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & -1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 & -1 & -1 \end{bmatrix}$$

Store this matrix as 2 rows of ASCII numbers into file Castle.matA.

The null hypothesis of the test for Factor B main effect is:

$$a + c + e = b + d + f \iff a - b + c - d + e - f = 0$$

So, the test for Factor B main effect can be represented by the following GLT matrix:

$$\mathbf{C} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$$

Store this matrix as 1 row of ASCII numbers into file Castle.matB.

Finally, the null hypothesis of the test for A×B interaction is:

$$\begin{aligned} a + d = b + c &\iff a - b - c + d = 0 \\ a + f = b + e &\iff a - b - e + f = 0 \end{aligned}$$

So, the test for A×B interaction can be represented by the following GLT matrix:

$$\mathbf{C} = \begin{bmatrix} 1 & -1 & -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Store this matrix as 2 rows of ASCII numbers into file Castle.matAB.

That's all you will need to execute the following script:

Program 3dDeconvolve Command Line for Example 1.4.5.2

```
3dDeconvolve \
-input1D 'Castle.data.1D[0]' \
-nfirst 0 \
-polort -1 \
-num_stimts 6 \
-stim_file 1 'Castle.data.1D[1]'   -stim_label 1 A1B1 \
-stim_file 2 'Castle.data.1D[2]'   -stim_label 2 A1B2 \
-stim_file 3 'Castle.data.1D[3]'   -stim_label 3 A2B1 \
-stim_file 4 'Castle.data.1D[4]'   -stim_label 4 A2B2 \
-stim_file 5 'Castle.data.1D[5]'   -stim_label 5 A3B1 \
-stim_file 6 'Castle.data.1D[6]'   -stim_label 6 A3B2 \
-glt 2 Castle.matA   -glt_label 1 'Factor A' \
-glt 1 Castle.matB   -glt_label 2 'Factor B' \
-glt 2 Castle.matAB  -glt_label 3 'AB Interaction'
```

■

This generates the following output:

Program 3dDeconvolve Screen Output for Example 1.4.5.2

```
Program:            3dDeconvolve
Author:             B. Douglas Ward
Initial Release:    02 Sep 1998
Latest Revision:    27 Feb 2002
```

```
Results for Voxel #0:
```

(same as above)

```
General Linear Test:  Factor A
 LC[0] coef =  -46.0000   LC[0] t-st =  -10.1187    p-value =  5.4150e-05
 LC[1] coef =    4.0000   LC[1] t-st =    0.8799    p-value =  4.1277e-01
        R^2 =    0.9614       F[2,6] =   74.7097    p-value =  5.7536e-05

General Linear Test:  Factor B
 LC[0] coef =  -6.0000   LC[0] t-st =  -1.0776    p-value =  3.2261e-01
        R^2 =   0.1622       F[1,6] =   1.1613    p-value =  3.2261e-01

General Linear Test:  AB Interaction
 LC[0] coef =  6.0000   LC[0] t-st =  1.3198    p-value =  2.3501e-01
 LC[1] coef =  6.0000   LC[1] t-st =  1.3198    p-value =  2.3501e-01
        R^2 =  0.2791       F[2,6] =  1.1613    p-value =  3.7470e-01
```

■

Compare these results with the textbook values. Compare the calculated $F$-statistics and $p$-values for factor main effects and interactions.

### 1.4.6 Concatenation of Runs

**Example 1.4.6.1**

Let the 0-3 point IRF be:

$$h(t) = \{ \ 0 \quad 10 \quad 20 \quad 10 \ \}$$

Let the stimulus function be:

$$f(t) = \{ \ 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \ \}$$

Save $f(t)$ as a single column of numbers in file f.1D. Convolving the stimulus function with the IRF, we get:

$$f(t) \otimes h(t) = \{ \ 0 \quad 0 \quad 0 \quad 0 \quad 10 \quad 20 \quad 10 \quad 0 \quad 0 \quad 0 \ \}$$

If we model the measured response by adding a constant (100) plus slope (1),

$$y(t) = \{ \ 100 \quad 101 \quad 102 \quad 103 \quad 114 \quad 125 \quad 116 \quad 107 \quad 108 \quad 109 \ \}$$

Save $y(t)$ as a single column of numbers in file y.1D. Now, execute the following script:

```
3dDeconvolve \
-input1D y.1D \
-num_stimts 1 \
-stim_file 1 f.1D   -stim_label 1 'f'   -stim_maxlag 1 3
```

■

Verify that this yields the correct values for $h(t)$.

**Example 1.4.6.2**

Now, concatenate the stimulus function with itself:

```
cat f.1D f.1D > fcat.1D
```

and do the same with the measured data:

```
cat y.1D y.1D > ycat.1D
```

Be sure to remove any blank lines that may be present in files fcat.1D and ycat.1D. Now, analyze the concatenated datasets using 3dDeconvolve:

```
3dDeconvolve \
-input1D ycat.1D \
-num_stimts 1 \
-stim_file 1 fcat.1D   -stim_label 1 'fcat'   -stim_maxlag 1 3
```

■

Note that this yields the following estimate for the IRF:

$$h(t) = \left\{ \begin{array}{cccc} -2.2619 & 8.6447 & 19.5513 & 10.4579 \end{array} \right\}$$

Considering the fact that there is no noise present, can you explain why the estimate is so poor? Hint: What is the underlying model for the data?

**Example 1.4.6.3**

From the previous Example, we see that it is necessary to account for the fact that the separate runs have separate baselines (constant plus slope). One approach is to remove the linear trend from each run separately, prior to concatenation. To determine the linear trend, run 3dDeconvolve using y.1D as input, but without any stim functions (i.e., just use the baseline model = constant plus linear slope.). This can be done as follows:

```
3dDeconvolve   -input1D y.1D   -num_stimts 0
```

■

This yields $b_0 = 102.9091$ and $b_1 = 1.2424$. (Note: $b_0 \neq 100$ and $b_1 \neq 1$. Why?)

Now, remove the linear trend from the original data:

$$\begin{aligned} y_{rlt}(t) &= y(t) - b_0 - b_1 \cdot t \\ &= \{ \; -2.9091 \quad -3.1515 \quad -3.3939 \quad -3.6363 \quad 6.1213 \\ &\quad\; 15.8789 \quad 5.6365 \quad -4.6059 \quad -4.8483 \quad -5.0907 \; \} \end{aligned}$$

Store this as a single column in file yrlt.1D. Now, concatenate the detrended time series:

```
cat yrlt.1D yrlt.1D > yrltcat.1D
```

As before, make sure that any blank lines are removed. Now, apply 3dDeconvolve to the detrended, concatenated data:

```
3dDeconvolve \
-input1D yrltcat.1D \
-num_stimts 1 \
-stim_file 1 fcat.1D   -stim_label 1 'fcat'   -stim_maxlag 1 3
```

■

This yields:

$$h(t) = \{ \; 0.5483 \quad 10.3285 \quad 20.1088 \quad 9.8890 \; \}$$

This answer is much better than the previous answer (which did not use trend removal). However, the answer is still not perfect, even though no noise is present. Why? (Answer: The stimulus function is not orthogonal to a constant plus linear trend. Therefore, removing the linear trends prior to estimating the model parameters is *not* equivalent to estimating the linear trends and the model parameters simultaneously.)

**Example 1.4.6.4**

Here, we will use the -concat option to specify that the dataset is composed of concatenated runs, and to indicate where the individual runs begin. The program will then include separate linear trends for each run as the *baseline model* component of the *full model.* For the present case, there are 2 runs, which begin at index numbers 0 and 10. Therefore, save

```
0
10
```

as file runs.1D, and execute the following script:

```
3dDeconvolve \
-input1D ycat.1D \
-concat runs.1D \
-num_stimts 1 \
-stim_file 1 fcat.1D   -stim_label 1 'fcat'   -stim_maxlag 1 3
```

■

Verify that this yields the correct result:

$$h(t) = \{\ 0 \quad 10 \quad 20 \quad 10\ \}$$

**Example 1.4.6.5 Using -concat with a 3d+time dataset**

In Example 1.4.4.6, we calculated GLT's for a 3d+time dataset. We will continue with that example, only now we will assume that the input dataset is actually a concatenation of 4 runs, each of length 125 TR. In order to use the -concat option, it is first necessary to create a file containing the index numbers of the concatenated runs, e.g.,

```
0
125
250
375
```

Save this as file cat.1D.

Recall that in Example 1.4.4.6, we considered 3 different experimental conditions (A, B, and C). As before, for each of the 3 conditions, the IRF is to be estimated at 5 time points (0 TR through 4 TR). The individual impulse response functions are then to be compared with each other (A - B and A - C).

However, unlike the previous example, the full parameter vector now has 23 elements:

$$\begin{aligned}\boldsymbol{\beta}^t \;=\; & (\beta_{1,0}, \beta_{1,1}, \beta_{2,0}, \beta_{2,1}, \beta_{3,0}, \beta_{3,1}, \beta_{4,0}, \beta_{4,1}, \gamma_{a,0}, \gamma_{a,1}, \gamma_{a,2}, \gamma_{a,3}, \gamma_{a,4}, \\ & \gamma_{b,0}, \gamma_{b,1}, \gamma_{b,2}, \gamma_{b,3}, \gamma_{b,4}, \gamma_{c,0}, \gamma_{c,1}, \gamma_{c,2}, \gamma_{c,3}, \gamma_{c,4})\end{aligned}$$

Note that $\beta_0$ and $\beta_1$ are estimated independently for each of the 4 runs, accounting for 8 of the parameters. Suppose that the null hypotheses are specified by:

$$H_{o,1} : \gamma_{a,0} + \gamma_{a,1} + \gamma_{a,2} + \gamma_{a,3} + \gamma_{a,4} = \gamma_{b,0} + \gamma_{b,1} + \gamma_{b,2} + \gamma_{b,3} + \gamma_{b,4}$$

and

$$H_{o,2} : \gamma_{a,0} + \gamma_{a,1} + \gamma_{a,2} + \gamma_{a,3} + \gamma_{a,4} = \gamma_{c,0} + \gamma_{c,1} + \gamma_{c,2} + \gamma_{c,3} + \gamma_{c,4}$$

So, there are 2 GLT's. Each GLT is determined by a matrix (with $s = 1$ row and $P = 23$ columns), which must be stored in the user specified files DiffAB3.mat and DiffAC3.mat:

Contents of file DiffAB3.mat:

$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ -1\ -1\ -1\ -1\ -1\ 0\ 0\ 0\ 0\ 0$

Contents of file DiffAC3.mat:

$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ -1\ -1\ -1\ -1\ -1$

Note that the first 8 elements in each row (corresponding to the $\beta_0$'s and $\beta_1$'s) are set to zero. The following batch command file can be used to conduct the general linear tests (GLT's) for testing these two hypotheses, for the concatenated dataset.

Program 3dDeconvolve Batch Command File for Example 1.4.6.5

```
3dDeconvolve \
-input Subj307+orig   -concat cat.1D \
-polort 1   -rmsmin 1.0   -progress 1000 \
-num_stimts 3 \
-stim_file 1 CondA.1D   -stim_label 1 CondA   -stim_maxlag 1 4 \
-stim_file 2 CondB.1D   -stim_label 2 CondB   -stim_maxlag 2 4 \
-stim_file 3 CondC.1D   -stim_label 3 CondC   -stim_maxlag 3 4 \
-glt 1 DiffAB3.mat   -glt_label 1 ''A-B'' \
-glt 1 DiffAC3.mat   -glt_label 2 ''A-C'' \
-fout   -bucket Subj307.buck
```

■

After program 3dDeconvolve has finished execution, program afni can be used to view the output files. The format for the bucket dataset Subj307.buck+orig is illustrated below.

| Brick | Label | | Brick | Label |
|---|---|---|---|---|
| 0 | Run #1 t^0 Coef | | 16 | CondB[2] Coef |
| 1 | Run #1 t^1 Coef | | 17 | CondB[3] Coef |
| 2 | Run #2 t^0 Coef | | 18 | CondB[4] Coef |
| 3 | Run #2 t^1 Coef | | 19 | CondB F-stat |
| 4 | Run #3 t^0 Coef | | 20 | CondC[0] Coef |
| 5 | Run #3 t^1 Coef | | 21 | CondC[1] Coef |
| 6 | Run #4 t^0 Coef | | 22 | CondC[2] Coef |
| 7 | Run #4 t^1 Coef | | 23 | CondC[3] Coef |
| 8 | CondA[0] Coef | | 24 | CondC[4] Coef |
| 9 | CondA[1] Coef | | 25 | CondC F-stat |
| 10 | CondA[2] Coef | | 26 | A-B LC[0] |
| 11 | CondA[3] Coef | | 27 | A-B F-stat |
| 12 | CondA[4] Coef | | 28 | A-C LC[0] |
| 13 | CondA F-stat | | 29 | A-C F-stat |
| 14 | CondB[0] Coef | | 30 | Full F-stat |
| 15 | CondB[1] Coef | | | |

Note that the first 8 sub-bricks contain the estimated baseline parameters.

### 1.4.7 Censoring of Individual Time Points

Due to large measurement error, or for other reasons, the user may wish to remove individual time points from the measurement data. However, because of the time-dependent nature of the system response model used by program 3dDeconvolve, it is *not* correct to simply remove time points from a 3d+time dataset (and delete the corresponding time points from the input stimulus functions). For this reason, the -censor command should be used instead. This command allows for removal of individual time points from the analysis, while preserving the correct time dependencies. This is illustrated by the following examples.

**Example 1.4.7.1 Deleting Time Points**

Recall that in Example 1.4.2.3, since no noise was present, the program output gave the exactly correct parameter vector:

$$\mathbf{b}^t = \left\{ \begin{array}{ccccccc} 100.00 & 1.00 & 0.00 & 5.00 & 10.00 & 5.00 & 2.00 \end{array} \right\}$$

Suppose that we now delete one of the "measurements", along with the corresponding time point of the stimulus function. For example, if we delete the 9th time point, we have the input stimulus function:

$$gp(t) = \left\{ \begin{array}{ccccccccccccccccccc} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{array} \right\}$$

and the noisy measurement data:

$$wp(t) = \{ \begin{array}{cccccccccc} 100 & 106 & 117 & 118 & 111 & 112 & 121 & 127 & 116 & \\ 117 & 121 & 117 & 120 & 124 & 125 & 133 & 137 & 135 & 126 \end{array} \}$$

Save the above as columns into files gp.1D and wp.1D. Now, execute the following script, which uses the $gp(t)$ stimulus function, and the $wp(t)$ "data".

Program 3dDeconvolve Command Line for Example 1.4.7.1

```
3dDeconvolve \
-input1D wp.1D   -num_stimts 1  \
-stim_file 1 gp.1D   -stim_label 1 ''gp''   -stim_maxlag 1 4
```

■

The output estimated full model parameter vector is *not* correct:

$$\mathbf{b}^t = \left\{ \begin{array}{ccccccc} 102.08 & 1.25 & -1.04 & 4.71 & 7.71 & 2.31 & -0.81 \end{array} \right\}$$

even though there is *no* noise present, and even though there is sufficient redundancy in the data to yield the correct answer.

**Example 1.4.7.2 Censoring Time Points**

The correct way to remove one or more time points is to use the -censor command option in program 3dDeconvolve. For example, to eliminate the 9th data point, use

$$c(t) = \left\{ \begin{array}{cccccccccccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array} \right\}$$

Save this as a column into file c.1D. Note that this array consists of all 1's, except for a 0 at the time point to be "censored". Also, note that the arrray has the same length as the original input data. Execute the following script, which, in addition to the censor array $c(t)$, also uses the *original* $g(t)$ stimulus function:

$$g(t) = \{\ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \}$$

and the *original* $w(t)$ "data":

$$w(t) = \{ \begin{array}{cccccccccc} 100 & 106 & 117 & 118 & 111 & 112 & 121 & 127 & 125 & 116 \\ 117 & 121 & 117 & 120 & 124 & 125 & 133 & 137 & 135 & 126 \end{array} \}$$

Program 3dDeconvolve Command Line for Example 1.4.7.2

```
3dDeconvolve \
-input1D w.1D   -num_stimts 1  \
-censor c.1D \
-stim_file 1 g.1D   -stim_label 1 ''g''   -stim_maxlag 1 4
```

■

This again yields the correct answer, this time using only 19 of the 20 data points:

$$\mathbf{b}^t = \{\ 100.00\ \ 1.00\ \ 0.00\ \ 5.00\ \ 10.00\ \ 5.00\ \ 2.00\ \}$$

**Exercise 1.4.7.3**
Can you explain the difference in the above results? Why did simple deletion of a time point produce the wrong answer? (Hint: Repeat the above examples, but add the -xout command. Examine and compare the experimental design matrices for these two examples.)

### 1.4.8 Sub-TR Input Stimulus Functions

In all of the previous examples, it was assumed that the input stimulus functions could change state only at multiples of the TR time unit. In this section, we consider a stimulus function that is allowed to change at a sub-multiple of TR. Note that the following examples make use of program RSFgen, which will be described later in Section 3, and program 3dConvolve, which will be described in Section 4.

**Example 1.4.8.1 NPTR = 1**
To establish a baseline for comparison, we will generate "data" and an input stimulus function which are sampled at the same rate (i.e., 1/TR). The following command line will generate a random binary stimulus function of length 200 time points.

Program RSFgen Command Line for Example 1.4.8.1

```
RSFgen   -nt 200   -num_stimts 1   -nreps 1 100 \
-one_file   -prefix myRandBin
```

■

The above command line generates the following 200 point random binary sequence, which is stored in file myRandBin.1D:

$$
\begin{array}{ll}
f(t) = \{ & 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1 \\
 & 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ \ldots \}
\end{array}
$$

The model that we will use for the measured data is:

$$
\begin{aligned}
y(n) &= b_0 + b_1 n + f(n) \otimes h(n) \\
&= b_0 + b_1 n + h_0 f(n) + h_1 f(n-1) + \cdots + h_5 f(n-5)
\end{aligned}
$$

where the baseline parameters $\mathbf{b}$ and the IRF $\mathbf{h}$ are given by:

$$
\begin{aligned}
\mathbf{b}^t &= \begin{bmatrix} b_0 & b_1 \end{bmatrix} & \mathbf{h}^t &= \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & h_4 & h_5 \end{bmatrix} \\
&= \begin{bmatrix} 100.0 & 0.1 \end{bmatrix} & &= \begin{bmatrix} 0.0 & 2.0 & 4.0 & 5.0 & 3.0 & 1.0 \end{bmatrix}
\end{aligned}
$$

In order to create the "data", we will convolve the stimulus function $f(t)$ with the IRF $h(t)$, and add the baseline. First, store the baseline parameter vector $\mathbf{b}$ as a column in file myBase.1D, and store the IRF vector $\mathbf{h}$ as a column in file myIRF.1D. Now, create the data with the following command line:

Program 3dConvolve Command Line for Example 1.4.8.1

```
3dConvolve \
-input1D \
-nfirst 0   -nlast 199   -polort 1 \
-base_file myBase.1D \
-num_stimts 1 \
-stim_file 1 myRandBin.1D   -stim_maxlag 1 5 \
-iresp 1 myIRF.1D \
-output myData
```

■

The output file, mydata.1D, contains the data vector $\mathbf{y}$, which is displayed below. Plot the stimulus function $f(t)$ and the time series "data" $y(t)$ using program 1dplot. Now, use program 3dDeconvolve to analyze the "data":

Program 3dDeconvolve Command Line for Example 1.4.8.1

```
3dDeconvolve \
-input1D myData.1D \
-nfirst 0   -nlast 199   -polort 1 \
-num_stimts 1 \
-stim_file 1 myRandBin.1D   -stim_maxlag 1 5 \
-xout
```

■

The -xout command causes the design matrix **X** to be written to the screen. The design matrix **X**, and the data vector **y**, are displayed below.

$$\mathbf{X} = [\mathbf{x}_0\ \mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3\ \mathbf{x}_4\ \mathbf{x}_5\ \mathbf{x}_6\ \mathbf{x}_7] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 5 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 6 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 7 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 8 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 10 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 11 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 12 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 13 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 14 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 15 & 1 & 0 & 1 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} 100.0 \\ 100.1 \\ 100.2 \\ 100.3 \\ 102.4 \\ 106.5 \\ 111.6 \\ 112.7 \\ 111.8 \\ 110.9 \\ 113.0 \\ 115.1 \\ 114.2 \\ 110.3 \\ 107.4 \\ 106.5 \\ \vdots \end{bmatrix}$$

Verify that the columns of **X** consist of time delayed version of $f$, as listed below:

$$\begin{array}{llll} \mathbf{x}_0[k] = 1 & \mathbf{x}_1[k] = k & \mathbf{x}_2[k] = f[k] & \mathbf{x}_3[k] = f[k-1] \\ \mathbf{x}_4[k] = f[k-2] & \mathbf{x}_5[k] = f[k-3] & \mathbf{x}_6[k] = f[k-4] & \mathbf{x}_7[k] = f[k-5] \end{array}$$

Since the "data" contains no noise, verify that the 3dDeconvolve output estimated parameter vector is exactly correct:

$$\left[\mathbf{b}^t\ \mathbf{h}^t\right] = \left[\ 100.00 \quad 0.10 \quad 0.00 \quad 2.00 \quad 4.00 \quad 5.00 \quad 3.00 \quad 1.00\ \right]$$

**Example 1.4.8.2 NPTR = 2**

For direct comparison with the previous example, we will generate data that corresponds to a sub-sampling of the input data. That is, we will effectively double the TR by discarding every 2nd data point from the above y(t) time series. This can be done manually, which is somewhat tedious, or we can use the following command line to accomplish the same thing. Note that the IRF coefficient file, myIRF.1D, is the same as before. However, due to the new TR, the new baseline parameter file, myBase2.1D, must be modified slightly from before, i.e.:

$$\mathbf{b}^t = \begin{bmatrix} 100.0 & 0.2 \end{bmatrix}$$

Now, execute the following:

Program 3dConvolve Command Line for Example 1.4.8.2

```
3dConvolve \
-input1D \
-nfirst 0   -nlast 99   -polort 1 \
-base_file myBase2.1D \
-num_stimts 1 \
-stim_file 1 myRandBin.1D   -stim_maxlag 1 5 \
-stim_nptr 1 2 \
-iresp 1 myIRF.1D \
-output myData2
```

■

Verify that the output file myData2.1D, which is displayed below as vector $\mathbf{y}$, contains the 100 time point sub-sampled data. Use program 1dplot to plot both the original and the sub-sampled data, and compare. Since the TR has been doubled, this means that the original input stimulus function $f(t)$ is sampled at a rate of $\frac{2}{TR}$ relative to the new TR. The model that we will use for the measured data with the new TR is:

$$\begin{aligned} y(n) &= b_0 + b_1 n + f(2n) \otimes h(n) \\ &= b_0 + b_1 n + h_0 f(2n) + h_1 f(2n-1) + \cdots + h_5 f(2n-5) \end{aligned}$$

Therefore, in order to analyze this data, it is necessary to use the -stim_nptr 1 2 command, as indicated below.

Program 3dDeconvolve Command Line for Example 1.4.8.2

```
3dDeconvolve \
-input1D myData2.1D \
-nfirst 0   -nlast 99   -polort 1 \
-num_stimts 1 \
-stim_file 1 myRandBin.1D   -stim_maxlag 1 5 \
-stim_nptr 1 2 \
-xout
```

■

As before, the -xout command is used to write the $\mathbf{X}$ matrix to the screen. Compare this $\mathbf{X}$ matrix (displayed below) with the previous $\mathbf{X}$ matrix.

$$
\mathbf{X} = [\mathbf{x}_0\ \mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3\ \mathbf{x}_4\ \mathbf{x}_5\ \mathbf{x}_6\ \mathbf{x}_7] = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 3 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 4 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 5 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 6 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 7 & 0 & 1 & 0 & 0 & 1 & 1 \\
1 & 8 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 9 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 10 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 11 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 12 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 13 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 14 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 15 & 0 & 0 & 1 & 1 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix}
100.0 \\ 100.2 \\ 102.4 \\ 111.6 \\ 111.8 \\ 113.0 \\ 114.2 \\ 107.4 \\ 108.6 \\ 113.8 \\ 117.0 \\ 115.2 \\ 108.4 \\ 111.6 \\ 108.8 \\ 112.0 \\ \vdots
\end{bmatrix}
$$

Verify that the columns of the current **X** matrix are given by:

$$
\begin{array}{llll}
\mathbf{x}_0[k] = 1 & \mathbf{x}_1[k] = k & \mathbf{x}_2[k] = f[2k] & \mathbf{x}_3[k] = f[2k-1] \\
\mathbf{x}_4[k] = f[2k-2] & \mathbf{x}_5[k] = f[2k-3] & \mathbf{x}_6[k] = f[2k-4] & \mathbf{x}_7[k] = f[2k-5]
\end{array}
$$

As before, since no noise is present, verify that the output parameter vector is exactly correct.

**Exercise 1.4.8.3**

Repeat Example 1.4.8.1, but use a TR that is 3 times longer than the original TR. This will require using the command -stim_nptr 1 3. Can you predict the contents of the **X** matrix?

**Exercise 1.4.8.4**

The previous examples used input data with no noise. Try adding noise to the input data, and compare the parameter estimates with the "no noise" results. For example, add the command -sigma 1.0 to the script for program 3dConvolve.

# 2 Program plug_deconvolve

## 2.1 Purpose

Program plug_deconvolve is an *AFNI* "plug-in" which displays either the fitted output waveform (on top of the actual time series data), or the residuals from fitting the output waveform, or the estimated impulse response function waveform, for voxels of interest. Program plug_deconvolve is the interactive version of the batch command program 3dDeconvolve. The reader is strongly advised to consult the documentation for program 3dDeconvolve first.

## 2.2 Usage

To use plug_deconvolve, first one must be running afni. Display the Image and Graph for Axial, Sagittal, or Coronal views, for the measured FMRI 3d+time dataset. 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 Deconvolution.

This opens the Deconvolution 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 is the Control option line, the Concat option line, the Censor option line, followed by 20 StimFnc option lines, and, finally, 20 GLT Mat option lines.

On the Control option line, there are four option choosers: Base, NFirst, NLast, and IRF. The Base option lets the user specify the baseline (polynomial) model: None, Const, Linear, Quadrtc, Cubic, Quartic, or Quintic. That is, the baseline is modeled by either '0', or '$a$', or '$a + bt$', or '$a + bt + ct^2$', or etc. The NFirst box allows the user to specify the initial time series data point to include when performing the deconvolution analysis. The default value NFirst = -1 has the special meaning: "Set NFirst to the maximum of the MaxLag's", as defined below. The third number chooser on the Control option line is labeled NLast. This option allows the user to specify the number of the last image to use in the analysis. The last option on the Control option line is IRF. When plotting the impulse response function, and when there are multiple stimulus functions, this option allows the user to select, by means of the attached Label, which impulse response function is to be plotted.

This is followed by the Concat option line. This option is used to specify that the input 3d+time dataset is a concatenation of runs. The File input is used to select the name of the .1D file containing the list of volume indices of starting points for the individual runs. For a multi-column .1D file, the Col # option lets the user specify which column contains the list of concatenated run indices. The optional Label input lets the user attach a name to the concatenation input. See Section 1.2.12 for more details about concatenation of runs.

Next is the Censor option line. This option is used to specify that certain time points are to be removed from the analysis. The File input is used to select the name of the .1D file containing a column of 1's (for time points which are used) and 0's (for time points which are excluded). For a multi-column .1D file, the Col # option lets the user specify which column contains the list of points to be censored. The optional Label input lets the user attach a name to the censor input. See Section 1.2.13 for more details about censoring of data points.

Below the Censor option line are 20 StimFnc option lines. As the name implies, the StimFnc option line allows the user to select the time series data file which represents the input stimulus function. There are seven options on each StimFnc option line. The first option, Label, allows the user to enter a short character string for identification of that particular stimulus. This is particularly useful for labeling the program output when there are multiple stimuli. The second option, File, lets the user specify the name of the .1D time series file which represents the stimulus function. The third option, Col #, allows the user to specify the column of the .1D time series file to use for the stimulus function. This option allows the program to access multi-column .1D files. The fourth and fifth options allow the user to specify the minimum and maximum time lags, MinLag and MaxLag, for the estimated impulse response function. The sixth option, NPTR, allows the user to specify the number of time points $p$ per TR for this stimulus. The default value is $p = 1$. Note that the minimum and maximum time lags are specified in units of TR/$p$. The seventh option, Base, allows the user to specify whether this stimulus function should be included as part of the baseline model, for calculation of the full model regression statistics.

Below the Stimulus option lines there are 20 GLT Matrix option lines. The user may optionally assign a Label for the GLT output. For each GLT to be computed, the user must specify the number of Rows (i.e., number of linear constraints) in the matrix, and the name of the File which contains the matrix.

## 2.3 Examples

**Example 2.3.1**

We will assume that the current subdirectory contains the *AFNI* 3d+time dataset of interest, which we will take to be v2:time+orig, plus the bucket dataset file v2:bucket+orig, which last was created by program 3dDeconvolve. To start the program, type afni. First, from the main menu, click on Switch Anatomy. From the Anatomy submenu, choose v2:time. Then, 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 Function. From the pop-up menu, choose v2:bucket [fbuc], which contains the *AFNI* statistical parametric maps which were generated by program 3dDeconvolve. Click on Set to select this option. For the Func sub-brick choose the Cosine[0] Coef., and for the Thr sub-brick, choose the Cosine F-stat (partial F-statistic for significance of the Cosine stimulus function). 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 fit to the data for the Cosine stimulus function within the multiple linear regression 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 the lag-0 Cosine stimulus function coefficient (in this case). (Note that any parameter subbrick can be selected as the Func subbrick for color coding, and any statistical subbrick can be selected as the Thr subbrick for thresholding).

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 x-, y-, and z-coordinates of the center voxel are displayed in the Graph window. Use the crosshairs to move to: x=29, y=29, z=8. The time series can be vertically rescaled by using the '+', '-', and 'a' keys.

To initialize the deconvolution 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 Deconvolution. This pops up the Deconvolution Function control box. For the Base, choose Linear. *Important:* For NFirst, choose 6.

Press the button next to the first StimFnc option line. For the first stimulus Label, enter Cosine. For the File, choose the time series file name cos7.00.1D. For MinLag, choose 0, and for MaxLag, choose 1. Now, press Run + Keep. The program will then write to the text window the following information:

```
Program:            plug_deconvolve
Author:             B. Douglas Ward
Initial Release:    09 Sept 1998
Latest Revision:    29 Jan 2002

Controls:
Baseline   =  Linear
NFirst     =  6
NLast      =  32767
IRF label  =

Concatenation:  Label =            Column = 0
Run #1          Initial Point = 0

Stimulus:  Label = Cosine  Column = 0  Min.Lag = 0  Max.Lag = 1  NPTR = 1
```

■

Check to make sure that the option choices listed agree with the choices that you intended to make.

To overlay a plot of the deconvolution 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 DC_Fit. The resulting plot should be similar to that which was obtained in Example 1.4.3.1.

For each voxel whose time series is displayed, the program writes relevant statistical information into the text window. To view the multiple linear regression parameters for 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, as depicted below:

Deconvolution Plugin Screen Output for Example 2.3.1

```
Ignore   =  0
x voxel  =  29
y voxel  =  29
z voxel  =  8
min      =  800
max      =  934
mean     =  841.353
sigma    =  15.68226
Median   =  841.5
MAD      =  7.5

Baseline:
 t^0 coef =  835.2533    t^0 t-st =  329.9855    p-value =  1.1980e-06
 t^1 coef =    0.1494    t^1 t-st =    2.3987    p-value =  1.9689e-02

Stimulus:  Cosine
 h[0] coef =  0.0074    h[0] t-st =   3.2738    p-value =  1.7917e-03
 h[1] coef =  0.0019    h[1] t-st =   0.8550    p-value =  3.9605e-01
       R^2 =  0.3546      F[2,58] =  15.9330    p-value =  3.0559e-06

Full Model:
 MSE =  76.6046
 R^2 =   0.3546    F[2,58] =  15.9330    p-value =  3.0559e-06
```

■

**Example 2.3.2**

We continue the previous example, but will now enter the estimated motion parameters as additional stimulus functions. We will assume that the estimated motion parameters are contained in file v2.motion.1D (see Example 1.4.3.2). The six columns of this file will be entered as additional stimulus functions.

Since we are going to enter 6 more input stimuli, press the button next to StimFnc option on 6 separate lines. This is in addition to the previous StimFnc option line, which is left unchanged. For the stimulus Label's, enter Roll, Pitch, Yaw, dS, dL, and dP. For the File, choose the time series file name v2.motion.1D, on each of the 6 lines. Leave MinLag and MaxLag at their default values (0). Since the motion parameters are part of the baseline model (corresponding to the null hypothesis), set Base to True for each of these 6 stimulus functions. Now, press Run + Keep. Note that this *only* initializes the control parameters, but does *not* cause the program to perform the calculations for this new model. To do this, you must place the cursor in the center box, and press the left-most button on the mouse (i.e., Button 1). The program will then calculate and display the fit for the new model for each voxel in the graph. Now, move the cursor into the center box 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, as depicted below:

Deconvolution Plugin Screen Output for Example 2.3.2

⋮
etc.
⋮

```
Baseline:
 t^0 coef =  820.7942    t^0 t-st =  181.1599    p-value =  1.6640e-74
 t^1 coef =    0.3943    t^1 t-st =    3.2174    p-value =  2.2281e-03

Stimulus:  Cosine
 h[0] coef =  0.0054    h[0] t-st =   2.6237    p-value =  1.1390e-02
 h[1] coef =  0.0040    h[1] t-st =   2.0555    p-value =  4.4864e-02
       R^2 =  0.4637      F[2,52] =  22.4770    p-value =  9.2337e-08

Baseline:  Roll
 h[0] coef =  132.4883    h[0] t-st =  2.0368    p-value =  4.6778e-02
       R^2 =    0.0739      F[1,52] =  4.1485    p-value =  4.6778e-02

Baseline:  Pitch
 h[0] coef =  -18.0019    h[0] t-st =  -0.4593    p-value =  6.4796e-01
       R^2 =    0.0040      F[1,52] =   0.2109    p-value =  6.4796e-01

Baseline:  Yaw
 h[0] coef =  -87.6356    h[0] t-st =  -2.9877    p-value =  4.2826e-03
       R^2 =    0.1465      F[1,52] =   8.9261    p-value =  4.2826e-03

Baseline:  dS
 h[0] coef =  24.2552    h[0] t-st =  2.7178    p-value =  8.9066e-03
       R^2 =   0.1244      F[1,52] =  7.3865    p-value =  8.9066e-03

Baseline:  dL
 h[0] coef =  18.3589    h[0] t-st =  0.4066    p-value =  6.8597e-01
       R^2 =   0.0032      F[1,52] =  0.1653    p-value =  6.8597e-01

Baseline:  dP
 h[0] coef =  -29.6341    h[0] t-st =  -0.6048    p-value =  5.4791e-01
       R^2 =    0.0070      F[1,52] =   0.3658    p-value =  5.4791e-01

Full Model:
 MSE =  46.9907
 R^2 =   0.4637    F[2,52] =  22.4770    p-value =  9.2337e-08
```

■

Both from the graphical display of the fitted curve, as well as from the displayed statistical values, it is obvious that including the motion parameters has yielded a better fit to the data.

# 3 Program RSFgen

## 3.1 Purpose

Program RSFgen is a simple program for generating random stimulus functions. In addition to specifying the total number of time points and the number of stimulus functions, the user can input the number of repetitions for each stimulus function, and the block length (i.e., stimulus duration) for each stimulus function. The program randomly chooses the "on" times for each of the stimuli within the time allotted. Alternatively, the user can specify the Markov chain transition probability matrix for generating the stimulus functions as a random process. Program output consists of a separate ".1D" file for each stimulus function, or a single ".1D" file containing a separate column for each stimulus function.

The syntax and user options are described in Section 3.2. Several examples of applications of program RSFgen are presented in Section 3.3. Note that the output from program RSFgen can be used as input to program 3dDeconvolve. Specifically, the "-nodata" option within program 3dDeconvolve can be used to evaluate different alternative experimental designs created using RSFgen. This is discussed in Section 3.4.

## 3.2 Usage

### 3.2.1 Syntax

Syntax for Random Permutation method:
**RSFgen -nt** *n* **-num_stimts** *p* [**-seed** *s*] [**-one_file** | **-one_col**] [**-prefix** *pname*] \
**-nreps** *1 r1* **...** **-nreps** *p rp* [**-nblock** *1 k1*] **...** [**-nblock** *p kp*] [**-pseed** *s*]

Syntax for Markov Chain method:
**RSFgen -nt** *n* **-num_stimts** *p* [**-seed** *s*] [**-one_file** | **-one_col**] [**-prefix** *pname*] \
[**-nblock** *1 k1*] **...** [**-nblock** *p kp*] **-markov** *mfile* [**-pzero** *z*]

### 3.2.2 Options

| | |
|---|---|
| **-nt** *n* | $n$ = length of time series. |
| **-num_stimts** *p* | $p$ = number of input stimuli or experimental conditions. |
| **-nblock** *i k* | $k$ = block length (i.e., stimulus duration) for stimulus $i$ ($1 \leq i \leq p$). (default: $k = 1$) |
| **-seed** *s* | $s$ = random number seed (optional). This number can be changed to create different random sequences. |

**-one_file** The stimulus functions should be written as separate columns into a single .1D file.

**-one_col** The stimulus functions should be written as a single column of decimal integers into a single .1D file.

**-prefix** *pname* *pname* = prefix for $p$ output .1D stimulus functions (optional) e.g., pname1.1D, pname2.1D, ..., pnamep.1D
Note: If the -one_file option is used, all $p$ stimulus functions are output to file pname.1D.

The following Random Permutation and Markov Chain options are mutually exclusive:

Random Permutation options:

**-nreps** *i r* $r$ = number of repetitions for stimulus $i$ $(1 \leq i \leq p)$.

**-pseed** *s* $s$ = stimulus label permutation random number seed (optional)

*Note:* Obviously, it is necessary that: $n \geq \sum_{i=1}^{p} (r[i] * k[i])$ .

Markov Chain options:

**-markov** *mfile* $mfile$ = file containing Markov chain transition probability matrix.

**-pzero** *z* $z$ = probability of a zero (i.e., null) state
(default: $z = 0$)

*Warning:* This program will overwrite pre-existing .1D files that have the same name(s).

## 3.3 Examples

In this section we present several examples of possible applications of program RSFgen. We begin by using RSFgen to create event-related experimental designs. This is followed by considering randomized block designs. We then delve into stimulus label permutations. This section ends with an example of the Markov chain option for generating random stimulus functions.

### 3.3.1 Event Related Designs

Program RSFgen can be used to create event-related designs. By an "event-related" design, we mean an experiment in which the minimum stimulus duration is 1 TR. There may be one or more than one type of event. A separate stimulus function is generated for each type of event. The different events are placed randomly within the time allotted.

In the first example, we consider the more common case of mutually exclusive events; i.e., at most one type of event can occur at any given point in time. The second example considers the case where the different event types occur independently; that is, the different events can occur simultaneously.

**Example 3.3.1.1 Mutually Exclusive Events**

A researcher wishes to conduct an experiment using 6 different experimental conditions: A, B, C, D, E, and F. These experimental conditions are mutually exclusive; i.e., at most one experimental condition can apply at any given time point. The total imaging run will last 200 TR. Within this 200 TR, conditions A and B are to occur 20 times each, conditions C and D are to occur 25 times each, and conditions E and F are to occur 30 times each. Each "on" period corresponds to 1 TR. The command line for randomly generating the 6 stimulus functions is presented below:

Program RSFgen Command Line for Example 3.3.1.1:

```
RSFgen \
-nt 200   -num_stimts 6  \
-nreps 1 20   -nreps 2 20   -nreps 3 25  \
-nreps 4 25   -nreps 5 30   -nreps 6 30  \
-seed 1234567   -one_file   -prefix Stim
```

■

The program first creates a time series with 0 = "rest", 1 = "condition A", 2 = "condition B", 3 = "condition C", etc. , with each symbol occurring the specified number of times. The order of appearance of the stimuli is then randomized by shuffling the time series. Finally, the individual stimulus functions are written to file Stim.1D, as indicated below:

Program RSFgen Screen Output for Example 3.3.1.1:

```
Program:  RSFgen
Author:  B. Douglas Ward
Date:  14 January 2000


nt = 200
num_stimts = 6
nreps[1] = 20
nreps[2] = 20
nreps[3] = 25
nreps[4] = 25
nreps[5] = 30
nreps[6] = 30
seed = 1234567


Original array:
```

```
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3
 3  3  3  3  3  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
 6  6  6  6  6  6  6  6  6  6  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

Shuffled array:
 0  5  6  1  3  5  0  2  1  1  1  6  6  2  6  2  1  5  3  3
 3  6  0  4  4  0  0  1  2  5  0  1  2  0  4  6  0  1  6  5
 4  0  4  5  5  6  4  3  5  0  0  5  6  2  6  3  6  5  3  6
 5  5  6  2  0  0  0  6  3  6  6  4  2  6  4  0  2  6  5  1
 1  5  4  6  4  0  3  3  2  5  0  5  3  3  2  6  5  0  1  3
 0  1  0  1  0  3  0  0  3  0  1  4  0  5  0  3  2  6  4  5
 3  0  2  3  6  0  0  3  4  6  5  0  2  0  4  4  5  6  6  4
 6  6  2  4  0  3  6  0  0  2  4  0  0  4  0  6  1  2  0  0
 0  2  1  0  4  2  0  2  4  0  3  3  0  4  4  5  3  1  5  6
 1  5  5  5  0  5  0  5  3  1  4  5  0  0  0  4  5  1  3  0
```

■

The output file Stim.1D contains 6 columns; each column corresponds to a particular stimulus, and contains a "1" at each time point where that stimulus occurs, and a "0" elsewhere, as shown below:

Contents of file Stim.1D:

| | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Column #0: | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | etc. |
| Column #1: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | etc. |
| Column #2: | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | etc. |
| Column #3: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | etc. |
| Column #4: | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | etc. |
| Column #5: | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | etc. |

Note that the separate stimulus functions can be accessed by program 3dDeconvolve through the column indices (0,...,5).

**Example 3.3.1.2 Overlapping (Independent) Experimental Conditions**

In this example, we consider an experiment in which 4 different conditions (A, B, C, and D) occur independently of each other, and possibly coincide. In this case, we execute program RSFgen 4 times, once for each condition. It is very important that each of the 4 runs uses a *different* random number seed, in order to create 4 independent sequences. This is illustrated by the following script:

Script File for Example 3.3.1.2:

```
RSFgen  -nt 200  -num_stimts 1  -nreps 1 50  -seed 1234561  -prefix StimA
RSFgen  -nt 200  -num_stimts 1  -nreps 1 50  -seed 1234562  -prefix StimB
RSFgen  -nt 200  -num_stimts 1  -nreps 1 50  -seed 1234563  -prefix StimC
RSFgen  -nt 200  -num_stimts 1  -nreps 1 50  -seed 1234564  -prefix StimD

1dtranspose StimA1.1D  StimA.1Dt
1dtranspose StimB1.1D  StimB.1Dt
1dtranspose StimC1.1D  StimC.1Dt
1dtranspose StimD1.1D  StimD.1Dt

rm -f Stim*.1D

cat StimA.1Dt StimB.1Dt StimC.1Dt StimD.1Dt > Stim.1Dt

1dtranspose Stim.1Dt Stim.1D

rm -f Stim*.1Dt
```

■

Note that the above script, in addition to creating the 4 stim functions, assembles these stim functions as 4 columns into the single file Stim.1D. This is not required; however, it might make things simpler to have all of the stim functions contained in a single file.

The file Stim.1D contains 4 columns; each column corresponding to a particular experimental condition. That is, each column contains a "1" at each time point where that condition occurs, and a "0" elsewhere, as shown below:

Contents of file Stim.1D:

| | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Column #0: | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | etc. |
| Column #1: | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | etc. |
| Column #2: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | etc. |
| Column #3: | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | etc. |

Note that at some time points none of the experimental conditions occur; at other time points, as many as all 4 conditions occur simultaneously.

### 3.3.2 Random Block Designs

The previous examples assumed that the minimum duration for an experimental condition was 1 TR. In the following example, we consider an experimental design where the "on" state for each experimental condition has a fixed length (i.e., the "block" length) that is a multiple of 1 TR.

**Example 3.3.2.1 Mutually Exclusive Random Blocks**

In this example, we consider how to use program RSFgen to create a randomized block design, where the experimental conditions are mutually exclusive. Suppose that there are 3 experimental conditions: A, B, and C. For conditions A, B, and C, each "on" period has duration 5, 3, and 2 TR, respectively. The entire imaging run will last 200 TR. We will use $r = 10$ repetitions for each stimulus.

Program RSFgen Command Line for Example 3.3.2.1

```
RSFgen  \
-nt 200   -num_stimts 3  \
-nreps 1 10   -nblock 1 5 \
-nreps 2 10   -nblock 2 3 \
-nreps 3 10   -nblock 3 2 \
-seed 1234567   -one_file   -prefix RandBlock
```

■

Note that the -nblock command was used 3 times in order to specify the duration for each of the stimulus functions. The following is the screen output generated by the above commands:

Program RSFgen Screen Output for Example 3.3.2.1

```
Program:  RSFgen
Author:  B. Douglas Ward
Date:  14 January 2000


nt = 200
num_stimts = 3
nreps[1] = 10    nblock[1] = 5
nreps[2] = 10    nblock[2] = 3
nreps[3] = 10    nblock[3] = 2
seed = 1234567


Original array:
 1  1  1  1  1  1  1  1  1  1  2  2  2  2  2  2  2  2  2  2
 3  3  3  3  3  3  3  3  3  3  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0

Shuffled array:
 3  0  1  0  0  0  2  0  0  0  0  0  0  1  0  0  1  0  1  0
 0  0  2  0  0  2  2  0  3  0  0  0  2  0  0  0  0  2  1  1
 0  0  0  0  0  0  0  0  2  2  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  3  0  0  0  0
 0  0  0  0  0  0  3  3  3  0  0  0  0  0  0  0  0  0  0  0
 0  1  0  0  0  0  1  0  1  3  0  0  1  0  0  3  0  3  0  0
 0  0  3  2  0  0  0  0  0  0
```

```
Expanded array:
 3  3  0  1  1  1  1  1  0  0  0  2  2  2  0  0  0  0  0  0
 1  1  1  1  1  0  0  1  1  1  1  1  0  1  1  1  1  1  0  0
 0  2  2  2  0  0  2  2  2  2  2  2  0  3  3  0  0  0  2  2
 2  0  0  0  0  2  2  2  1  1  1  1  1  1  1  1  1  1  0  0
 0  0  0  0  0  0  2  2  2  2  2  2  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  2  2  2  0  0  0  0  0  0  3
 3  0  0  0  0  0  0  0  0  0  0  3  3  3  3  3  3  0  0  0
 0  0  0  0  0  0  0  0  0  1  1  1  1  1  0  0  0  0  1  1
 1  1  1  0  1  1  1  1  1  3  3  0  0  1  1  1  1  1  0  0
 3  3  0  3  3  0  0  0  0  3  3  2  2  2  0  0  0  0  0  0
```

■

Note that the "Expanded array" is formed by replacing each '1', '2', or '3' from the "Shuffled array' with a '11111', '222', or '33', respectively.

The output file "RandBlock.1D" contains 3 columns, corresponding to the 3 different stimulus functions. A "1" appears at each time point where each individual stimulus occurs, and a "0" elsewhere, as shown below:

Contents of file RandBlock.1D:

| | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Column #0: | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | etc. |
| Column #1: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | etc. |
| Column #2: | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | etc. |

■

Note that the 3 stimulus functions can be referenced by using the column selectors: [0], [1], and [2]. Also, note that the column selectors must be enclosed by quotes.

**Exercise 3.3.2.2 Independent Random Block Design**

Using the method of Example 3.3.1.2, repeat the experimental block design of Example 3.3.2.1, but for independent (i.e., overlapping) experimental conditions.

### 3.3.3 Stimulus Label Permutation Designs

Normally, one would use the -seed $s$ option to create different random experimental designs. That is, different values for the random seed $s$ will create different random permutations of the sequence of stimulus events within the experimental design. However, there are some cases where one might wish to preserve the temporal locations of events, while at the same time performing a random permutation of the stimulus labels. This "randomization-within-a-randomization" can be accomplished with the -pseed $s$ option. The following examples should help clarify the difference between the -seed and -pseed options.

**Example 3.3.3.1 Initial Random Block Design**

First, we consider a random block design, with 3 experimental conditions: A, B, and C. Each condition is to occur 10 times, with a block length of 5 TR. This design is created with the following command line, using random number seed option -seed 1234567:

Program RSFgen Command Line for Example 3.3.3.1

```
RSFgen  \
-nt 200   -num_stimts 3   -seed 1234567\
-nreps 1 10   -nblock 1 5 \
-nreps 2 10   -nblock 2 5 \
-nreps 3 10   -nblock 3 5 \
-one_file   -prefix RB1
```

■

Just the final, expanded array, is reproduced below:

Program RSFgen Screen Output for Example 3.3.3.1

```
Expanded array:
 0  0  1  1  1  1  1  0  3  3  3  3  3  0  0  0  0  0  2  2
 2  2  2  0  0  0  1  1  1  1  1  0  0  3  3  3  3  3  0  0
 3  3  3  3  3  0  1  1  1  1  1  0  0  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  3  3  3  3  3  0  0  0  0  0  0  0
 0  0  2  2  2  2  2  0  0  0  3  3  3  3  3  1  1  1  1  1
 1  1  1  1  1  0  1  1  1  1  1  0  0  0  0  2  2  2  2  2
 0  1  1  1  1  1  0  2  2  2  2  2  1  1  1  1  1  2  2  2
 2  2  0  2  2  2  2  2  0  0  0  0  1  1  1  1  1  3  3  3
 3  3  0  0  2  2  2  2  2  3  3  3  3  3  0  3  3  3  3  3
 1  1  1  1  1  3  3  3  3  3  0  3  3  3  3  3  0  0  0  0
```

■

**Example 3.3.3.2 Another Random Block Design**

Now, for comparison, we repeat the previous example, but using a different random number seed option -seed 76543521:

Program RSFgen Command Line for Example 3.3.3.2

```
RSFgen  \
-nt 200   -num_stimts 3   -seed 7654321\
-nreps 1 10   -nblock 1 5 \
-nreps 2 10   -nblock 2 5 \
-nreps 3 10   -nblock 3 5 \
-one_file   -prefix RB2
```

■

Program RSFgen Screen Output for Example 3.3.3.2

```
Expanded array:
 2  2  2  2  2  3  3  3  3  3  0  1  1  1  1  1  0  1  1  1
 1  1  0  0  0  0  0  0  3  3  3  3  3  1  1  1  1  1  0  2
 2  2  2  2  1  1  1  1  1  0  0  0  0  1  1  1  1  1  1  1
 1  1  1  0  2  2  2  2  2  0  0  3  3  3  3  3  2  2  2  2
 2  0  0  0  0  1  1  1  1  1  0  0  3  3  3  3  3  0  0  0
 0  3  3  3  3  3  2  2  2  2  2  0  0  3  3  3  3  3  3  3
 3  3  3  0  0  0  2  2  2  2  2  2  2  2  2  2  0  0  0  0
 0  1  1  1  1  1  2  2  2  2  2  0  0  0  0  3  3  3  3  3
 0  0  0  0  0  3  3  3  3  3  1  1  1  1  1  0  2  2  2  2
 2  1  1  1  1  1  2  2  2  2  2  0  0  3  3  3  3  3  0  0
```

■

Note how the random block design of this example differs from that of the previous example. In particular, note that the blocks occur in different locations.

**Example 3.3.3.3 Stimulus Label Permutation**

For our third example, we use the same random number seed option -seed 1234567 as in the first example. However, this time we add the stimulus label permutation option -pseed 16489125.

Program RSFgen Command Line for Example 3.3.3.3

```
RSFgen  \
-nt 200   -num_stimts 3   -seed 1234567\
-nreps 1 10   -nblock 1 5 \
-nreps 2 10   -nblock 2 5 \
-nreps 3 10   -nblock 3 5 \
-pseed 16489125   -one_file   -prefix RB3
```

■

Program RSFgen Screen Output for Example 3.3.3.3

```
Expanded array:
 0  0  3  3  3  3  3  0  1  1  1  1  1  0  0  0  0  0  3  3
 3  3  3  0  0  0  3  3  3  3  3  0  0  3  3  3  3  3  0  0
 1  1  1  1  1  0  2  2  2  2  2  0  0  3  3  3  3  3  1  1
 1  1  1  3  3  3  3  3  1  1  1  1  1  0  0  0  0  0  0  0
 0  0  2  2  2  2  2  0  0  0  3  3  3  3  3  2  2  2  2  2
 1  1  1  1  1  0  1  1  1  1  1  0  0  0  0  1  1  1  1  1
 0  3  3  3  3  3  0  1  1  1  1  1  2  2  2  2  2  2  2  2
 2  2  0  1  1  1  1  1  0  0  0  0  2  2  2  2  2  2  2  2
 2  2  0  0  2  2  2  2  2  3  3  3  3  3  0  1  1  1  1  1
 2  2  2  2  2  3  3  3  3  3  0  2  2  2  2  2  0  0  0  0
```

■

Compare the output for this example with the output from the previous two examples. In particular, note that the blocks in Example 3.3.3.3 occur in exactly the same locations as the blocks in Example 3.3.3.1. However, due to the stimulus label permutation option -pseed, the *labels* of the blocks (i.e., the index numbers for the experimental conditions) have been shuffled. This capability may prove useful for some types of statistical analysis.

### 3.3.4 Markov Chain Designs

The previous examples used the Random Permutation method for creating the experimental designs. That is, the number of repetitions of each stimulus type was fixed; a specific experimental design was created by random permutation of the sequence in which the stimulus events occur. However, for some experimental designs, this approach may not have the needed flexibility. For example, if some types of events must not be allowed to follow other types of events, then additional structure must be introduced into the design specification. For such cases, program RSFgen has the Markov Chain option for creating random experimental designs.

**Example 3.3.4.1 Markov Chain Design**
(This example was suggested by Eli Merriam.)

For this example, there are 6 experimental conditions: A, B, C, D, E, and F. However, there are restrictions on the order in which the different events can occur. Specifically, we have the following requirements:

A can't follow D, E, or F
B can't follow A, B, or C
C can't follow A, B, or C
D can't follow A, B, or C
E can't follow D, E, or F
F can't follow D, E, or F

These conditions can be represented by the following transition probability matrix (TPM), where uniform probabilities are used for the allowed transitions:

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0.33 | 0.00 | 0.00 | 0.00 | 0.33 | 0.34 |
| B | 0.33 | 0.00 | 0.00 | 0.00 | 0.33 | 0.34 |
| C | 0.33 | 0.00 | 0.00 | 0.00 | 0.33 | 0.34 |
| D | 0.00 | 0.33 | 0.33 | 0.34 | 0.00 | 0.00 |
| E | 0.00 | 0.33 | 0.33 | 0.34 | 0.00 | 0.00 |
| F | 0.00 | 0.33 | 0.33 | 0.34 | 0.00 | 0.00 |

Reading the first row of the matrix, we see that state A can be followed by state A (with probability $\frac{1}{3}$), or by state E (with probability $\frac{1}{3}$), or by state F (with probability $\frac{1}{3}$).

Similarly, for rows B, C, D, E, and F. (The numbers have been rounded to 2 decimal places, and the last entry in each row is rounded up so that the row sums equal 1.0.) Looking at the nonzero entries in each column, we see that A can only follow A, B, or C; B can only follow D, E, or F; etc. Therefore, we see that the conditions stated above are satisfied by this Markov chain.

Now, store the above matrix (the numbers only!) as 6 rows and 6 columns into file tpm.mat. The script for executing RSFgen using the Markov chain model is as follows:

Program RSFgen Command Line for Example 3.3.4.1

```
RSFgen \
-nt 200  -num_stimts 6  -seed 123456789 \
-markov tpm.mat  -pzero 0.20  -one_file   -prefix ABCDEF
```

■

Note that the -pzero command lets the user specify the frequency of occurrence of the zero (null) state. For example, to get approximately 20% nulls, use -pzero 0.20.

Program RSFgen Screen Output for Example 3.3.4.1

```
 Program:            RSFgen
 Author:             B. Douglas Ward
 Initial Release:    06 July 1999
 Latest Revision:    27 April 2001

 nt             =  200
 num_stimts     =  6
 seed           =  123456789
 output prefix  =  ABCDEF
 TPM file       =  tpm.mat
 pzero          =  0.200000

TPM matrix:
 0.3300  0.0000  0.0000  0.0000  0.3300  0.3400
 0.3300  0.0000  0.0000  0.0000  0.3300  0.3400
 0.3300  0.0000  0.0000  0.0000  0.3300  0.3400
 0.0000  0.3300  0.3300  0.3400  0.0000  0.0000
 0.0000  0.3300  0.3300  0.3400  0.0000  0.0000
 0.0000  0.3300  0.3300  0.3400  0.0000  0.0000
```

```
Markov chain time series:
 0  0  1  5  0  3  6  3  1  1  1  5  3  6  4  0  0  4  0  0
 3  0  0  1  5  3  6  3  5  3  5  3  6  0  4  0  2  1  6  0
 4  2  0  5  2  0  5  0  3  1  5  2  0  1  0  1  0  5  0  2
 5  4  3  6  2  5  0  3  5  0  3  1  1  6  3  0  6  2  1  0
 6  2  6  0  4  2  1  6  4  3  5  2  1  6  0  0  2  5  4  4
 3  5  4  2  6  4  3  5  3  5  3  0  6  3  5  2  1  0  0  5
 3  0  6  2  5  3  6  3  5  3  6  2  6  4  0  2  1  1  6  3
 1  1  1  5  4  2  1  1  5  4  4  0  4  0  2  1  0  6  3  6
 0  4  4  0  3  0  5  3  0  5  2  0  5  3  6  4  3  1  0  6
 2  0  0  0  0  5  2  6  4  3  6  2  6  3  0  0  6  4  2  0
```

■

As before, the coding is A=1, B=2, C=3, etc. The output is stored as 6 columns of 0's and 1's into file ABCDEF.1D.

**Example 3.3.4.2 Markov Chain Block Design**

This example is similar to the previous example, only now we specify the block length for each of the 6 experimental conditions. Specifically, we require that condition A occur in blocks of length 1, condition B occurs in blocks of length 2, ..., condition F occurs in blocks of length 6. The state transition probability matrix is the same as in Example 3.3.4.1. The script for generating this Markov Chain block design is as follows:

Program RSFgen Command Line for Example 3.3.4.2

```
RSFgen \
-nt 200  -num_stimts 6  -seed 123456789 \
-nblock 1 1  -nblock 2 2  -nblock 3 3  \
-nblock 4 4  -nblock 5 5  -nblock 6 6 \
-markov tpm.mat  -pzero 0.20  -one_file   -prefix MCblock
```

■

The RSFgen screen output for this example is:

Program RSFgen Screen Output for Example 3.3.4.2

```
Program:            RSFgen
Author:             B. Douglas Ward
Initial Release:    06 July 1999
Latest Revision:    06 March 2002

nt             =  200
num_stimts     =  6
seed           =  123456789
output prefix  =  MCblock
TPM file       =  tpm.mat
pzero          =  0.200000
```

```
 nblock[1]  =  1
 nblock[2]  =  2
 nblock[3]  =  3
 nblock[4]  =  4
 nblock[5]  =  5
 nblock[6]  =  6

TPM matrix:
 0.3300  0.0000  0.0000  0.0000  0.3300  0.3400
 0.3300  0.0000  0.0000  0.0000  0.3300  0.3400
 0.3300  0.0000  0.0000  0.0000  0.3300  0.3400
 0.0000  0.3300  0.3300  0.3400  0.0000  0.0000
 0.0000  0.3300  0.3300  0.3400  0.0000  0.0000
 0.0000  0.3300  0.3300  0.3400  0.0000  0.0000

Markov chain time series:
 0  0  1  5  0  3  6  3  1  1  1  5  3  6  4  0  0  4  0  0
 3  0  0  1  5  3  6  3  5  3  5  3  6  0  4  0  2  1  6  0
 4  2  0  5  2  0  5  0  3  1  5  2  0  1  0  1  0  5  0  2
 5  4  3  6  2  5  0  3  5  0  3  1  1  6

Expanded array:
 0   0   1   5   5   5   5   5   0   3   3   3   6   6   6   6   6   6   3   3
 3   1   1   1   5   5   5   5   5   3   3   3   6   6   6   6   6   6   4   4
 4   4   0   0   4   4   4   4   0   0   3   3   3   0   0   1   5   5   5   5
 5   3   3   3   6   6   6   6   6   6   3   3   3   5   5   5   5   5   3   3
 3   5   5   5   5   5   3   3   3   6   6   6   6   6   6   0   4   4   4   4
 0   2   2   1   6   6   6   6   6   6   0   4   4   4   4   2   2   0   5   5
 5   5   5   2   2   0   5   5   5   5   5   0   3   3   3   1   5   5   5   5
 5   2   2   0   1   0   1   0   5   5   5   5   5   0   2   2   5   5   5   5
 5   4   4   4   4   3   3   3   6   6   6   6   6   6   2   2   5   5   5   5
 5   0   3   3   3   5   5   5   5   5   0   3   3   3   1   1   6   6   6   6
```

■

As before, the coding is A=1, B=2, C=3, etc. The output is stored as 6 columns of 0's and 1's into file MCblock.1D. Note that the last block (corresponding to condition F) is of length 4 rather than 6. This block was truncated in order to satisfy the specified time series length of nt=200.

## 3.4 Evaluation of the Experimental Design (continued)

In this section, we continue our previous discussion about evaluation of the experimental design. By using the output of RSFgen as input to 3dDeconvolve with the -nodata option, we can iteratively evaluate the experimental design(s). This not only helps to avoid multicollinearity; it can be used to create experimental designs that are optimized for estimation of the unknown parameters. We conclude this section with examples of statistical power estimation.

### 3.4.1 Parameter Estimation Accuracy

**Example 3.4.1.1 Random Event Design**

Consider an event-related design, in which there are two types of events, A and B. Since there are $nt = 300$ time points, we will initially divide these time points equally between event A, event B, and the null event (or "rest") condition. This design can be created with the following command line:

Program RSFgen Command Line for Example 3.4.1.1

```
RSFgen \
-nt 300   -num_stimts 2 \
-nreps 1 100   -nblock 1 1 \
-nreps 2 100   -nblock 2 1 \
-seed 123456789  -one_file   -prefix Event
```

■

To investigate the accuracy of the parameter estimates, the output stimulus file Event.1D is used as input to program 3dDeconvolve, using the -nodata option, as indicated by the following command line:

Program 3dDeconvolve Command Line for Example 3.4.1.1:

```
3dDeconvolve \
-nodata   -nlast 299   -polort 1   -num_stimts 2 \
-stim_file 1 'Event.1D[0]'  -stim_label 1 'Event A'  -stim_maxlag 1 4 \
-stim_file 2 'Event.1D[1]'  -stim_label 2 'Event B'  -stim_maxlag 2 4 \
-glt 1 A.mat     -glt_label 1 'A' \
-glt 1 B.mat     -glt_label 2 'B' \
-glt 1 A+B.mat   -glt_label 3 'A+B' \
-glt 1 A-B.mat   -glt_label 4 'A-B'
```

■

Note that the above command line includes 4 GLT's, which (approximately) test for:

IRF area for Event A $\neq$ 0,
IRF area for Event B $\neq$ 0,
IRF area for Event A + IRF area for Event B $\neq 0$, and
IRF area for Event A – IRF area for Event B $\neq 0$.

are specified by the following GLT matrix files.

GLT matrix files:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A.mat : | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| B.mat : | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| A+B.mat : | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| A-B.mat : | 0 | 0 | 1 | 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 |

■

Store the above numbers as single *rows* into files A.mat, B.mat, A+B.mat, A–B.mat, and then execute the above command line.

The program output includes the *normalized* variance-covariance matrix $(\mathbf{X}^t\mathbf{X})^{-1}$. For further discussion, see Section 1.2.7.

Program 3dDeconvolve Screen Output for Example 3.4.1.1

```
  Program:            3dDeconvolve
  Author:             B. Douglas Ward
  Initial Release:    02 Sept 1998
  Latest Revision:    11 August 2000

(X'X) inverse matrix:
```

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| .065 | −.000 | −.013 | −.014 | −.013 | −.013 | −.012 | −.013 | −.014 | −.014 | −.014 | −.014 |
| −.000 | .000 | .000 | .000 | .000 | .000 | .000 | .000 | .000 | .000 | .000 | .000 |
| −.013 | .000 | .021 | .000 | .001 | .000 | −.000 | .010 | .001 | .000 | .003 | −.000 |
| −.014 | .000 | .000 | .021 | .000 | .001 | .000 | −.001 | .010 | .001 | .000 | .003 |
| −.013 | .000 | .001 | .000 | .021 | .000 | .001 | .001 | −.000 | .010 | .001 | −.000 |
| −.013 | .000 | .000 | .001 | .000 | .021 | .000 | .000 | .001 | −.000 | .011 | .001 |
| −.012 | .000 | −.000 | .000 | .001 | .000 | .021 | .002 | .000 | .001 | −.001 | .010 |
| −.013 | .000 | .010 | −.001 | .001 | .000 | .002 | .021 | .002 | .000 | −.001 | .000 |
| −.014 | .000 | .001 | .010 | −.000 | .001 | .000 | .002 | .021 | .001 | .000 | −.001 |
| −.014 | .000 | .000 | .001 | .010 | −.000 | .001 | .000 | .001 | .021 | .001 | .000 |
| −.014 | .000 | .003 | .000 | .001 | .011 | −.001 | −.001 | .000 | .001 | .022 | .001 |
| −.014 | .000 | −.000 | .003 | −.000 | .001 | .010 | .000 | −.001 | .000 | .001 | .021 |

```
Stimulus:  Event A
 h[0]  norm.  std.  dev.  =  0.1438
 h[1]  norm.  std.  dev.  =  0.1445
 h[2]  norm.  std.  dev.  =  0.1436
 h[3]  norm.  std.  dev.  =  0.1438
 h[4]  norm.  std.  dev.  =  0.1438

Stimulus:  Event B
 h[0]  norm.  std.  dev.  =  0.1451
 h[1]  norm.  std.  dev.  =  0.1448
 h[2]  norm.  std.  dev.  =  0.1446
 h[3]  norm.  std.  dev.  =  0.1466
 h[4]  norm.  std.  dev.  =  0.1461

General Linear Test:  A
 LC[0]  norm.  std.  dev.  =  0.3361
```

```
General Linear Test: B
 LC[0] norm. std. dev. = 0.3408

General Linear Test: A+B
 LC[0] norm. std. dev. = 0.5994

General Linear Test: A-B
 LC[0] norm. std. dev. = 0.3147
```

■

From the above output, we see that the standard deviation of the estimate for each of the IRF coefficients is about $0.14\sigma$, where $\sigma$ is the square root of the measurement variance for a particular voxel.

To see how the accuracy depends on the number of repetitions, the above analysis was repeated using $r = 140$ repetitions for each stimulus. All commands remain the same as above, except that the -nreps k 100 command for program RSFgen was replaced with -nreps k 140, for k=1, 2. The output from program 3dDeconvolve indicates that, by increasing the number of repetitions from 100 to 140, the standard deviation of the regressor coefficients increases from $0.14\sigma$ to $0.25\sigma$.

Continuing in this manner, by varying the experimental design inputs, one can determine the sensitivity of the regression coefficient estimation accuracy to the design parameters. Of course, the accuracy of these estimates depends on the validity of the assumptions involved (e.g., that the system is linear and time-invariant, with independent errors, etc.). Also, other external considerations may be involved in deciding upon the ultimate experimental design. Nevertheless, this analysis tool should prove useful in arriving at an acceptable experimental design.

**Example 3.4.1.2 Random Block Design**

For contrast with Example 3.4.1.1, here we assume that conditions A and B occur in blocks of length 10 TR, with 10 repetitions of each during the run. This experimental design was created by the following command line:

Program RSFgen Command Line for Example 3.4.1.2

```
RSFgen \
-nt 300   -num_stimts 2 \
-nreps 1 10   -nblock 1 10 \
-nreps 2 10   -nblock 2 10 \
-seed 123456789  -one_file   -prefix Block
```

■

The output stimulus function file Block.1D was then used as input to program 3dDeconvolve, using the -nodata option:

Program 3dDeconvolve Command Line for Example 3.4.1.2:

```
3dDeconvolve \
-nodata   -nlast 299   -polort 1   -num_stimts 2 \
-stim_file 1 'Block.1D[0]'  -stim_label 1 'Block A'  -stim_maxlag 1 4 \
-stim_file 2 'Block.1D[1]'  -stim_label 2 'Block B'  -stim_maxlag 2 4 \
-glt 1 A.mat     -glt_label 1 'A' \
-glt 1 B.mat     -glt_label 2 'B' \
-glt 1 A+B.mat   -glt_label 3 'A+B' \
-glt 1 A-B.mat   -glt_label 4 'A-B'
```

■

The following shows the 3dDeconvolve evaluation of the random block design. Note that, using the random block design, the accuracy of the estimated regression coefficients has significantly decreased relative to that of the previous random event design, as indicated by the higher norm. std. dev.'s.

Program 3dDeconvolve Screen Output for Example 3.4.1.2

```
Program:            3dDeconvolve
Author:             B. Douglas Ward
Initial Release:    02 Sept 1998
Latest Revision:    11 August 2000
```

(X'X) inverse matrix:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| .031 | −.000 | −.006 | −.002 | −.002 | −.001 | −.006 | −.008 | −.002 | −.003 | −.002 | −.008 |
| −.000 | .000 | .000 | .000 | .000 | .000 | −.000 | .000 | .000 | .000 | .000 | .000 |
| −.006 | .000 | .057 | −.047 | −.002 | .000 | .004 | .005 | .001 | .004 | −.005 | −.000 |
| −.002 | .000 | −.047 | .096 | −.045 | −.003 | .000 | .004 | −.002 | −.002 | .008 | −.005 |
| −.002 | .000 | −.002 | −.045 | .096 | −.045 | −.002 | .001 | .003 | −.002 | −.002 | .004 |
| −.001 | .000 | .000 | −.003 | −.045 | .096 | −.047 | −.005 | .006 | .003 | −.002 | .001 |
| −.006 | −.000 | .004 | .000 | −.002 | −.047 | .058 | −.001 | −.005 | .002 | .004 | .006 |
| −.008 | .000 | .005 | .004 | .001 | −.005 | −.001 | .066 | −.055 | .001 | −.003 | .003 |
| −.002 | .000 | .001 | −.002 | .003 | .006 | −.005 | −.055 | .112 | −.055 | .003 | −.003 |
| −.003 | .000 | .004 | −.002 | −.002 | .003 | .002 | .001 | −.055 | .112 | −.055 | .001 |
| −.002 | .000 | −.005 | .008 | −.002 | −.002 | .004 | −.003 | .003 | −.055 | .112 | −.055 |
| −.008 | .000 | −.000 | −.005 | .004 | .001 | .006 | .003 | −.003 | .001 | −.055 | .066 |

```
Stimulus:  Block A
 h[0]  norm.  std.  dev.  =  0.2386
 h[1]  norm.  std.  dev.  =  0.3104
 h[2]  norm.  std.  dev.  =  0.3099
 h[3]  norm.  std.  dev.  =  0.3101
 h[4]  norm.  std.  dev.  =  0.2407
```

```
Stimulus:  Block B
 h[0]  norm.  std.  dev.  =  0.2580
 h[1]  norm.  std.  dev.  =  0.3350
 h[2]  norm.  std.  dev.  =  0.3348
 h[3]  norm.  std.  dev.  =  0.3354
 h[4]  norm.  std.  dev.  =  0.2563

General Linear Test:  A
 LC[0]  norm.  std.  dev.  =  0.1784

General Linear Test:  B
 LC[0]  norm.  std.  dev.  =  0.1801

General Linear Test:  A+B
 LC[0]  norm.  std.  dev.  =  0.3167

General Linear Test:  A-B
 LC[0]  norm.  std.  dev.  =  0.1680
```

■

### 3.4.2 Statistical Power Calculations

We again consider the multiple linear regression model:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where $\mathbf{y}$ = vector of measured data, $\mathbf{X}$ = experimental design matrix, $\boldsymbol{\beta}$ = (unknown) parameter vector, and $\boldsymbol{\varepsilon}$ = error vector, with $\varepsilon_n \overset{iid}{\sim} N(0, \sigma^2)$. The least squares estimate of $\boldsymbol{\beta}$ is $\mathbf{b}$:

$$\mathbf{b} = \widehat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

The variance for the estimated parameter vector $b$ is:

$$var(\mathbf{b}) = \sigma^2(\mathbf{X}'\mathbf{X})^{-1}$$

where $\sigma^2$ is the measurement error variance. Since $\sigma^2$ is unknown, it is estimated using the sample error variance $MSE$:

$$\begin{aligned} s^2(\mathbf{b}) &= MSE \cdot (\mathbf{X}'\mathbf{X})^{-1} \\ &= s^2 \cdot (\mathbf{X}'\mathbf{X})^{-1} \end{aligned}$$

where the sample standard deviation $s = \sqrt{MSE}$. Suppose we are interested in testing a (possibly vector-valued) linear combination of the $\mathbf{b}$ parameters:

$$\mathbf{L} = \mathbf{C}\mathbf{b}$$

Then the variance of $\mathbf{L}$ is given by:

$$\begin{aligned} var(\mathbf{L}) &= var(\mathbf{Cb}) \\ &= \mathbf{C}\, var(\mathbf{b})\, \mathbf{C}' \\ &= \sigma^2 \mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}' \end{aligned}$$

and the estimated variance of $\mathbf{L}$ is:

$$s^2(\mathbf{L}) = s^2 \cdot \mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}'$$

Now, let $l$ be the $i$th linear combination of the parameters:

$$l = \mathbf{L}_i$$

and let $d$ be the square root of the $i$th diagonal element of the $\mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}'$ matrix:

$$d = \sqrt{\left[\mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}'\right]_{i,i}}$$

then the standard deviation for the linear combination $l$ is given by:

$$s(l) = s \cdot d$$

Now, $l$ is an estimate for the true value $\theta$. Suppose that we wish to test:

$$\begin{aligned} & H_o: && \theta \leq 0 \\ \text{vs.} \quad & H_a: && \theta > 0 \end{aligned}$$

The question is: what is the probability of rejecting $H_o$ for $\theta > 0$? This probability is the statistical power of the test. In order to guard against false positives, we will reject $H_o$ only if the measured value $l$ is some $k$ standard deviations greater than zero; otherwise, we accept $H_o$. Note that $k$ is a constant, which is used to set the statistical significance of the test. The power of the test is calculated thus (see Ref.[4]):

$$\begin{aligned} Power_\theta(l) &= \Pr\left(\frac{l}{s \cdot d} > k\right) \\ &= \Pr\left(\frac{l-\theta}{s \cdot d} > k - \frac{\theta}{s \cdot d}\right) \\ &= \Pr\left(Z > k - \frac{\theta}{s \cdot d}\right) \end{aligned}$$

$$\text{where } Z = \frac{l-\theta}{s \cdot d} \text{ is approx. } N(0,1)$$

**Example 3.4.2.1 Random Event Design**

We continue the analysis of Example 3.4.1.1. Suppose that we wish to estimate the statistical power for detection of the difference in area under the IRF curve. We will assume that condition A has IRF area 10 units greater than condition B; i.e., $\theta = Area(A) - Area(B) = 10.0$. Now, from Example 3.4.1.1, we have:

```
General Linear Test:  A-B
 LC[0]  norm.  std.  dev.  =  0.3147
```

So, we will let $d = 0.3147$. Also, suppose that from previous experience, the measurement error standard deviation is known to be $s = 15.0$. For protection against false positives, we will use $k = 3.0$ (corresponding to $\alpha = 0.0013$). The probability of detecting the assumed difference in IRF areas is estimated by:

$$\begin{aligned} Power_\theta(l) &= \Pr\left(Z > k - \frac{\theta}{s \cdot d}\right) \\ &= \Pr\left(Z > 3.0 - \frac{10.0}{(15.0)(0.3147)}\right) \\ &= \Pr\left(Z > 0.882\right) \\ &= 0.189 \end{aligned}$$

**Example 3.4.2.2 Random Block Design**

We will repeat the above analysis, but using the random block design from Example 3.4.1.2. From that example, we obtained:

```
General Linear Test:  A-B
 LC[0]  norm.  std.  dev.  =  0.1680
```

Therefore, using $d = 0.1680$, with the other parameters the same as in Example 3.4.2.1, we estimate the statistical power thus:

$$\begin{aligned} Power_\theta(l) &= \Pr\left(Z > k - \frac{\theta}{s \cdot d}\right) \\ &= \Pr\left(Z > 3.0 - \frac{10.0}{(15.0)(0.168)}\right) \\ &= \Pr\left(Z > -0.968\right) \\ &= 0.834 \end{aligned}$$

Careful examination of Examples 3.4.1.1, 3.4.1.2, 3.4.2.1, and 3.4.2.2 may help explain the observed and (seemingly) contradictory results that event-related designs yield more accurate estimates of the system impulse response functions, whereas block-type designs often yield more active voxels. The above examples should highlight the fact that the optimum experimental design depends, among other things, on the particular parameters (or effects) that are of interest.

A word of *caution*: The theoretical estimates for statistical power should be used as a guide only. Many assumptions are involved in the above analysis, and many complicating factors have not been taken into account. As usual, it is assumed that the system is linear and time-invariant, with independent errors. No allowance is made for confounding of the input stimulus with subject motion or with natural physiological processes (e.g., respiration). Also, no allowance is made for habituation or anticipatory effects. These later points, in particular, would seem to argue in favor of event-related designs, even though they are not explicitly incorporated into the mathematical analysis.

# 4 Program 3dConvolve

## 4.1 Purpose

As the name implies, program 3dConvolve performs basically the inverse operation of program 3dDeconvolve. There are essentially four types of input to program 3dConvolve:

- One or more input stimulus functions. These can be the same .1D time series files that are used as input to program 3dDeconvolve.
- One or more system impulse response functions (IRF's) for each voxel in the dataset. There should be a 1-1 correspondence between the number of .1D stim functions and the number of IRF input datasets. The IRF coefficients can be input from the bucket dataset output of 3dDeconvolve, or from the 3d+time datasets created by the 3dDeconvolve -iresp command, or they can be "manually" created 3d+time datasets. Yet another alternative is to enter a single .1D time series file for each IRF. In the latter case, the output will be a single .1D time series file, instead of a 3d+time dataset.
- The baseline parameters for each voxel (say, from the 3dDeconvolve bucket dataset output).
- System noise (an optional input) can be added to the output response.

Program 3dConvolve convolves the input stimulus function(s) with the IRF(s) for each voxel, to which is added the baseline offset and the system noise (if included in the model); the predicted measurement time series (for each voxel) is then written to a 3d+time dataset. As an alternative, if the -input1D option is used, then all inputs are .1D time series files, and the output is a single .1D time series.

There are several possible applications for this program:

- Prediction of the system response to hypothetical inputs, given the system response to previous inputs. This may be useful in experimental design.
- Evaluation of the 3dDeconvolve parameter estimation accuracy. This is not possible using "real" data, since the actual parameter values are unknown. However, by using 3dConvolve to produce artificial time series corresponding to known parameter values, one can compare the "actual" parameter values with the parameter estimates obtained from program 3dDeconvolve.
- Validation of the 3dDeconvolve statistical output. Using program 3dConvolve, one can generate 3d+time datasets corresponding to either the null hypothesis (signal is not present), or the alternative hypothesis (signal is present). By examining the output from program 3dDeconvolve, one can then evaluate the statistical significance level (probability of a false positive) or the statistical power (probability of a true detection).

## 4.2 Theory

As in Section 1.2.2, we model the system response as the convolution of the input stimulus function $f(t)$ with the system impulse response function (IRF) $h(t)$, represented thus:

$$y(t) = f(t) \otimes h(t)$$

If multiple types of stimuli are used (say, $m$ stimulus functions), then the system response is modeled as the sum of the convolutions of each individual stimulus function with their corresponding IRF:

$$y(t) = f_1(t) \otimes h_1(t) + f_2(t) \otimes h_2(t) + \cdots + f_m(t) \otimes h_m(t)$$

The measured data is modeled as the sum of a baseline (e.g., a constant plus linear trend) plus the system response, plus measurement noise:

$$\begin{aligned} z(t) &= b_0 + b_1 t + y(t) + \varepsilon(t) \\ &= b_0 + b_1 t + f_1(t) \otimes h_1(t) + f_2(t) \otimes h_2(t) + \cdots + f_m(t) \otimes h_m(t) + \varepsilon(t) \end{aligned}$$

For program 3dConvolve, the inputs are the baseline parameters $b_0$ and $b_1$; the impulse response functions $h_1(t)$, $h_2(t)$,...,$h_m(t)$; the input stimulus functions $f_1(t)$, $f_1(t)$,..., $f_m(t)$; and (possibly) the measurement noise vector $\varepsilon(t)$. Program 3dConvolve output is the predicted measurement data $z(t)$.

## 4.3 Usage

### 4.3.1 Syntax

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

**3dConvolve** [**-input** *fname* | **-input1D** ] [**-mask** *mname*] [**-censor** *cname*] [**-concat** *rname*] [**-nfirst** *fnum*] [**-nlast** *lnum*] [**-polort** *pnum*] [**-base_file** *bname*] **-num_stimts** *num* **-stim_file** *k sname* [**-stim_minlag** *k m*] [**-stim_maxlag** *k n*] [**-stim_nptr** *k p*] **-iresp** *k iprefix* [**-errts** *eprefix*] [**-sigma** *s*] [**-seed** *d*] [**-xout**] [**-output** *tprefix*]

The different command line options are explained below.

### 4.3.2 Options

**-input *fname***

The -input command specifies that *fname* is the filename of the *AFNI* 3d+time data set to be used as a template for the program 3dConvolve output. The -input command is mandatory *except* when the -input1D command is used in its place.

**-input1D**
The -input1D command specifies that only .1D time series data files will be used as input for program 3dConvolve. That is, instead of 3d+time datasets, the inputs are given only for a single voxel. If this option is used, the predicted measurement time series is written to the screen and (optionally) to a .1D output file. This option allows prediction of the measured response time series for a single voxel.

**-mask *mname***
The optional -mask command specifies that *mname* is the filename of the *AFNI* 3d dataset to be used for "masking " the input data. That is, if a voxel in the mask dataset has value zero, then the corresponding voxel in the 3d+time input datasets will be ignored for computational purposes. All output corresponding to that particular voxel will be set to zero. If the mask dataset represents the brain, i.e., if the mask contains 1's only at locations inside the brain, and 0's at locations outside the brain, this will greatly improve the program execution speed. Of course, the mask dataset must have the same voxel dimensions as the input 3d+time template dataset.

**-censor *cname***
The optional -censor command is used to specify that *cname* is a .1D file, equal in length to that of the input 3d+time dataset, consisting of 1's and 0's. At time points corresponding to 0's in file *cname*, the output data values are replaced with the corresponding values from the input 3d+time template dataset.

**-concat *rname***
The optional -concat command specifies that *rname* is the filename of the *AFNI* .1D time series data file containing a list of the starting points for each individual run within a concatenated dataset.

**-nfirst *fnum***
The optional -nfirst command specifies that $fnum$ is the number of the first time point to be calculated by the convolution procedure. (Note: the first image in the dataset is numbered 0.) At time points prior to $fnum$, the output data values are replaced with the corresponding values from the input 3d+time template dataset. The default value is $fnum$ = maximum of the $maxlag$ values. See option -stim_maxlag below.

**-nlast *lnum***
The optional -nlast command specifies that *lnum* is the number of the last time point to be calculated by the convolution procedure. (Note: the first image in the dataset is numbered 0). At time points after $l\,num$, the output data values are replaced with the corresponding values from the input 3d+time template dataset. The default value is *lnum* = number of the last image in the dataset. (Program 3dinfo can be used to print out relevant information about a dataset.)

**-polort *pnum***
The optional -polort command specifies that *pnum* ($pnum = 0, 1, 2, ...$) is the degree of the polynomial in the baseline model. The default value is $pnum = 1$ (corresponding to the baseline model: $Z_n = \gamma_0 + \gamma_1 n$; i.e., the signal is a constant plus linear trend).

Note: The command "-polort -1" can be used to specify *no* basline model parameters; in this case, the baseline model is: $Z_n = 0$.

**-base_file *bname***
The -base_file command specifies that *bname* is the name of the 3d dataset which contains the baseline parameters for each voxel. Be sure to use the "[ ]" sub-brick selectors to indicate which specific sub-bricks contain the baseline parameters. Note: The number of baseline parameter sub-bricks must be *exactly* one more than the degree of the baseline polynomial specified with the -polort command. If the command "-polort -1" is used, do *not* use the -base_file command.

**-num_stimts *num***
The mandatory -num_stimts is used to indicate that *num* input stimulus time series will be used. Note: the -num_stimts command *must precede* the following commands.

**-stim_file *k sname***
The mandatory -stim_file command specifies that *sname* is the filename of the .1D time series representing the $k$th input stimulus function. For multi-column .1D files, this command has the alternative format:

**-stim_file *k "sname[j]"***
In this case, the -stim_file command specifies that the $k$th input stimulus function is contained in column $j$ of file *sname*. Note: The column numbering begins with 0; i.e., the first column corresponds to $j = 0$, etc. Also, note that the square brackets around the column index must be enclosed within quotation marks.

**-stim_minlag *k m***
The optional -stim_minlag command specifies that the minimum time lag is $m$ for the input impulse response function corresponding to the $k$th input stimulus. The default value is $m = 0$.

**-stim_maxlag *k n***
The optional -stim_maxlag command specifies that the maximum time lag is $n$ for the input impulse response function corresponding to the $k$th input stimulus. The default value is $n = 0$. Note that for each input stimulus, it is required that: $\min lag \leq \max lag$.

**-stim_nptr *k p***
The optional -stim_nptr command specifies that there are $p$ ($p = 1, 2, 3, ...$) time points in the $k$th input stimulus for each TR. The default value is $p = 1$. If the input 3d+time dataset contains $N$ points, then the $k$th input stimulus must contain at least $p \times N$ points.

If $p > 1$, then the user *must* align all input slices to 0 time offset beforehand (see program 3dTshift). Note: The IRF time limits specified with -stim_minlag and -stim_maxlag are in multiples of $TR/p$. Therefore, the commands:

-stim_minlag $k$ $m$
-stim_maxlag $k$ $n$
-stim_nptr $k$ $p$

indicate that the $k$th IRF extends from $m \times TR/p$ to $n \times TR/p$.

**-iresp *k iname*** The mandatory -iresp command instructs program 3dConvolve to read the impulse response function corresponding to the $k$th input stimulus. That is, the impulse response function for each voxel is read as a time series of length $\max lag - \min lag + 1$ from an *AFNI* 3d (+time) dataset. The input dataset has filename iname. Note: If the -input1D option is used, then the single impulse response function is read from the .1D file iname. For multi-column .1D files, this command has the alternative format:

**-iresp *k "iname[j]"***

In this case, the -iresp command specifies that the $k$th impulse response function is contained in column $j$ of file *iname*. Note: The column indexing begins with 0; i.e., the first column corresponds to $j = 0$, etc. Also note that the square brackets around the column index must be enclosed within quotation marks.

**-errts *ename***

The optional -errts command instructs program 3dConvolve to read the residual error time series, for each voxel, from an *AFNI* 3d+time dataset. These errors are then added to the full model (i.e., predicted measurement = full model fit + residual error). The residual error input dataset has filename ename. Note: If the -input1D option is used, then the single residual error time series is read from the .1D file iname.

**-sigma *s***

The optional -sigma command is used to indicate that $s$ = standard deviation of white Gaussian noise, which is added to the predicted response. The default value is $s = 0$.

**-seed *d***

The optional -seed command is used to indicate that $d$ = seed for the random number generator. The default value is $d = 1234567$.

**-xout**

The optional -xout command is used to write the experimental design matrix $\mathbf{X}$ to the screen. This may help the user to understand what the program is doing. Note that this option effects screen output only, and does not alter the content of the output dataset.

**-output** ***tprefix***

The -output command instructs program 3dConvolve to save the predicted measurement data time series for each voxel. The predicted measurement time series are stored into an *AFNI* 3d+time dataset with prefix filename tprefix.

However, if -output is used in conjunction with the -input1D command, then the single predicted measurement time series will be written to file tprefix.1D. Note: This will automatically overwrite a pre-existing tprefix.1D file.

## 4.4 Examples

**Example 4.4.1 Single Stimulus; No Noise; 1d Inputs**

As the inverse problem to Example 1.4.2.3, we will use program 3dConvolve to compute the single time series which results from the convolution

$$w(t) = b_0 + b_1 t + g(t) \otimes h(t)$$

where the baseline parameters are:

$$\begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} 100.0 \\ 1.0 \end{bmatrix}$$

Save the above as a single column of two numbers into file Base.1D. The input stimulus function $g(t)$ is the same as given in that example:

$$g(t) = \{\ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \}$$

We will assume that this stim function has already been saved in file g.1D (if not, then do so now). The impulse response function $h(t)$ from that example is as follows:

$$h(t) \quad = \quad \{ \quad 0.0 \quad 5.0 \quad 10.0 \quad 5.0 \quad 2.0 \quad \}$$

Save the above IRF as a single column of numbers into file h.1D. Now, execute the following script:

Program 3dConvolve Command Line for Example 4.4.1

```
3dConvolve  \
-input1D   -nfirst 0   -nlast 19   -polort 1  \
-base_file Base.1D  \
-num_stimts 1 \
-stim_file 1 g.1D   -stim_maxlag 1 4  \
-iresp 1 h.1D
```

■

Verify that the output to the screen is identical to the $w(t)$ time series displayed in Example 1.4.2.3:

$$\begin{array}{lllllllllllll} w(t) & = & \{ & 100 & 106 & 117 & 118 & 111 & 112 & 121 & 127 & 125 & 116 & \\ & & & 117 & 121 & 117 & 120 & 124 & 125 & 133 & 137 & 135 & 126 & \} \end{array}$$

Pay particular attention to the difference between this example and Example 1.4.2.3. In the earlier example, the inputs were $g(t)$ and $w(t)$; the output consisted of the baseline parameters $b_0$ and $b_1$, and impulse response function $h(t)$. In the current example, the inputs are $g(t)$, baseline parameters $b_0$ and $b_1$, and impulse response function $h(t)$. The output is the predicted measurement $w(t)$.

**Example 4.4.2 Single Stimulus; External Noise; 1d Inputs**

This example is the same as the previous example, except that we specify the measurement error at each time point. Referring to Example 1.4.2.4, let the measurement errors be given by:

$$\begin{aligned}\varepsilon(t) &= wn(t) - w(t) \\ &= \begin{array}{l}\{\ -0.22\ \ -0.54\ \ -0.70\ \ \ 5.51\ \ -2.40\ \ -0.99\ \ -0.16\ \ -0.58\ \ -1.89\ \ 0.85 \\ \ \ \ -2.45\ \ -2.82\ \ \ 0.58\ \ -1.07\ \ \ 1.01\ \ \ 1.21\ \ \ 2.23\ \ \ 3.22\ \ \ 3.75\ \ 1.28\ \}\end{array}\end{aligned}$$

If these errors are stored in file eps.1D, then they can be added to the predicted measurement using the -errts command, as illustrated below:

Program 3dConvolve Command Line for Example 4.4.2

```
3dConvolve  \
-input1D   -nfirst 0   -nlast 19   -polort 1  \
-base_file Base.1D  \
-num_stimts 1 \
-stim_file 1 g.1D   -stim_maxlag 1 4  \
-iresp 1 h.1D  \
-errts eps.1D
```

■

Verify that the output to the screen is equal to the $wn(t)$ time series displayed in Example 1.4.2.4, i.e.:

$$\begin{aligned}wn(t) &= \\ \{\ &99.78\ \ 105.46\ \ 116.30\ \ 123.51\ \ 108.60\ \ 111.01\ \ 120.84\ \ 126.42\ \ 123.11\ \ 116.85 \\ &114.55\ \ 118.18\ \ 117.58\ \ 118.93\ \ 125.01\ \ 126.21\ \ 135.23\ \ 140.22\ \ 138.75\ \ 127.28\ \}\end{aligned}$$

**Example 4.4.3 Multiple Stimuli; No Noise; 1d Inputs**

In Example 1.4.3.4, there were three separate stimulus time series functions, corresponding to the times of presentation for the three different word categories.

$$\begin{aligned}r(t) &= \{\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ \} \\ m(t) &= \{\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ \} \\ e(t) &= \{\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ \}\end{aligned}$$

If you haven't already done so, store these rows as single columns into files Random.1D, Markov.1D, and English.1D, respectively. As before, let the 3 point IRF's be given by:

$$\begin{array}{rcl} h_R(t) & = & \{ \ 2 \ \ 7 \ \ 5 \ \} \\ h_M(t) & = & \{ \ 1 \ \ 4 \ \ 6 \ \} \\ h_E(t) & = & \{ \ 3 \ \ 9 \ \ 2 \ \} \end{array}$$

Store these IRF's as 3 columns of 3 numbers each into file IRF.1D.

Suppose that we model the system response as the sum of the convolutions of these IRF's with their corresponding stimulus functions:

$$y(t) = r(t) \otimes h_R(t) + m(t) \otimes h_M(t) + e(t) \otimes h_E(t)$$

Now, model the measured data as a constant (100) plus slope (1) plus the system response, e.g.:

$$\begin{array}{rcl} z(t) & = & b_0 + b_1 t + y(t) \\ & = & 100 + 1 \cdot t + r(t) \otimes h_R(t) + m(t) \otimes h_M(t) + e(t) \otimes h_E(t) \end{array}$$

The following command line can be used to calculate $z(t)$:

Program 3dConvolve Command Line for Example 4.4.3

```
3dConvolve  \
-input1D   -nfirst 0   -nlast 19   -polort 1  \
-base_file Base.1D  \
-num_stimts 3 \
-stim_file 1 Random.1D   -stim_maxlag 1 2  \
-stim_file 2 Markov.1D   -stim_maxlag 2 2  \
-stim_file 3 English.1D  -stim_maxlag 3 2  \
-iresp 1 'IRF.1D[0]'  \
-iresp 2 'IRF.1D[1]'  \
-iresp 3 'IRF.1D[2]'
```

■

Verify that the program output agrees with the input data from Example 1.4.3.4:

$$\begin{array}{rcl} z(t) & = & \{ \ 100 \ \ 103 \ \ 110 \ \ 115 \ \ 119 \ \ 108 \ \ 110 \ \ 116 \ \ 119 \ \ 118 \\ & & \ \ \ 117 \ \ 121 \ \ 127 \ \ 119 \ \ 120 \ \ 115 \ \ 117 \ \ 124 \ \ 135 \ \ 128 \ \} \end{array}$$

**Example 4.4.4 Multiple Stimuli; Additive Gaussian Noise; 1d Inputs**

Now, let's add white Gaussian noise to the above data. This is easily accomplished with the -sigma $s$ command, which adds $N(0, s^2)$ random variates to the output.

Program 3dConvolve Command Line for Example 4.4.4

```
3dConvolve  \
-input1D   -nfirst 0   -nlast 19   -polort 1  \
-base_file Base.1D  \
-num_stimts 3 \
-stim_file 1 Random.1D   -stim_maxlag 1 2  \
-stim_file 2 Markov.1D   -stim_maxlag 2 2  \
-stim_file 3 English.1D  -stim_maxlag 3 2  \
-iresp 1 'IRF.1D[0]'  \
-iresp 2 'IRF.1D[1]'  \
-iresp 3 'IRF.1D[2]'  \
-sigma 1.0   -seed 123456789
```

■

The screen output should look like this:

$wn(t) =$

{ 102.40 101.02 110.65 116.07 118.70 108.58 109.34 116.35 119.92 118.59
115.74 120.87 126.85 118.91 122.16 114.98 118.23 123.67 136.45 128.22 }

Repeat the above command line, but use different values for the initial random number seed, by changing the argument of the -seed command. How does this effect the output?

**Example 4.4.5 (Continuation of Examples 1.4.3.2 and 2.3.2)**

In Example 1.4.3.2, we modeled the measured data, for a particular voxel, using the input "ideal" function ($Cos$) and the estimated motion parameters ($Roll$, $Pitch$, $Yaw$, $dS$, $dL$, $dP$) thus:

$$\begin{aligned} z(t) &= b_0 + b_1 \cdot t + c_0 \cdot Cos(t) + c_1 \cdot Cos(t-1) \\ &\quad + d \cdot Roll(t) + e \cdot Pitch(t) + f \cdot Yaw(t) + g \cdot dS(t) + h \cdot dL(t) + i \cdot dP(t) \end{aligned}$$

From Example 2.3.2, we have the following parameter estimates:

$$\mathbf{b} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} 820.7942 \\ 0.3943 \end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 0.0054 \\ 0.0040 \end{bmatrix},$$

$$\begin{aligned} \mathbf{m}' &= \begin{bmatrix} d & e & f & g & h & i \end{bmatrix} \\ &= \begin{bmatrix} 132.4883 & -18.0019 & -87.6356 & 24.2552 & 18.3589 & -29.6341 \end{bmatrix} \end{aligned}$$

Save **b** as a single column into file b.coef.1D; save **c** as a single column into file c.coef.1D, and save $\mathbf{m}'$ as a single row into file m.coef.1D. As before, we will assume that the "ideal" time series is contained in file cos7.00.1D, and that the 6 estimated motion parameter time series are contained in file v2.motion.1D. Now, execute the following command line:

Program 3dConvolve Command Line for Example 4.4.5

```
3dConvolve  \
-input1D   -nfirst 6   -nlast 67   -polort 1  \
-base_file b.coef.1D  \
-num_stimts 7 \
-stim_file 1 cos7.00.1D   -stim_maxlag 1 1  \
-stim_file 2 'v2.motion.1D[0]' \
-stim_file 3 'v2.motion.1D[1]' \
-stim_file 4 'v2.motion.1D[2]' \
-stim_file 5 'v2.motion.1D[3]' \
-stim_file 6 'v2.motion.1D[4]' \
-stim_file 7 'v2.motion.1D[5]' \
-iresp 1 c.coef.1D  \
-iresp 2 'm.coef.1D[0]'  \
-iresp 3 'm.coef.1D[1]'  \
-iresp 4 'm.coef.1D[2]'  \
-iresp 5 'm.coef.1D[3]'  \
-iresp 6 'm.coef.1D[4]'  \
-iresp 7 'm.coef.1D[5]'  \
-xout   -output myPred
```

■

The predicted measurement time series has been written to file myPred.1D. This time series can be plotted using the command

```
1dplot  -ignore 6  myPred.1D
```

However, if you wish to overlay the predicted time series on top of the actual data, which is contained in file 029_029_008.1D, this can be accomplished with the following script:

Script for Plotting Predicted Data on top of Actual Data

```
1dcat  029_029_008.1D  myPred.1D  >  Overlay.1D
1dplot  -ignore 6  -one  Overlay.1D
```

■

**Example 4.4.6 Single Stimulus; No Noise; 3d+time Input**

As a further illustration of the relationship between programs 3dDeconvolve and 3dConvolve, we present the following example. Here, we analyze the hypothetical 3d+time dataset Paula+orig, which we considered in Example 1.4.2.5. Again, please try this example using your own 3d+time dataset.

Recall that in Example 1.4.2.5, three output datasets were generated by program 3dDeconvolve. The first, Paula.bucket+orig, is a bucket dataset containing the estimates for the regression parameters and $t$-statistics, along with the $F$-statistics and $R^2$ values for the statistical significance of the fit, at each voxel. File Paula.irf+orig is a 3d+time dataset containing the estimated impulse response function for each voxel. Finally, file Paula.fit+orig is a 3d+time dataset containing the full model least squares fit to the observed data for each voxel.

Use the following command line to "predict" the measurement at each voxel location.

Program 3dConvolve Command Line for Example 4.4.6

```
3dConvolve  \
-input Paula+orig   -polort 1  \
-base_file 'Paula.bucket+orig[0,2]'  \
-num_stimts 1 \
-stim_file 1 Visual.1D   -stim_maxlag 1 10  \
-iresp 1 Paula.irf+orig  \
-xout   -output Paula.pred1
```

■

The -base_file command indicates that the baseline parameters $b_0$ and $b_1$, for each voxel, are to be read from the #0 and #2 sub-bricks, respectively, of the bucket dataset Paula.bucket+orig. The -num_stimts command indicates that there is one input stimulus function (and one IRF for each voxel). The -iresp command indicates that the IRF for each voxel is to be read from the 3d+time dataset Paula.irf+orig. The predicted measurement data will then be stored in the 3d+time dataset Paula.pred1+orig.

The above script reads the 11-point IRF for each voxel from the 3d+time dataset Paula.irf+orig, which was created by program 3dDeconvolve using the -iresp command. However, an entirely equivalent procedure is to read the IRF coefficients directly from the output bucket dataset Paula.buck+orig, as illustrated below:

Program 3dConvolve Command Line for Example 4.4.6

```
3dConvolve  \
-input Paula+orig   -polort 1  \
-base_file 'Paula.bucket+orig[0,2]'  \
-num_stimts 1 \
-stim_file 1 Visual.1D   -stim_maxlag 1 10  \
-iresp 1 'Paula.bucket+orig[4..24(2)]'  \
-xout   -output Paula.pred2
```

■

The command -iresp 1 'Paula.bucket+orig[4..24(2)]' indicates that the 11-point IRF coefficients are to be read from sub-bricks #4, #6, #8, #10, ..., #22, and #24 of dataset Paula.bucket+orig. This demonstrates that it is not necessary to save the IRF coefficients as a separate 3d+time dataset, if it is not convenient to do so. Obviously, it is important that the user correctly specify which sub-bricks of the bucket dataset contain the IRF coefficients. When in doubt, use program 3dinfo to print the sub-brick labels of the bucket dataset.

Verify that the 3d+time datasets Paula.pred1+orig and Paula.pred2+orig agree with the 3dDeconvolve fitted time series output Paula.fit+orig.

**Example 4.4.7 Multiple Stimuli; External Noise; 3d+time Input**

Here we consider the inverse to the problem considered in Example 1.4.3.6. Given the 3 input stimulus functions Random.1D, Markov.1D, English.1D, and the corresponding 3 IRF's for each voxel, and given the residual error time series for each voxel, the objective is to reconstruct the measured data contained in file Monica+orig.

Program 3dConvolve Command Line for Example 4.4.7

```
3dConvolve  \
-input Monica+orig   -polort 1  \
-base_file 'Monica.bucket+orig[0,1]'  \
-num_stimts 3 \
-stim_file 1 Random.1D  -stim_file 2 Markov.1D  -stim_file 3 English.1D \
-stim_minlag 1 2        -stim_minlag 2 2        -stim_minlag 3 2 \
-stim_maxlag 1 5        -stim_maxlag 2 5        -stim_maxlag 3 5 \
-iresp 1 'Monica.bucket+orig[2..5]'  \
-iresp 2 'Monica.bucket+orig[7..10]'  \
-iresp 3 'Monica.bucket+orig[12..15]'  \
-errts Monica.err+orig  \
-xout   -output Monica.recon
```

■

Note that since there are 3 input stimulus functions, there must be exactly 3 -iresp commands to read the corresponding IRF's. The -errts command indicates that the measurement error at each time point for each voxel is to be read from the 3d+time dataset Monica.err+orig.

Verify that the 3d+time dataset Monica.recon+orig agrees with the 3dDeconvolve input dataset Monica+orig.

# 5 References

[1 ] F. Stremler, *Fourier Methods of Signal Analysis*, Lecture Notes. (1974).

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

[3 ] W. H. Beyer (ed.), *Basic Statistical Tables.* Cleveland, OH: The Chemical Rubber Co. (1971).

[4 ] G. Casella, R. L. Berger, *Statistical Inference.* Pacific Grove, CA: Wadsworth & Brooks/Cole (1990).