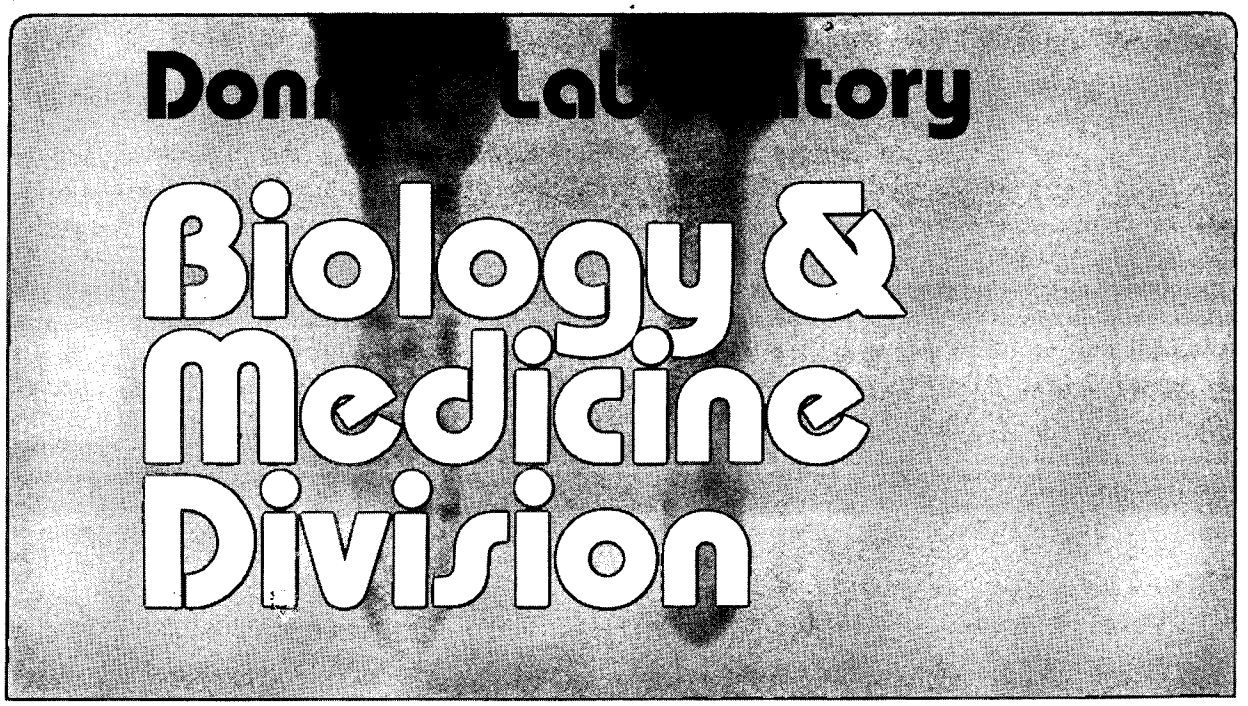
Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

KINETIC ANALYSIS OF DYNAMIC PET DATA

B. Knittel
(M.S. Thesis)

December 1983



Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

DISCLAIMER

Kinetic Analysis of Dynamic PET Data

Brian Knittel
Master's Report
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley

230 Donner Laboratory
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Table of Contents

Kinetic Analysis of Dynamic PET Data

Brian Knittel
Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley
and Lawrence Berkeley Laboratory

1. Introduction

Our goal is to quantify regional physiological processes such as blood flow and metabolism by means of tracer kinetic modeling and positron emission tomography (PET). Compartmental models are one way of characterizing the behavior of tracers in physiological systems. This paper describes a general method of estimating compartmental model rate constants from measurements of the concentration of tracers in blood and tissue, taken at multiple time intervals. A computer program which applies the method is described, and examples are shown for simulated and actual data acquired from the Donner 280-Crystal Positron Tomograph.

2. Background

2.1. Compartmental Models

Compartmental models can account for the exchange of tracer between physiological spaces such as blood and tissue, and between chemical states such a metabolic substrate and its products.

For example, Figure 1 shows a model which might be used to represent the exchange of a tracer between capillary blood and cells in an organ, where $b(t)$ is the concentration of tracer in the blood and $q(t)$ the concentration of tracer in the cells. The rate constants of exchange k_1 and k_2 are the parameters which describe the behavior of the tracer in the differential equation

$$\frac{dq}{dt}(t) = k_1 b(t) - k_2 q(t) . \tag{2.1}$$

The figure shows the concentration of tracer measured in the blood and in tissue after an ideal rapid bolus injection. The blood concentration $b(t)$ is called the input function. The cell concentration function resulting from an impulse injection of a unit amount of tracer is called the impulse response, and is denoted $h(t)$. For the model in Figure 1,

$$h(t) = k_1 e^{-k_2 t} . \tag{2.2}$$

The tissue measurement $w(t)$ is called the residue function, and includes contributions from both the cell concentration $q(t)$, and blood in the vasculature.

The shape of the input function in actual experiments depends on the duration of the injection, the blood flow to the organ, and the behavior of the tracer in the rest of the body. The response of the model to an arbitrary input function is given by the convolution integral

$$q(t) = \int_{-\infty}^{+\infty} h(\tau)\, b(t-\tau)\, d\tau = \int_{-\infty}^{+\infty} b(\tau)\, h(t-\tau)\, d\tau \tag{2.3}$$

and is denoted $q(t) = b \otimes h(t)$. $h(t) = 0$ for $t < 0$ and we assume that $b(t) = 0$ for $t < 0$, so

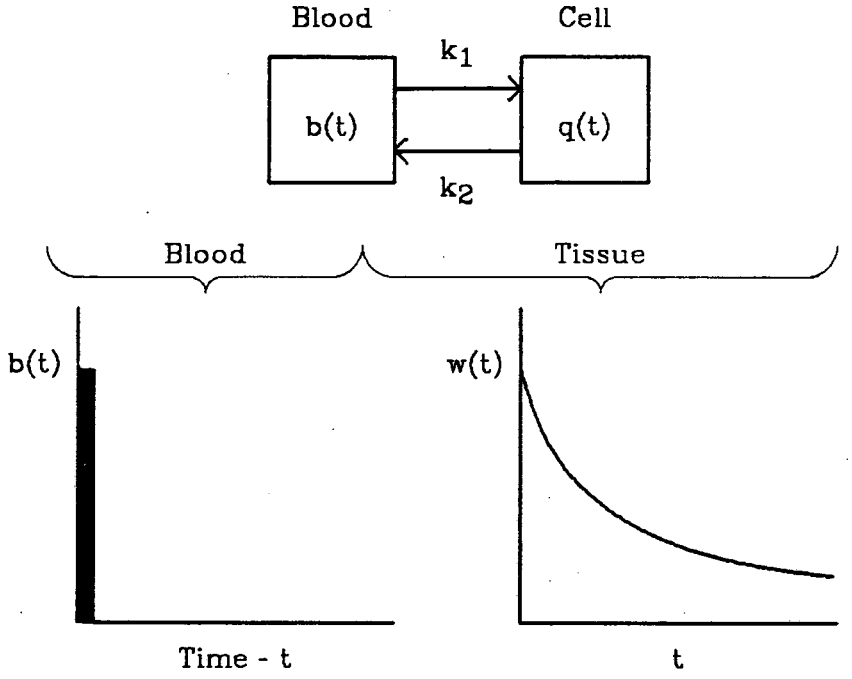


Figure 1. Two Compartment Model and Impulse Response.

$$q(t) = \int_0^t h(\tau)\, b(t-\tau)\, d\tau \quad . \tag{2.4}$$

The tissue tracer concentration $w(t)$ must account for the tissue volume occupied by blood in the vasculature. If the fractional volume of blood in a region of tissue is f_v, then the residue function is

$$\begin{aligned} w(t) &= f_v\, b(t) + (1-f_v)\, q(t) \\ &= f_v\, b(t) + (1-f_v)\, b \otimes h(t) \quad . \end{aligned} \tag{2.5}$$

This is the basic equation for all the models discussed here.

In tomographic experiments, the two measurable quantities are $b(t)$ and $w(t)$; it is not possible to independently measure the compartments contributing to w. The impulse response required for the computation of $w(t)$ is that of the combination of all nonvascular compartments. The determination of $h(t)$ for arbitrary compartmental models is discussed in section 4.2.

The blood activity is sometimes measured at a location somewhat distant to the site which we are modeling. We assume that the input function is shifted in time only. That is, we assume that all blood leaving the heart at a given moment has the same concentration of tracer, but the time it takes to reach different parts of the body varies. To determine the residue function from such a shifted input function we need only shift the time of evaluation of the model function by the same amount.

2.2. Parameter Estimation

After injecting a tracer, we collect measurements of the input function and residue function which include errors due to the statistics of radioactive decay and artifacts due to PET reconstruction. After selecting a physiologically appropriate compartmental model, we wish to determine what values of the model's parameters gave rise to the measurements. We will use the following symbols for the quantities under discussion:

Notation:

R	italic letter: scalar function or variable
$\mathbf{E}$	bold letter: column vector function or variable
E_i	i'th element of vector $\mathbf{E}$
B	capital roman letter: matrix
B_{ij} or $[B]_{ij}$	element in i'th row, j'th column of matrix B.
$\mathbf{E}^T$	row vector: transpose of $\mathbf{E}$
B^T	transpose of matrix B

Measurements:

N_b	number of blood measurements
N_w	number of tissue measurements
B_i	i'th blood measurement, $i = 1, 2, \ldots, N_b$
W_i	i'th tissue measurement, $i = 1, 2, \ldots, N_w$
T_{B_i}	time of i'th blood measurement
T_{W_i}	time of i'th tissue measurement
σ_{W_i}	uncertainty in i'th tissue measurement

Parameters:

$\boldsymbol{\beta}$	an arbitrary set of k model parameters (a vector of dimension k). These could be rate constants, vascular partial volumes, time shifts, etc.
$\boldsymbol{\beta}^*$	the true values of $\boldsymbol{\beta}$ in the region of interest.
$\hat{\boldsymbol{\beta}}$	our estimate of $\boldsymbol{\beta}^*$
σ_{β_j}	uncertainty in determined $\hat{\beta}_j$

Model Functions:

$b(t)$	input function
$w(\boldsymbol{\beta},t)$	residue function
$w_i(\boldsymbol{\beta})$	residue function evaluated at T_{W_i}

We assume that there are only random errors in the measurements W_j. That is, $W_j = w_j(\boldsymbol{\beta}^*) + \varepsilon_j$, where ε_j are random with zero mean. Since there are errors in the measurements, we cannot determine the exact rate constants $\boldsymbol{\beta}^*$ which generated the observed residue function. We must find some method of estimating $\boldsymbol{\beta}^*$ from imprecise measurements. Estimation theory is a branch of mathematical statistics which deals with problems of this nature. For an introduction to estimation theory with an emphasis on the type of modeling problem discussed here, the reader is referred to a text such as Bard [1].

We hope to find a criterion for selecting a $\hat{\boldsymbol{\beta}}$ which is as close to $\boldsymbol{\beta}^*$ as possible. Since the measurements have random variations and would vary in repetitions of the same experiment, we expect that our parameter estimates would vary as well. Two desirable constraints on this variability are that

(1) the average of $\hat{\boldsymbol{\beta}}$ in repetitions of the experiment should be $\boldsymbol{\beta}^*$, that is, the expectations $E(\hat{\beta}_i) = \beta_i^*$, and

(2) the average squared errors (variances) in the estimated parameters are as small as possible, that is, $E(\hat{\beta}_i - \beta_i^*)^2 = \sigma_{\beta_i}^2$ are the smallest obtainable for any estimator of β_i^*, for every possible value of β_i^*.

This is called the "uniform minimum variance unbiased" (UMVU) criterion. Proof that an estimation method is UMVU can only be obtained in certain cases. For example, when the function w is linear in $\boldsymbol{\beta}$ (that is,

$$w(\boldsymbol{\beta}, t) = \sum_{i=1}^{k} \beta_i g_i(t) \tag{2.6}$$

where g_i are arbitrary functions of t), and the errors ε_j are independent and distributed normally with mean 0 and variance $\sigma_{W_j}^2$, then the parameters $\hat{\boldsymbol{\beta}}$ minimizing

$$R(\boldsymbol{\beta}) = \sum_{j=1}^{N_w} \frac{(W_j - w_j(\boldsymbol{\beta}))^2}{\sigma^2_{W_j}} \tag{2.7}$$

are a UMVU estimate of $\boldsymbol{\beta}^*$ [2].

However, in PET kinetic modeling, the errors are not exactly normally distributed, and the function w is not linear in its parameters. Still, the least squares estimate is probably the best available estimate. If the model function is "locally" linear, that is, w varies more or less linearly when its parameters are varied small amounts (as we often consider the surface of the Earth to be locally flat), and the measurement errors are approximately normally distributed, then the $\hat{\boldsymbol{\beta}}$ minimizing R in Equation 2.7 at least approximately meets the UMVU criterion.

We now need an algorithm to find the values of the rate constants which minimize R. The algorithm is to produce this $\hat{\boldsymbol{\beta}}$ and is also to estimate $\boldsymbol{\sigma}_{\beta}$.

2.3. Parameter Covariance

The variance $\sigma^2_{\beta_i}$ of an estimate β_i is the expected squared deviation of the estimate from its true value β_i^*. This is the variance we would expect to measure in $\hat{\beta}_i$ if the experiment were repeated many times. If our assumptions about the normal distribution of the errors in the residue function are true, there is a ≈66 percent (1 standard deviation) chance that $\hat{\beta}_i$ is in the range $\beta_i^* - \sigma_{\beta_i} \le \hat{\beta}_i \le \beta_i^* + \sigma_{\beta_i}$.

σ_{β_i} is estimated as the amount β_i must be varied from $\hat{\beta}_i$ to increase $R(\boldsymbol{\beta})$ from the minimum by one (Fig. 2). The more β_i can be varied without much changing R, the greater its variance. When there is more than one parameter, we must consider the amount that β_i can be varied to increase R by one when the other parameters are allowed to vary as well. Figure 3 is a contour map of R as a function of β_1 and β_2. We see that σ_{β_1} and σ_{β_2} are the distances which increase R from the minimum $R_{\min} = R(\hat{\boldsymbol{\beta}})$ by one, under the most unfavorable circumstances (β_1 and β_2 increased together). When changing parameters together produces little net change in R, we say that the parameters are highly correlated. The covariance matrix C describes this relationship between parameters. Its elements are defined by

$$C_{ij} = \mathrm{E}(\hat{\beta}_i - \beta_i^*)(\hat{\beta}_j - \beta_j^*) \ . \tag{2.8}$$

Note that the diagonal elements $C_{ii} = \mathrm{E}(\hat{\beta}_i - \beta_i^*)^2$ are the variances $\sigma_{\beta_i}{}^2$. The *correlation coefficient* defined by

$$r_{ij} = \frac{C_{ij}}{\sqrt{\sigma^2_{\beta_i}\,\sigma^2_{\beta_j}}} \tag{2.9}$$

conveniently describes the covariance between parameters β_i and β_j. It takes values between -1 and +1. r is zero for uncorrelated parameters and increases in magnitude toward 1 as correlation becomes greater.

The covariance matrix or the correlation matrix is important in the consideration of the significance of fit parameters, for they describe the reliability of the determination of the rate constants. One needs covariances to accurately estimate the uncertainty of functions (e.g. sums or ratios) of the determined parameters.

When comparing competing models for the explanation of tracer kinetics, one can consider the sum of squared errors $R(\boldsymbol{\beta})$; the model which produces the lowest R is considered better. R can always be lowered by increasing the number of compartments; with more parameters to adjust, the data can be better fit. However, as the number of parameters is increased, so are the parameter variances. When the number of parameters is increased beyond that required to adequately represent the data, the correlation between the fit parameters increases dramatically. The covariance matrix thus indicates the information content of the parameters.

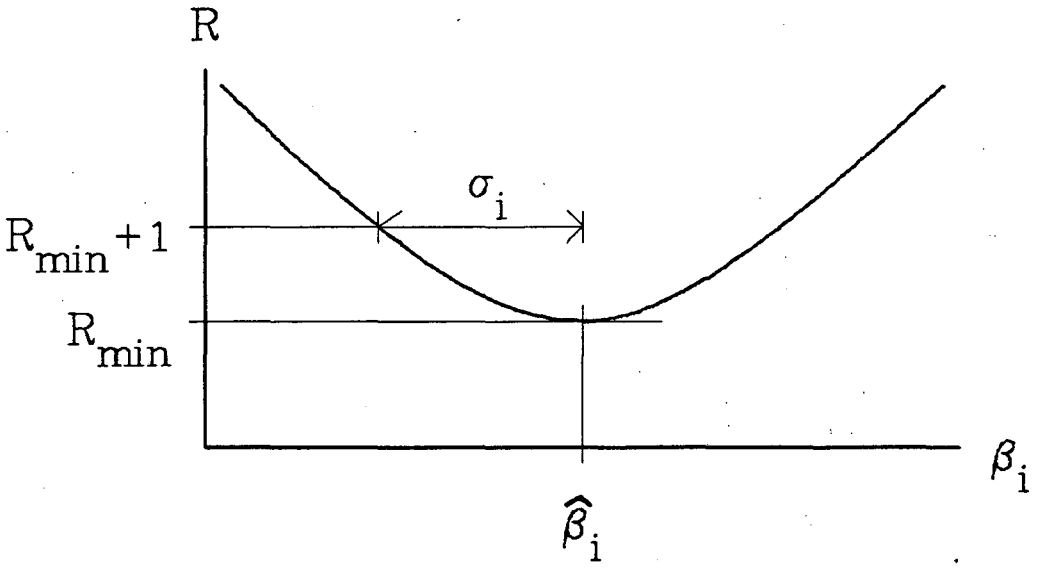


Figure 2. Parameter Variance.

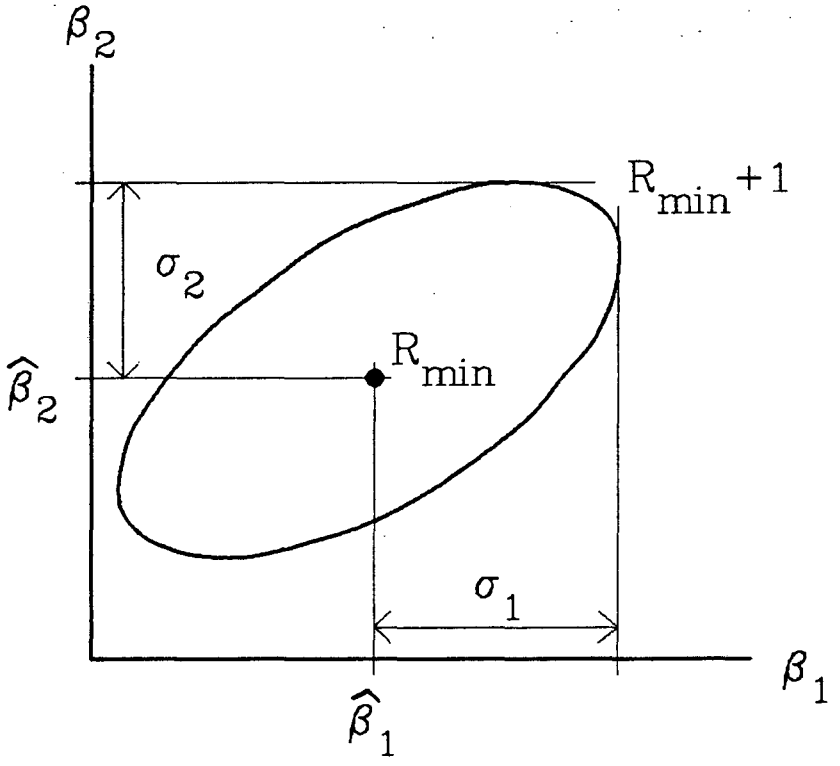


Figure 3. Parameter Variance with Correlation.

3. The Minimization Method

The minimum of $R(\boldsymbol{\beta})$ is found at the zero gradient point

$$\frac{\partial R(\hat{\boldsymbol{\beta}})}{\partial \beta_i} = 0, \quad i = 1, \ldots, k$$

or in vector form

$$\nabla R(\hat{\boldsymbol{\beta}}) = \mathbf{0} . \tag{3.1}$$

The gradient of R as defined in (2.7) is

$$\frac{\partial R}{\partial \beta_i} = -2 \sum_{j=1}^{N_w} \frac{W_j - w_j(\boldsymbol{\beta})}{\sigma_j^2} \frac{\partial w}{\partial \beta_i} \tag{3.2}$$

or in matrix notation

$$R(\boldsymbol{\beta}) = [\mathrm{W} - \mathrm{w}(\boldsymbol{\beta})]^{\mathrm{T}} \Psi^{-1} [\mathrm{W} - \mathrm{w}(\boldsymbol{\beta})]$$

$$\nabla R(\boldsymbol{\beta}) = -2\, \mathrm{T}(\boldsymbol{\beta})^{\mathrm{T}} \Psi^{-1} [\mathrm{W} - \mathrm{w}(\boldsymbol{\beta})] \tag{3.3}$$

where $\mathbf{W}$ is the observation vector, Ψ is the covariance matrix for the observations, $\mathbf{w}(\boldsymbol{\beta})$ is the vector of model values generated from the parameters $\boldsymbol{\beta}$, and $\mathrm{T}(\boldsymbol{\beta})$ is the gradient of $\mathbf{w}$:

$$\mathbf{w}(\boldsymbol{\beta}) = \begin{bmatrix} w_1(\boldsymbol{\beta}) \\ w_2(\boldsymbol{\beta}) \\ \vdots \\ w_{N_w}(\boldsymbol{\beta}) \end{bmatrix},$$

$$\mathbf{W} = \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_{N_w} \end{bmatrix},$$

$$\Psi = \begin{bmatrix} \sigma^2_{W_1} & \mathrm{Cov}(W_1, W_2) & \cdots & \mathrm{Cov}(W_1, W_{N_w}) \\ \mathrm{Cov}(W_1, W_2) & \sigma^2_{W_2} & \cdots & \mathrm{Cov}(W_2, W_{N_w}) \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{Cov}(W_1, W_{N_w}) & \mathrm{Cov}(W_2, W_{N_w}) & \cdots & \sigma^2_{W_{N_w}} \end{bmatrix},$$

and

$$\mathrm{T}(\boldsymbol{\beta}) = \begin{bmatrix} \frac{\partial w_1}{\partial \beta_1} & \frac{\partial w_1}{\partial \beta_2} & \cdots & \frac{\partial w_1}{\partial \beta_k} \\ \frac{\partial w_2}{\partial \beta_1} & \frac{\partial w_2}{\partial \beta_2} & \cdots & \frac{\partial w_2}{\partial \beta_k} \\ \vdots & \vdots & & \vdots \\ \frac{\partial w_{N_w}}{\partial \beta_1} & \frac{\partial w_{N_w}}{\partial \beta_2} & \cdots & \frac{\partial w_{N_w}}{\partial \beta_k} \end{bmatrix} = \nabla_{\boldsymbol{\beta}} \mathbf{w}(\boldsymbol{\beta}) .$$

When we assume that the measurements are uncorrelated, the Cov terms in Ψ are zero.

If $w(\boldsymbol{\beta},t)$ is a linear function of $\boldsymbol{\beta}$, R is quadratic and has a unique solution, easily obtained from (3.1). When $w(\boldsymbol{\beta},t)$ is not linear in $\boldsymbol{\beta}$, the solution to (3.1) may not be unique – there may be many extrema of R. Furthermore, it may be difficult to solve (3.1) for $\hat{\boldsymbol{\beta}}$. Nonlinear least squares methods usually proceed by choosing $\boldsymbol{\beta}_0$, an initial estimate of $\hat{\boldsymbol{\beta}}$, examining R at $\boldsymbol{\beta}_0$, and iteratively moving $\boldsymbol{\beta}_0$ toward the minimum.

3.1. Gauss-Newton Method

We can make a linear approximation to w with the first order Taylor expansion

$$w(\boldsymbol{\beta},t) \cong w(\boldsymbol{\beta}_0,t) + \sum_{j=1}^{k} (\beta_j - \beta_{0j}) \frac{\partial w(\boldsymbol{\beta}_0,t)}{\partial \beta_j} \tag{3.4}$$

or in matrix notation

$$\mathbf{w}(\boldsymbol{\beta}) \cong \mathbf{w}(\boldsymbol{\beta}_0) + \mathrm{T}(\boldsymbol{\beta}_0)\,(\boldsymbol{\beta} - \boldsymbol{\beta}_0) \ . \tag{3.5}$$

if $\boldsymbol{\beta}$ is close to $\boldsymbol{\beta}_0$. If we use this approximation in (3.3) and assume that $\mathrm{T}(\hat{\boldsymbol{\beta}}) \cong \mathrm{T}(\hat{\boldsymbol{\beta}}_0)$ then the minimum is found at

$$\mathrm{T}(\boldsymbol{\beta}_0)^{\mathrm{T}}\, \Psi^{-1} \left[\mathbf{W} - \mathbf{w}(\boldsymbol{\beta}_0) - \mathrm{T}(\boldsymbol{\beta}_0)^{\mathrm{T}}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}_0)\right] = \mathbf{0} \ . \tag{3.6}$$

Defining $\mathrm{A} = \mathrm{T}(\boldsymbol{\beta}_0)^{\mathrm{T}}\Psi^{-1}\mathrm{T}(\hat{\boldsymbol{\beta}})$ and $\mathbf{G} = \mathrm{T}(\boldsymbol{\beta}_0)^{\mathrm{T}}\Psi^{-1}(\mathbf{W} - \mathbf{w}(\boldsymbol{\beta}_0))$, we can solve (3.6) for $\hat{\boldsymbol{\beta}}$:

$$\mathbf{G} - \mathrm{A}(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}^0) = \mathbf{0}$$

$$\mathrm{A}\,(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}_0) = \mathbf{G}$$

$$\hat{\boldsymbol{\beta}} = \boldsymbol{\beta}_0 + \mathrm{A}^{-1}\mathbf{G} = \boldsymbol{\beta}_0 + \boldsymbol{\delta}_t \ . \tag{3.7}$$

where $\boldsymbol{\delta}_t = \mathrm{A}^{-1}\mathbf{G}$ is the Taylor series correction vector (iteration step).

When the measurements are uncorrelated, Ψ is diagonal, and the elements of the matrix A and vector $\mathbf{E}$ are

$$\begin{aligned} A_{mn} &= \sum_{j=1}^{N_w} \frac{\partial w_j}{\partial \beta_m} \frac{\partial w_j}{\partial \beta_n} \frac{1}{\sigma_j} \\ E_m &= \sum_{j=1}^{N_w} \frac{\partial w_j}{\partial \beta_m} \frac{W_j - w_j(\boldsymbol{\beta}_0)}{\sigma_j} \end{aligned} \ . \tag{3.8}$$

In the Gauss-Newton minimization method, one iteratively solves (3.7) for steps $\boldsymbol{\delta}_t$. The magnitude of $\boldsymbol{\delta}_t$ must be reduced if it takes $\boldsymbol{\beta}$ to a higher value of R than $R(\boldsymbol{\beta}_0)$.

3.2. Steepest Descent Method

The gradient of R at an estimate $\boldsymbol{\beta}_0$ is given by

$$\nabla R(\boldsymbol{\beta}) = -2\ \mathrm{T}(\boldsymbol{\beta}_0)^{\mathrm{T}}\, \Psi^{-1} \left[\mathbf{W} - \mathbf{w}(\boldsymbol{\beta}_0)\right] = -2\mathbf{G} \ . \tag{3.9}$$

Iterative steps $\boldsymbol{\delta}_g$ proportional to ∇R always lead to a lower value of R for a sufficiently small proportion of $\boldsymbol{\delta}_g$, but are slow to converge near the minimum. Near the minimum, where R is relatively flat, gradient steps tend to zig-zag across the true direction of the minimum (as streams will meander across a meadow). For this reason, strict steepest descent methods are seldom used in practice.

3.3. Marquardt Interpolation

The Marquardt Algorithm [3] interpolates between the Taylor and gradient steps $\boldsymbol{\delta}_t$ and $\boldsymbol{\delta}_g$ with a step computed by

$$\boldsymbol{\delta} = (\mathrm{A} + \mathrm{I}\lambda)^{-1}\, \mathbf{G} \tag{3.10}$$

As $\lambda \to 0$, $\boldsymbol{\delta} \to \mathrm{A}^{-1}\mathbf{G}$, the Taylor step, and as $\lambda \to \infty$, $\boldsymbol{\delta} \to \mathbf{G}/\lambda$, the steepest descent step. The algorithm attempts to use as small a value of λ as possible. If the step $\boldsymbol{\delta}$ would increase R, and $\boldsymbol{\delta}$ is not near the gradient $\boldsymbol{\delta}_g$, say more than 37° apart, the step is recomputed with a larger λ. λ is typically increased or decreased by a factor $v = 10$, changing λ

When $\boldsymbol{\delta}$ increases R but $\boldsymbol{\delta}$ and $\boldsymbol{\delta}_g$ are close together, changing λ won't help as much as reducing the magnitude of $\boldsymbol{\delta}$, so the step is divided by two until R is no longer increased.

The numerical aspects of the algorithm are improved if the matrix A has one on the diagonal. To achieve this, the computation of $\boldsymbol{\delta}$ is performed with

scaled variables A^* and $\mathbf{G}^*$. The true step δ is computed by reversing the scaling on $\boldsymbol{\delta}^*$.

3.4. Algorithm Outline

1. Initialize:
   $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_0$, $\lambda \leftarrow .1$.
2. Start an iteration. Try reducing λ, save initial conditions:
   $\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta}$, $R' \leftarrow R(\boldsymbol{\beta})$, $\lambda \leftarrow \lambda / \upsilon$.
3. compute A, E by Equation (3.8).
4. Scale A and E so that A has 1 on the diagonal:
   $A_{ij}^* \leftarrow A_{ij} / \sqrt{A_{ii}\, A_{jj}}$.
   $G_i^* \leftarrow G_i / \sqrt{A_{ii}}$.
5. Compute (still scaled) step:
   $\boldsymbol{\delta}^* \leftarrow (A^* + \lambda I)^{-1} \mathbf{G}^*$.
6. Unscale step:
   $\delta_i \leftarrow \delta_i^* / \sqrt{A_{ii}}$.
7. Compute (tentative) new value of $\boldsymbol{\beta}$:
   $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}' + \boldsymbol{\delta}$.
8. Bad Step? – if so, alter the step:
   if $R(\boldsymbol{\beta}) \geq R'$
   if angle between δ and δ_g < 37°, reduce step size:
   (i.e. if $\boldsymbol{\delta}^T \cdot \boldsymbol{\delta}_g / \sqrt{\boldsymbol{\delta}^T \cdot \boldsymbol{\delta}\ \ \boldsymbol{\delta}_g^T \cdot \boldsymbol{\delta}_g} \leq \cos(37°)$)
   repeat
   $\delta_i \leftarrow \delta_i / 2$
   $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}' + h\boldsymbol{\delta}$
   until $R(\boldsymbol{\beta}) \leq R'$, then go to 9
   otherwise, repeat step calculation with increased λ:
   $\lambda \leftarrow \upsilon\lambda$, then go to 5.
9. check convergence:
   if any $\dfrac{|\delta_i|}{|\beta_i| + \tau} \geq \varepsilon$, go to 2 for another iteration.
10. stop.

The convergence test parameters τ and ε are typically 10^{-6} and 10^{-4} respectively.

4. Program *Fit* — Numerical Methods

Computer program *fit* was written to implement the kinetic analysis method described above. The Marquardt Algorithm requires $w(t,\beta)$ and the derivatives $\partial w / \partial \beta_i$. This section describes the numerical methods used to compute the required functions.

4.1. Input Function Model

The computer program has four selectable input function models:

1. $$b(t) = \sum_{j=1}^{2} A_j \, e^{-M_j (t - t_I)} \tag{4.1}$$

2. $$b(t) = \sum_{j=1}^{2} A_j \, (t - t_I) \, e^{-M_j (t - t_I)} \tag{4.2}$$

3. $$b(t) = \sum_{j=1}^{2} A_j \, (t - t_I) \, e^{-M_j (t - t_I)^2} \tag{4.3}$$

where A_j units of blood activity
M_j rate constants, min^{-1}
t_I input function starting time in seconds
The times and time shifts are in seconds and the rate constants in min^{-1}; the program inserts the required 1/60 conversion factors.

4. $b(t)$ = linear interpolation or extrapolation of input measurements:

$$= \begin{cases} 0, & t < 0 \\ B_1 + \dfrac{t - T_{B_1}}{T_{B_2} - T_{B_1}} (B_2 - B_1), & 0 \le t \le t_1 \\ B_{j-1} + \dfrac{t - T_{B_{j-1}}}{T_{B_j} - T_{B_{j-1}}} (B_j - B_{j-1}), & t_{j-1} < t \le t_j \\ B_{N-1} + \dfrac{t - T_{B_{N-1}}}{T_{B_N} - T_{B_{N-1}}} (B_N - B_{N-1}), & t > t_N \end{cases} \tag{4.4}$$

(clipped to 0 if the interpolated or extrapolated value is negative).

Model 4 requires no fitting to the input measurements and does not make any assumptions about its form. However, as the input measurements are time averages of the input function over the image collection intervals, some information is necessarily lost. In Figure 4, we show a fast-rising input function, the averaged samples, and the resulting Model 4 function. The shaded areas show the error in the approximation. The areas of over- and underestimation should approximately cancel each other. If the input function is not fast-rising, these errors are small.

Also, Model 4 introduces statistical errors into the *model* function, due to the $f_v \, b(t)$ term in Equation (2.5), which we do not currently account for in the fitting process. The uncertainty of the input measurements should be incorporated into the weighting of the squared-error function $R(\boldsymbol{\beta})$.

4.2. Impulse Response Computation

The program fits rate constants for the three-compartment model below (Fig. 5), which we apply to several physiological systems. Compartments q_1 and q_2 represent tracer in two spaces or chemical states in tissue. The differential equations for this system are

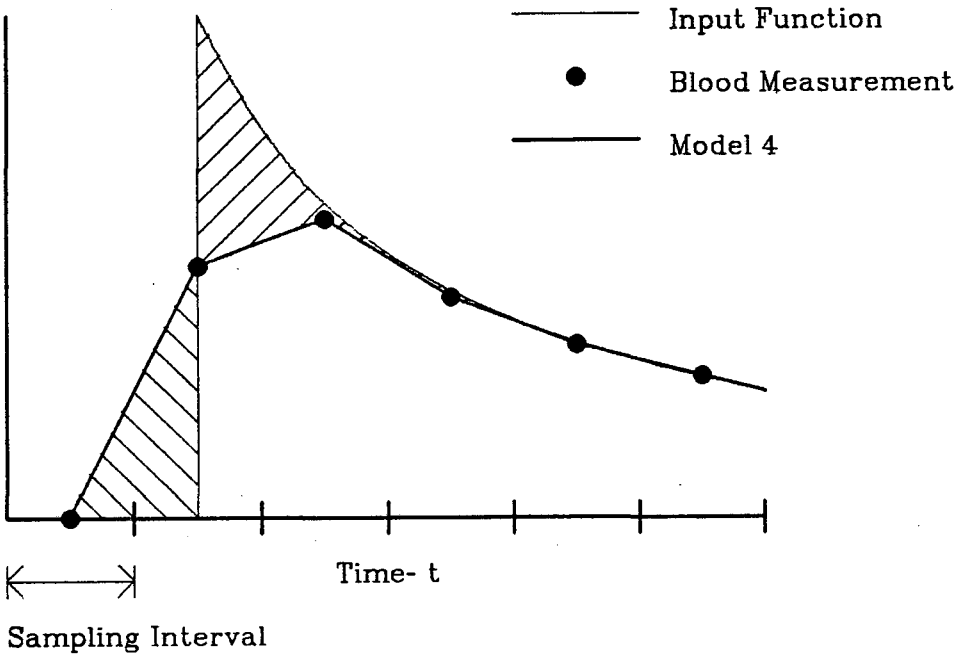


Figure 4. Actual Input Function and Model 4.

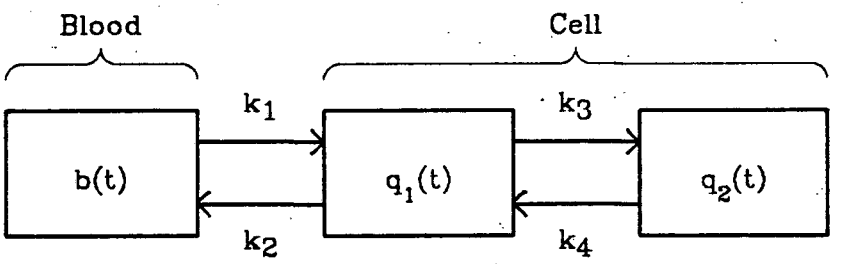


Figure 5. Three Compartment Model

$$\frac{dq_1}{dt}(t) = k_1 b(t) - (k_2 + k_3) q_1(t) + k_4 q_2(t)$$
$$\frac{dq_2}{dt}(t) = k_3 q_1(t) - k_4 q_2(t) \qquad (4.5)$$

The impulse response may be derived by solving the differential equations for a delta function input $b(t) = \delta(t)$, or by solving for $b(t) = 0$ with initial conditions $Q_1(0) = k_1$ and $Q_2(0) = 0$. Impulse response is more easily calculated by signal-flow graph analysis, which yields the impulse response of an arbitrary network of compartments with minimal effort. See Mason & Zimmermann[4] for a discussion of the method.

The impulse response $h(t)$ of the sum of compartments q_1 and q_2 is

$$h(t) = f_1 e^{-\alpha_1 t} + f_2 e^{-\alpha_2 t} , \qquad (4.6)$$

where

$$\alpha_1 = (\beta_1 - \beta_2)/2 \qquad \alpha_2 = (\beta_1 + \beta_2)/2$$

$$\beta_1 = k_2 + k_3 + k_4 \qquad \beta_2 = \sqrt{\beta_1^2 - 4k_2k_4}$$

$$f_1 = \frac{k_1(k_3 + k_4 - \alpha_1)}{\beta_2} \qquad f_2 = \frac{k_1(\alpha_2 - k_3 - k_4)}{\beta_2} .$$

When $k_3 = 0$, the model is effectively reduced to two compartments, and the impulse response is correctly evaluated using Equation (4.6); the function reduces to

$$h(t) = k_1 e^{-k_2 t} . \tag{4.7}$$

4.3. Convolution Method

The cell concentration $q(t)$ is computed by the convolution integral in Equation (2.3). Rather than explicitly solve the convolution integral for all input function models, we use an approximate method. To compute the convolution integral, we evaluate the impulse response and input function at the tissue and blood measurement times T_{W_j} and T_{B_j} respectively. Function **con** computes the convolution of the two linearly interpolated functions $b'(t)$ and $h'(t)$ described by these points. The error introduced by this piecewise-linearization is less than one percent with the exponential and near-exponential functions encountered in our studies; see section 6.3 for a discussion.

The integral of $h'(\tau)b'(t-\tau)$ over the whole interval $0 \le \tau \le t$ is the sum of the integrals over intervals bounded by the set of times $\{0, T_{W_1}, T_{W_2}, \ldots, T_{N_w}\} \cup \{t, t-T_{W_1}, t-T_{W_2}, \ldots, t-T_{N_w}\}$ (Fig. 6). The integrand over one of these intervals, say $r \le \tau \le s$, is the product of the line segments $(r, b_r) \to (s, b_s)$ and $(r, h_r) \to (s, h_s)$, where $h_r = h(r)$, $h_s = h(s)$, $b_r = b(t-r)$, and $b_s = b(t-s)$. The integral is

$$\int_r^s \left(h_r + \frac{\tau - r}{s - r}(h_s - h_r)\right)\left(b_r + \frac{\tau - r}{s - r}(b_s - b_r)\right) d\tau . \tag{4.8}$$

Let $\Delta\tau = s - r$, $\Delta h = h_s - h_r$, $\Delta b = b_s - b_r$, and change the variable of integration to $\eta = \tau - r$:

$$= \int_0^{\Delta\tau} \left(h_r + \frac{\eta}{\Delta\tau}\Delta h\right)\left(b_r + \frac{\eta}{\Delta\tau}\Delta b\right) d\eta \tag{4.9}$$

$$= \Delta\tau \left(h_r b_r + \frac{h_r \Delta b}{2} + \frac{b_r \Delta h}{2} + \frac{\Delta h \Delta b}{3}\right) \tag{4.10}$$

$$= \frac{\Delta\tau}{6}(2h_r b_r + 2h_s b_s + h_r b_s + h_s b_r) . \tag{4.11}$$

so the complete convolution is

$$q(t) = \sum_{\substack{\text{intervals}(r,s) \\ \text{spanning } 0,t}} \frac{\Delta\tau}{6}(2h_r b_r + 2h_s b_s + h_r b_s + h_s b_r) . \tag{4.12}$$

The code in function **con** steps through the set of times T_{W_j} and $t - T_{B_k}$, looks for boundary points, and sums the interval integrals.

4.4. Residue Function

The residue function is computed as in Equation (2.5), as the sum of the vascular and cell components. There is an additional time shift parameter *t0* which accounts for a difference in the sampling time between the blood and tissue sites, as discussed in Section 2.1. The model function is

$$w(t, \boldsymbol{\beta}) = f_v b(t - t_0) + (1 - f_v) q(\boldsymbol{\beta}, t - t_0) . \tag{4.13}$$

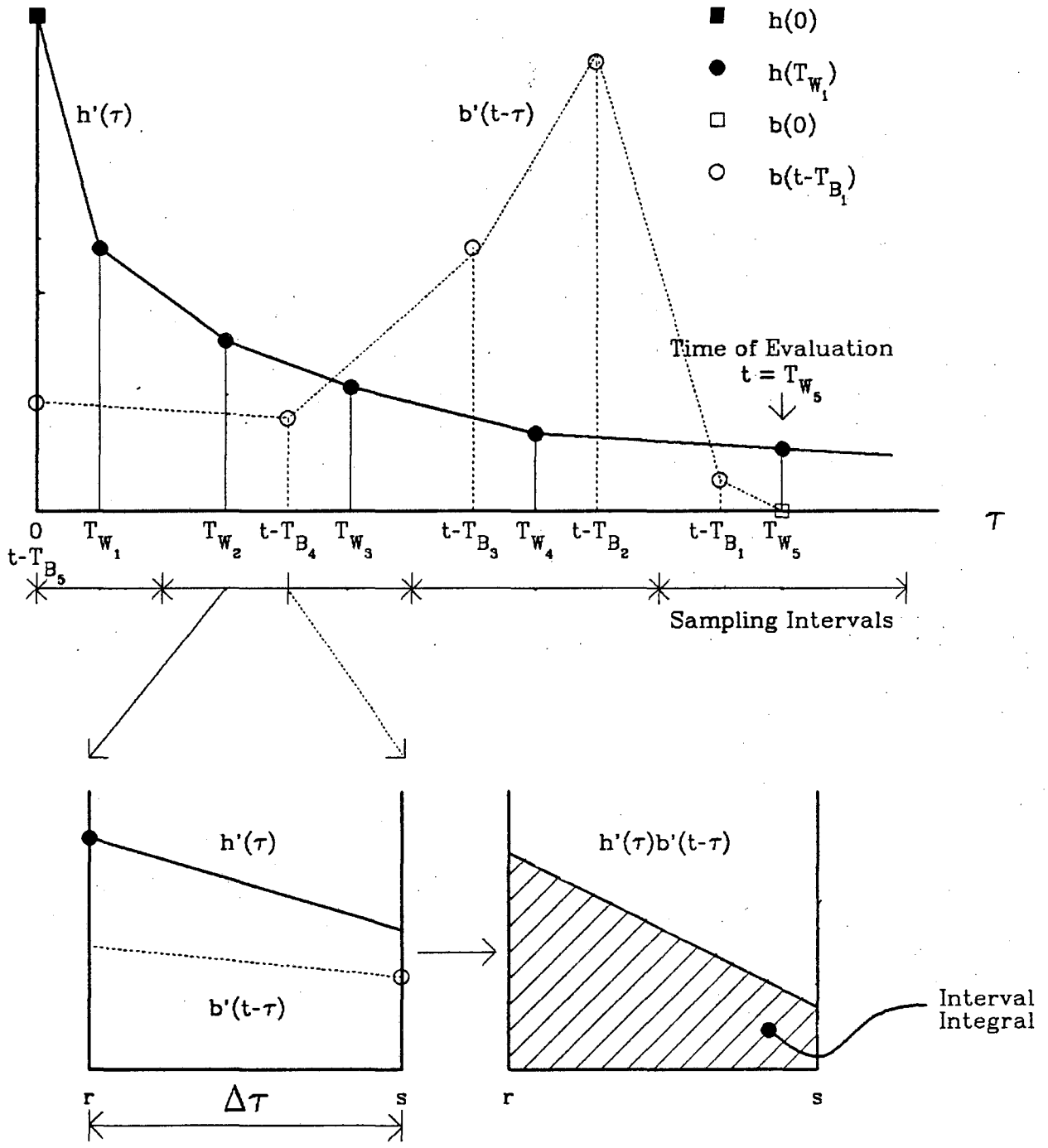


Figure 6. Convolution by Summation of Linear Intervals.

4.5. Partial Derivatives

We compute the model residue function by the convolution method described above, and its derivatives by the forward difference equation. For a given parameter β_i in the vector β,

$$\frac{\partial w_j(\beta_i)}{\partial \beta_i} \cong \frac{w_j(\beta_i+h) - w_j(\beta_i)}{h} \quad , h>0. \tag{4.13}$$

The error in this estimate is a function of h, given by

$$E(h) = \frac{1}{2}h\,|w''(\eta)| \quad , \quad \beta_i \le \eta \le \beta_i+h \tag{4.14}$$

for some η in the range $\beta_i \le \eta \le \beta_i+h$. Reducing h to zero would reduce the error to zero if it were not for the finite precision of digital computer floating point representations. Due to the round-off or discretization error in the calculation of $w(\beta_i+h)-w(\beta_i)$, which we will call Δ, the error in the derivative estimate is

$$E(h) = \frac{1}{2}h\,|w''(\eta)| + \frac{\Delta}{h} \tag{4.15}$$

The minimum E is found at

$$h_{opt} = \left[\frac{2\Delta}{|w''(\eta)|}\right]^{1/2} \tag{4.16}$$

For single precision (24 bit mantissa) on the PDP-11, $\Delta = 5\times10^{-7}$. We find $h=|.001\beta_i|$ to work well in our application. The error in the derivative estimate is around one percent with parameters in the range we encounter.

While other numerical derivative formulae offer lower errors, the forward difference requires only one additional w evaluation for each required derivative. This is a considerable saving in this application, for we must compute, store, and convolve a complete set of impulse function samples for each derivative.

4.6. Covariance Matrix

The covariance matrix C of a set of linear parameters $\boldsymbol{\beta}$ is the inverse of the derivative matrix B (defined in Equation 3.9) evaluated at $\boldsymbol{\beta}^*$. The Marquardt subroutine estimates C by inverting B evaluated at $\hat{\boldsymbol{\beta}}$ under the assumptions that $w(\boldsymbol{\beta})$ is linear in the neighborhood of $\hat{\boldsymbol{\beta}}$, that $\boldsymbol{\beta}$ is close to $\boldsymbol{\beta}^*$, and that B evaluated at our estimate $\hat{\boldsymbol{\beta}}$ is a good approximation to B evaluated at $\boldsymbol{\beta}^*$.

After each fit we print the parameter uncertainties (square root of their variance, from the covariance matrix), and the correlation matrix:

$$\begin{aligned} \sigma_{\beta_i} &\cong \sqrt{[B^{-1}]_{ii}} \\ r_{ij} &\cong \frac{[B^{-1}]_{ij}}{\sigma_{\beta_i}\,\sigma_{\beta_j}} \end{aligned} \tag{4.17}$$

5. Implementation

5.1. Software Tools

Program *fit* (listing in Appendix A, documentation in Appendix D) is written largely in Ratfor for operation under the Software Tools (ST) Virtual Operating System. Software Tools is portable program development environment which is modeled after UNIX*, and whose design and philosophy are expounded in *Software Tools* by Brian W. Kernighan and P.J. Plauger [5]. ST provides the same programming and command languages, user interface, documentation, utilities, and library subroutines for all operating systems and computers on which it is supported. We use the RSX-11M V4.0† implementation of the Software Tools Virtual Operating System, obtained from the Computer Science and Applied Mathematics group at Lawrence Berkeley Laboratory[6]. This ST system is currently running on a PDP-11/44 computer.

An invaluable feature of ST is the ability to conveniently specify at run time whether the program's input and output are to be connected to the user's terminal, to disk files, or directly to other programs. This enables the same program to be used interactively, as a "batch" type program, or as part of a metaprogram comprised of several tools. In the words of the authors,

> Whenever possible we will build more complicated programs up from the simpler; whenever possible we will ***avoid*** building at all, by finding new uses for existing tools, singly or in combination. Our programs ***work together***; their cumulative effect is much greater than you could get from a similar collection of programs that you couldn't easily connect [7].

For example, the simulation data presented in Section 6 were generated, fit, plotted, and summarized by applying both newly-built and existing tools, with almost no manual manipulation. The versatility of ST makes it useful in the development and testing of scientific data analysis programs.

5.2. Source of the Data

Positron emission tomography (PET) noninvasively measures radioactivity in tissue volumes as small as one cubic centimeter, without superposition of activity from other regions.

The Donner 280-crystal positron tomograph[8] is capable of taking cross-sectional images as frequently as every second, and can synchronize data collection with the beating of the heart. The spatial resolution of 8 mm full width at half-maximum (FWHM) is sufficient to quantify radioisotope concentration in regions of tissue of 2 cm. dimension.

Images are typically taken every 2.5 to 5.0 seconds for the first one or two minutes after a rapid intravenous injection of 5 - 10 seconds duration, and at longer intervals thereafter.

The input function is measured tomographically if the left ventricle or aorta is visible in the field of view, otherwise the input function is measured by sampling arterial or arterialized blood from a catheter.

After imaging, regions of interest (ROIs) are drawn over a high statistics image in which anatomical details are well-defined. A region of interest in the middle of the left ventricle of the heart or the aorta may supply the input function.

Sequential PET images are reconstructed[9] and the activity density in each region is computed after appropriate corrections for radioactive decay, attenuation, and detector efficiency. The units of activity for PET data are "PET events per second per pixel." A pixel is a unit of volume, and is a function of the reconstruction pixel size and the slice thickness.

The uncertainty in the number of events in a ROI is currently approximated by a naive estimate which assumes a Poisson distribution for the number of events in a entire region. The uncertainty of the per pixel quantity is taken as the square root of the number of events in the region, divided by the number of

*UNIX is a trademark of Bell Laboratories.

†PDP and RSX are trademarks of Digital Equipment Corporation.

pixels and multiplied by the decay correction factor. This estimate is an order of magnitude too small, and must be compensated for when the parameter uncertainties are reported (see subroutine **dofit** in Appendix A, page 42). A new uncertainty estimation algorithm has been developed correctly propagates errors through the entire reconstruction process, and will yield accurate uncertainties[10].

If taken, blood samples are counted on a gamma well counter with a multichannel analyzer. The well counter data are corrected for radioactive decay, weight of sample, counting duration, and background radiation. The units of these data are "well-counter events per gram per minute." The blood data differ from the PET data by three scale factors:

1) counts per minute vs. counts per second (factor of 60),
2) activity/pixel vs. activity/gm (function of blood density and pixel-volume correspondence), and
3) PET events vs. well counter events (function of the sensitivity of the two devices).

The overall scale factor for converting blood activity data to the corresponding PET activity has been determined empirically and is verified at each experiment by counting and imaging a vial of a radioactive solution.

The program read two input data file formats: ".ROI" files from the PET image analysis program and ".JOB" files from the blood analysis program. The format of these files is shown in Appendix C.

All data files for a given experimental subject, along with comments describing the experimental protocol and a history of the data processing steps, are combined into a single ASCII file in the Software Tools **ar** archive format. This is called the patient study archive. The flow of data from the PET to graphs and analysis results is shown in Figure 7.

5.3. Program Design

The program has three phases: initialization, data reading, and command processing. Command processing includes parameter setting, data fitting, and reporting.

In the initialization phase, subroutine **init** sets global variables to default values: the number of blood and tissue data points is set to zero, their descriptive labels to "Undefined".

In the data reading phase, subroutine **getdat** examines the program's command line arguments for data input instructions. The arguments may specify

1) a file from which data are to be read,
2) a scale factor to apply to the next region-of-interest read, and
3) a region of interest from which to read the blood or tissue measurements. These are specified by their cardinal order in the data file. At this point times, activities, and uncertainties are read and scaled as necessary. Routine **getfun** reads these data by calling format-dependent routines **getroi** or **getjob**. The blood measurements taken in time intervals $(0, t_{b_1})$, (t_{b_1}, t_{b_2}), ... are the time averages over these intervals, and the measurement times T_{B_1}, T_{B_2} are taken to be the middle of the intervals: $T_{B_i} = (t_{b_{i-1}} + t_{b_i})/2$. Likewise, the PET measurement times are taken to be the middle of the image collection intervals.

The command processor subroutine **getcmd** reads commands from the standard input, which is the user's terminal in interactive mode or a file in batch mode. Parameter setting commands are handled by subroutine **setvar**, the display of data and model values by subroutine **dowrit**, and fitting by subroutine **dofit**, which in turn invokes the Marquardt algorithm routine subroutine **marq**, and the parameter value and uncertainty display subroutines **shopar** and **shocov**. Subroutine **setvar** allows the user to select the input and residue model functions, to set model parameters, and to alter the Marquardt parameters τ, ε, ν, etc.

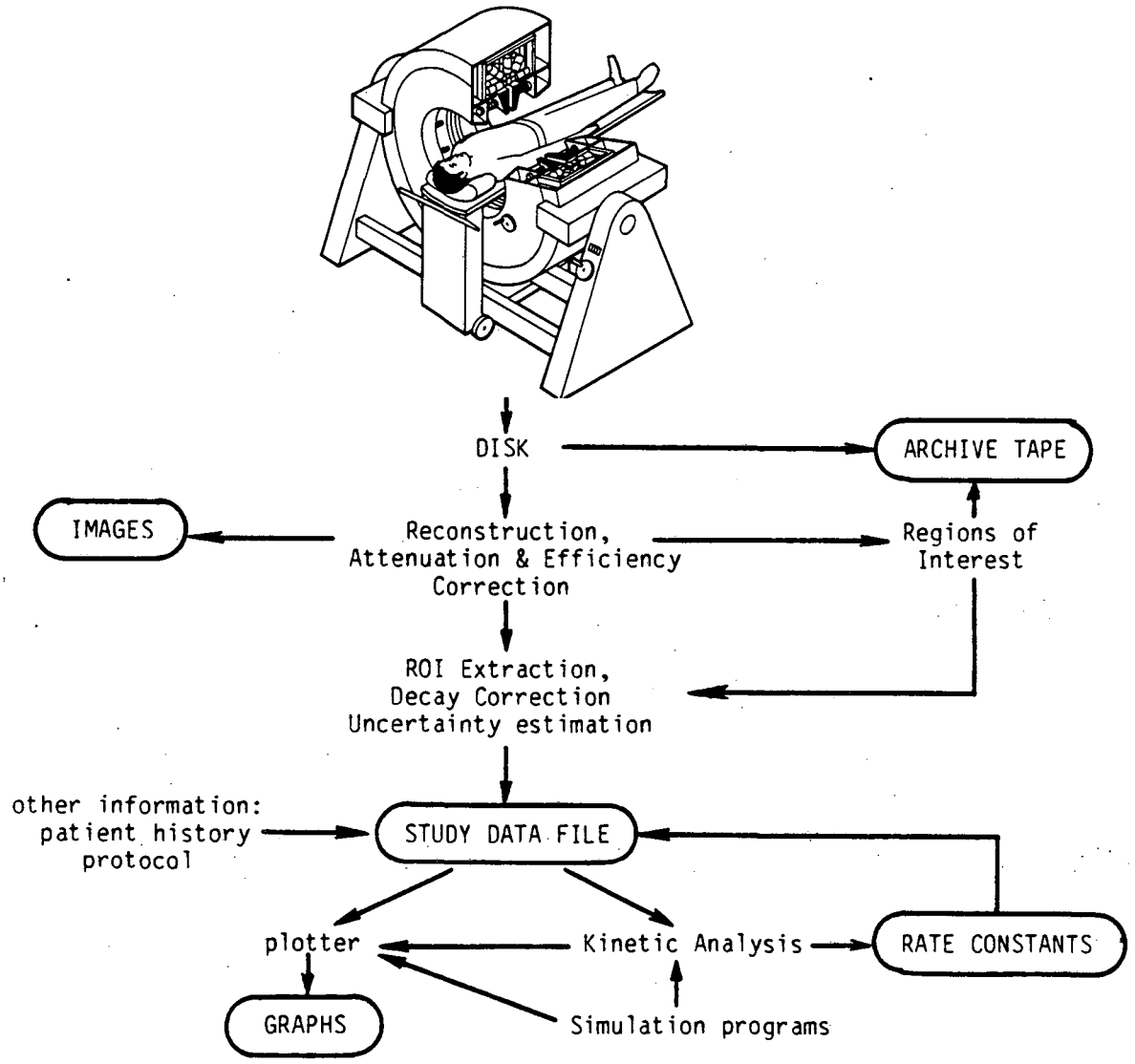


Figure 7. Data Flow.

The input function and residue function models are evaluated by functions **funin** and **funup**. **Funup** contains code to evaluate and convolve the impulse response and input functions as necessary, and to compute the numerical derivatives. Only the impulse response function **fimpls** needs to know the particulars of the compartmental model in use; it can provide the impulse responses of any models of interest. The version of **fimpls** in Appendix A contains five impulse responses; the first is the function in Equation (4.6), and the others will not be discussed here.

Globally accessible data are stored in three named common blocks: model parameter names in **/namcom/** (these are set by **finit**, which is easily changed along with **fimpls**), the current set of model parameters in **/parcom/** and the input and residue measurements and uncertainties in **/datcom/**.

The general outline of the program is shown in Figure 8, with the smaller utility and library routines omitted.

In a typical fitting session, the operator invokes *fit* with a command line specifying the source of the data. Model functions are selected and initial parameters are set with commands of the form "parametername = value." The parameters are fit with the "fit parameter, parameter, ..." command. A file containing the measurements and model values can be created with the "write" command, to be fed to a suitable plotting program.

The program is also useful for simulation of compartment models, given a source of blood and tissue measurement times (from existing data files). The user may specify an input function and residue model, set rate constants, and generate the expected response with the "write" command.

```
fit
    init
        finit
    getdat
        getfun
            getbld
            getroi
                tinit
                datlin
                penter
                pget
    getcmd
        dowrit
            funin
            funup
                funin
                fimpls
                con
        setvar
            whopar
            finit
        dofit
            setmap
                whopar
            marq
                mqchi
                    fun(in or up)
                mqder
                    fun(in or up)
                mqmap
                spdinv
                    dot
            shopar
        shocov
```

Figure 8. Outline of fitting program.

6. Simulations

A program was written to simulate PET data. Data were generated using a biexponential input function with typical model parameters, and compared to the results of the fitting program. The method of simulation is described below.

6.1. Simulation Program

For the biexponential input function (model 1),

$$b(t) = \sum_{j=1}^{2} A_j \, e^{-M_j t} \tag{6.1}$$

the exact solution for the convolution of the input with the three compartment impulse response (Eq. 4.6) is

$$\begin{aligned} q(t) &= b \otimes h(t) \\ &= \sum_{k=1}^{2} \sum_{j=1}^{2} \frac{A_j \, f_k}{\alpha_k - M_j} \left[e^{-M_j t} - e^{-\alpha_k t} \right] \end{aligned} \tag{6.2}$$

For PET images taken over intervals (T_{i-1}, T_i), the simulated measured activities B_i and W_i are averages over the collection interval:

$$\begin{aligned} B_i &= \frac{1}{t_i - t_{i-1}} \int_{t_{i-1}}^{t_i} b(\tau) \, d\tau \\ W_i &= \frac{1}{t_i - t_{i-1}} \int_{t_{i-1}}^{t_i} w(\tau) \, d\tau \end{aligned} \tag{6.3}$$

These functions are:

$$\begin{aligned} B_i &= \sum_{j=1}^{2} \frac{A_j}{(t_i - t_{i-1}) \, M_j} \left[e^{-M_j t_{i-1}} - e^{-M_j t_i} \right] \\ Q_i &= \sum_{j=1}^{2} \sum_{k=1}^{2} \frac{A_j \, f_k}{(\alpha_k - M_j)(t_i - t_{i-1})} \left\{ \frac{e^{-M_j t_{i-1}} - e^{-M_j t_i}}{M_j} - \frac{e^{-\alpha_k t_{i-1}} - e^{-\alpha_k t_i}}{\alpha_k} \right\} \\ W_i &= f_v B_i + (1 - f_v) Q_i \end{aligned} \tag{6.4}$$

The simulation program adds Gaussian errors with mean zero and standard deviations γB_i and γW_i to B_i and W_i respectively, $\gamma \geq 0$. Gaussian noise is generated by projecting the computer's pseudorandom, uniform [0,1) numbers onto a polynomial approximation to the inverse of the normal distribution function. This distribution is not quite realistic, for the relative error γ should be a function of the activity in a region.

6.2. Two-Compartment Model

Data were generated for a two-compartment system, with parameters typical for injections of $H_2{}^{15}O$ in the dog heart.

Input Function:		**Model Parameters:**		**Collection Intervals:**
A_1	50.	k_1	2.35 min^{-1}	24 × 5 sec
M_1	6.2 min^{-1}	k_2	1.75 min^{-1}	18 × 10 sec
A_2	13.	k_3	0.	
M_2	0.12 min^{-1}	k_4	0.	
		f_v	0.15	

Number of Simulations

1 with no noise,
10 each with γ = .03, .06, .09, .12, .15, .18.

When the correct model values were given to the fitting program, the rms error in the model computation was 1.3 percent. The peak error was 1.5 percent and the average error was 0.5 percent. The computed value was always greater than the actual value. This error in the model computation is due to the errors in the piecewise linear approximation of the exponential impulse response and

input function, and to the fact that the interval-center value of the model will be higher than the interval-average value. This is only a problem with the functions are rapidly changing, as with injections of $H_2{}^{15}O$. This systematic error will result in slightly smaller fit values for k_1.

Representative fits to two compartments (k_3 held at 0.) are shown in Figure 9, and fit rate constants are summarized in Figure 10a. The mean and standard deviation of the fit values are shown, along with the standard error of the mean (SEM), the bias (error in mean), and mean estimated uncertainty. The useful comparisons are error in mean to SEM (accuracy of fit), and standard deviation to mean estimated uncertainty (accuracy of uncertainty estimate). Figure 10b shows relative uncertainty (standard deviation / true value) vs. noise.

The k_1 fits show that the estimated uncertainty in the rate constant determination is roughly correct, approximately equal to the sample standard deviation. While there is a small systematic error in the model computation, the error in the rate constant determination is of the same magnitude as the standard error of the mean for these simulations. The program gives even better estimates of the k_2 and f_v values and uncertainties.

In Figure 10b we see that relative uncertainty increases roughly linearly with noise, up to the 18 percent case. The sharp rise in uncertainty at 18 percent is due to the increasing frequency of poor fits observed when noise reaches approximately 20 percent. Some manual coaxing could have improved the bad fits.

When forced to fit three compartments to two compartment data, the program would not converge in the no-noise and several noise-added fits.. The fits were discontinued after the 15 percent category. The partial results shown in Figure 11.

The estimated uncertainties clearly indicate that the program detects the lack of significance of the estimates, especially $k3$ and $k4$.

6.3. Three-Compartment Model

Data were generated for a three-compartment system, with parameters typical for the dog heart in injections of F-18 fluorodeoxyglucose.

Input Function:		**Model Parameters:**		**Collection Intervals:**
A_1	6.0	k_1	0.30 min^{-1}	24 × 5 sec.
M_1	0.82 min^{-1}	k_2	0.50 "	12 × 10 sec.
A_2	4.8	k_3	0.05 "	10 × 60 sec.
M_2	0.03 min^{-1}	k_4	0.006 "	5 × 60 sec
		f_v	0.15	

Number of Simulations:
1 with no noise
10 each with γ = .03, .06, .09, .12, .15, and .18.

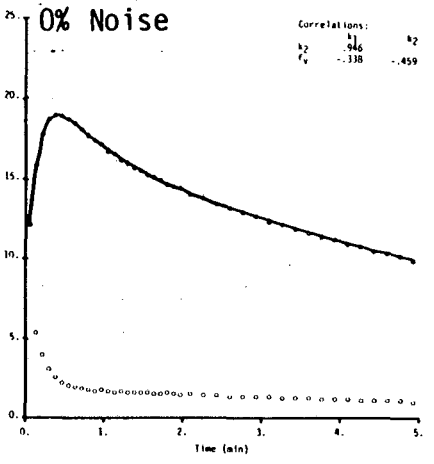
Representative fits are shown in Figure 12, and the results are summarized in Figures 13a and 13b.

In the three compartment case, estimation of k_1, k_2, and f_v and their uncertainties are good even with high noise. The evaluation of k_3 and k_4 is poor at high noise with the sampling intervals used, but the uncertainty estimate grows large as well, so there is no false confidence in the poor estimates. The sharp rise in uncertainty is again seen at 18 percent noise.

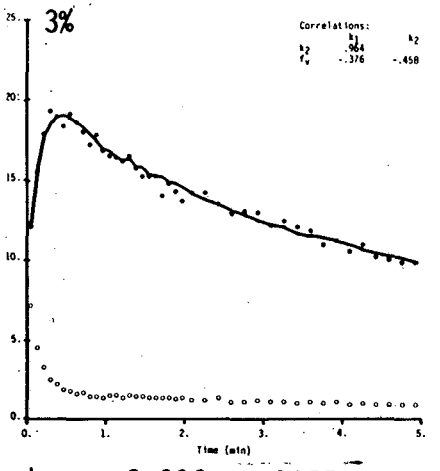
Two Compartment Simulations

$k_1 = 2.35$ $k_2 = 1.75$
$f_v = .15$

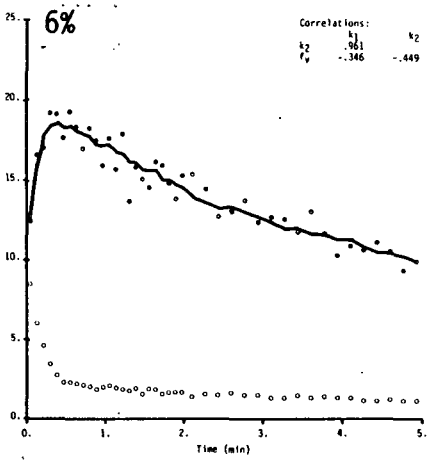
o = blood × fit f_v
● = tissue
— = fit model



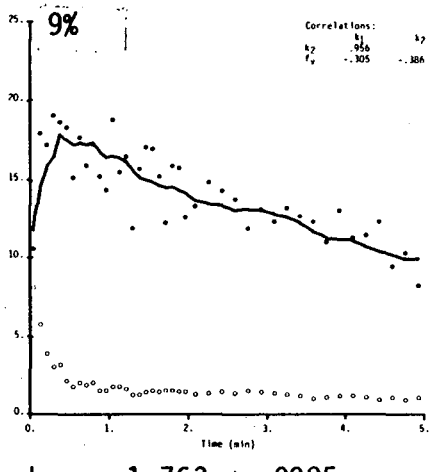
$k_1 = 2.298$
$k_2 = 1.718$
$f_v = 0.150$



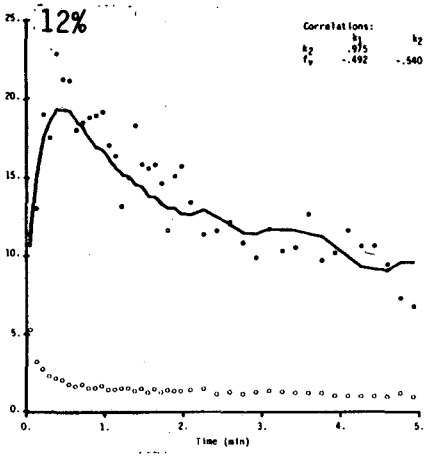
$k_1 = 2.296 \pm .0465$
$k_2 = 1.742 \pm .0396$
$f_v = 0.128 \pm .0073$



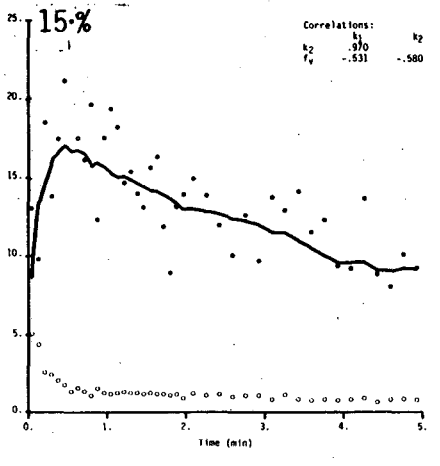
$k_1 = 2.277 \pm .0949$
$k_2 = 1.673 \pm .0793$
$f_v \pm 0.169 \pm .0159$



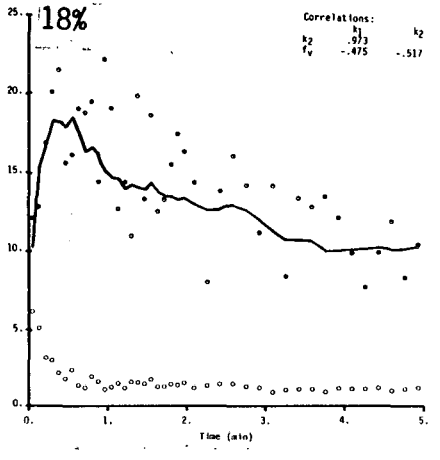
$k_1 = 1.763 \pm .0985$
$k_2 = 1.348 \pm .0870$
$f_v = 0.145 \pm .0180$



$k_1 = 2.862 \pm .2667$
$k_2 = 2.373 \pm .2392$
$f_v = 0.097 \pm .0338$



$k_1 = 2.135 \pm .2344$
$k_2 = 1.749 \pm .2146$
$f_v = 0.115 \pm .0416$



$k_1 = 2.430 \pm .3422$
$k_2 = 2.063 \pm .3178$
$f_v = 0.131 \pm .0496$

Figure 9. Representative Two Compartment Simulation Fits

Figure 10a. Two-Compartment Fits to Two-Compartment Data

Parameter k_1 = 2.35

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
0	2.298		-.052		
3	2.303	.020	-.047	.063	.048
6	2.268	.059	-.082	.188	.095
9	2.078	.084	-.272	.266	.124
12	2.223	.060	-.127	.190	.186
15	2.151	.112	-.199	.355	.229
18	2.292	.192	-.058	.609	.321

Parameter k_2 = 1.75

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
0	1.718		-.032		
3	1.723	.015	-.027	.046	.040
6	1.714	.046	-.036	.146	.081
9	1.599	.068	-.151	.215	.108
12	1.719	.046	-.031	.145	.162
15	1.668	.104	-.082	.329	.200
18	1.786	.141	.036	.445	.280

Parameter f_v = .15

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
0	.150		0.		
3	.152	.003	.002	.010	.008
6	.157	.005	.007	.015	.015
9	.144	.007	-.006	.022	.021
12	.141	.009	-.009	.027	.030
15	.168	.021	.018	.066	.039
18	.184	.030	.034	.094	.050

Figure 10b. Relative Uncertainty vs. Noise
for Two-Compartment Simulations.

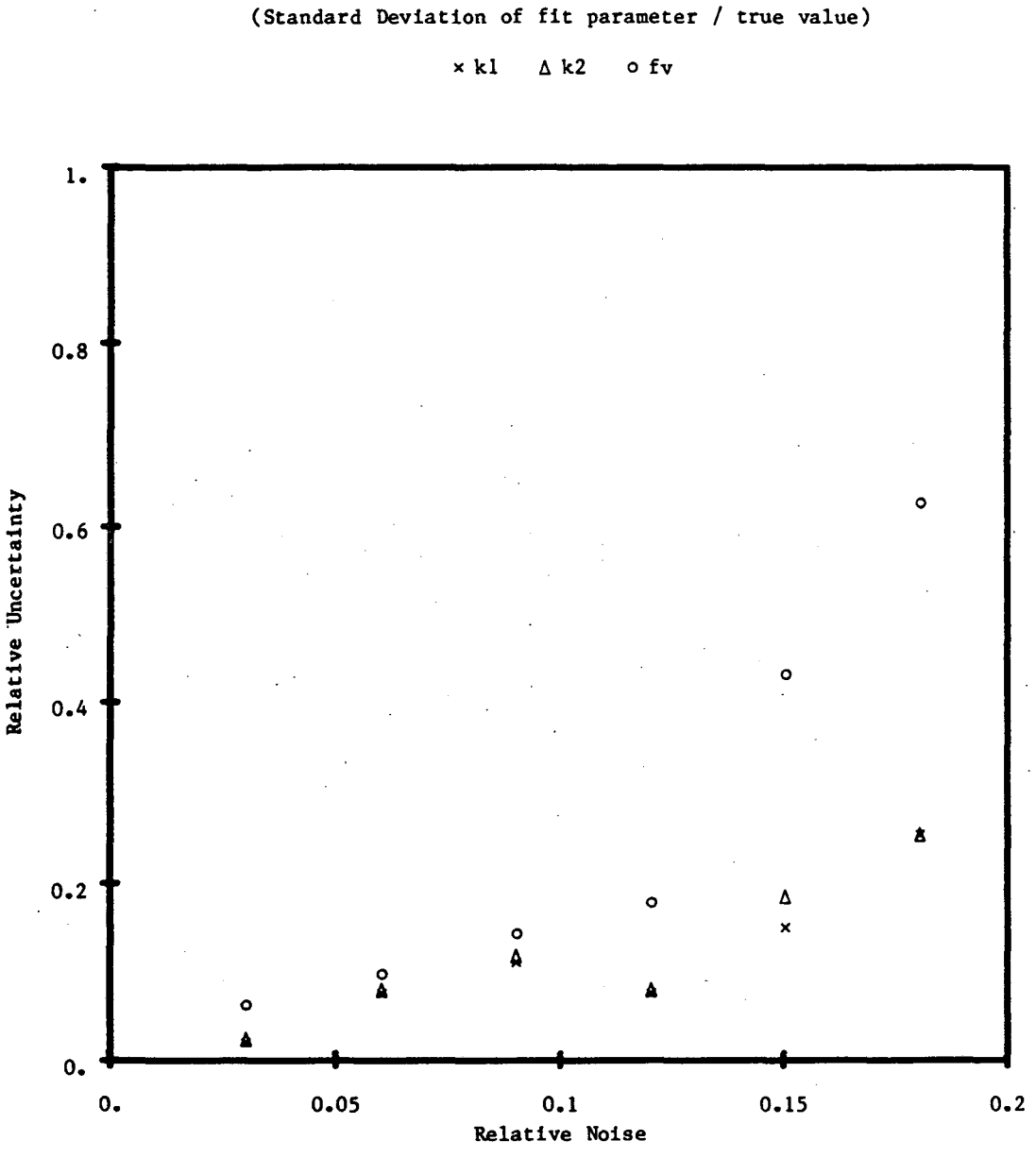


Figure 11. Three-Compartment Fits to Two-Compartment Data

Parameter k_1 = 2.35

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
3	3.97	1.67	1.62	5.28	4.27
6	2.10	0.19	-0.25	0.60	9.31
9	2.23	0.17	-0.12	0.55	7.35
12	3.12	0.73	0.77	2.30	1.90
15	1.98	0.15	-0.37	0.45	4.35

Parameter k_2 = 1.75

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
3	10.40	8.74	8.65	27.65	8.24
6	2.61	1.17	0.86	3.69	42.24
9	1.83	0.30	0.08	0.94	0.50
12	4.81	2.17	3.06	6.87	8.57
15	1.62	0.28	-.12	0.83	2.72

Parameter k_3 = 0.

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
3	-0.44	1.23	-0.43	3.90	11.80
6	1.84	1.82	1.84	5.76	48.40
9	0.09	0.12	0.09	0.39	0.79
12	0.71	0.48	0.71	1.52	59.42
15	-0.26	0.42	-0.26	1.27	19.23

Parameter k_4 (no definite value)

% Noise	Mean Fit Value	SEM	Standard Deviation	Mean Estimated Uncertainty
3	4.66	2.28	7.21	33.67
6	3.53	1.71	5.40	256.
9	2.58	2.32	7.36	1106.
12	5.37	3.03	9.59	2172.
15	1.83	0.98	2.95	114.

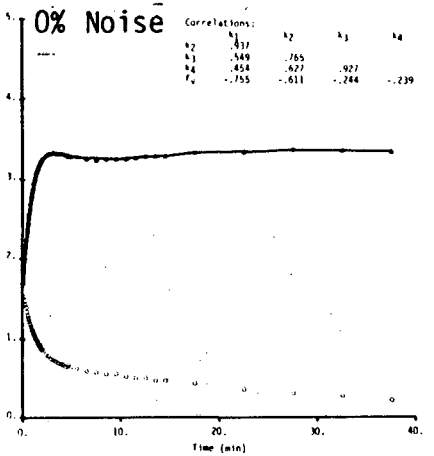
Parameter f_v = 0.15

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
3	.094	.058	-.056	.184	.148
6	.170	.014	.020	.045	.196
9	.160	.010	.010	.032	.154
12	.107	.033	-.043	.104	.079
15	.129	.020	-.021	.060	.122

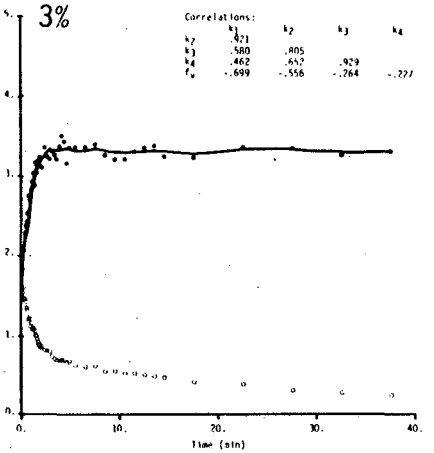
Three Compartment Simulations

k_1 = .3 k_2 = .5
k_3 = .05 k_4 = .006
f_v = .15

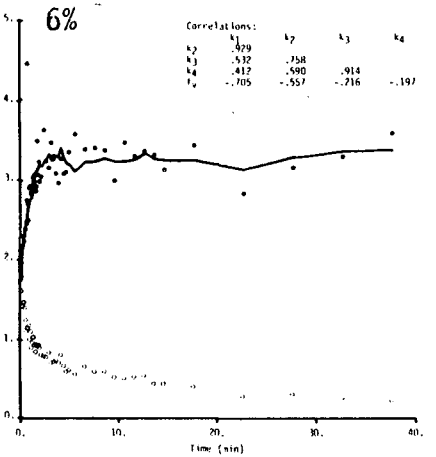
o = blood × fit f_v
● = tissue
— = fit model



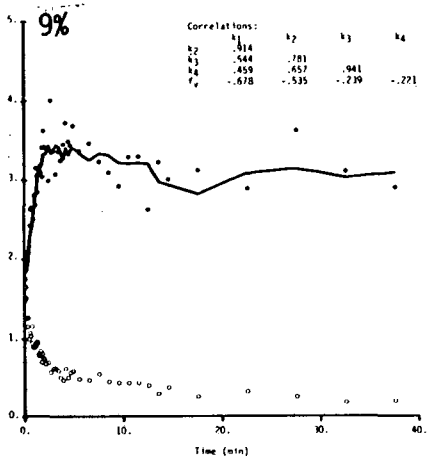
k_1 = 0.3000
k_2 = 0.5006
k_3 = 0.0499
k_4 = 0.0060
f_v = 0.1499



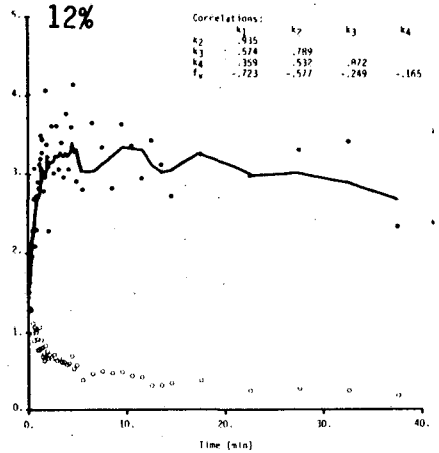
k_1 = 0.2751 ± 0.0075
k_2 = 0.4411 ± 0.0242
k_3 = 0.0481 ± 0.0062
k_4 = 0.0072 ± 0.0061
f_v = 0.1534 ± 0.0032



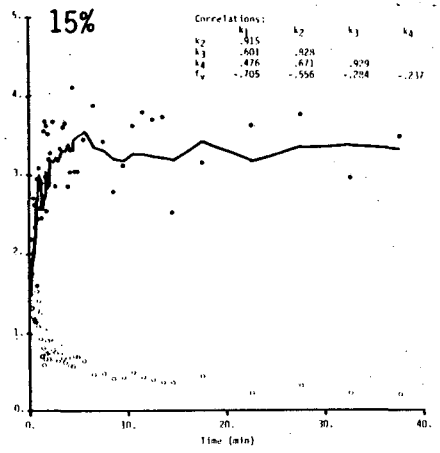
k_1 = 0.3118 ± 0.0185
k_2 = 0.5689 ± 0.0599
k_3 = 0.0566 ± 0.0115
k_4 = 0.0060 ± 0.0097
f_v = 0.1498 ± 0.0067



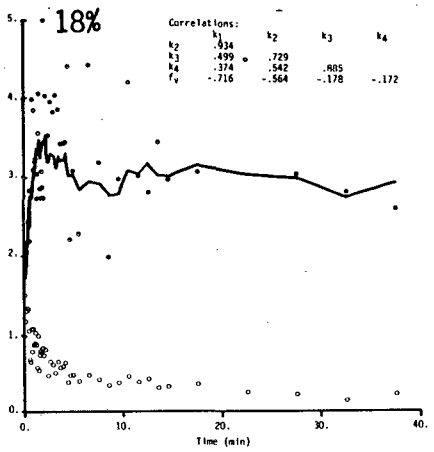
k_1 = 0.3038 ± 0.0206
k_2 = 0.4600 ± 0.0590
k_3 = 0.0313 ± 0.0121
k_4 = -0.0055 ± 0.0175
f_v = 0.1255 ± 0.0087



k_1 = 0.3484 ± 0.0456
k_2 = 0.7351 ± 0.1727
k_3 = 0.0976 ± 0.0379
k_4 = 0.0527 ± 0.0225
f_v = 0.1195 ± 0.0128



k_1 = 0.2407 ± 0.0309
k_2 = 0.3730 ± 0.1062
k_3 = 0.0494 ± 0.0340
k_4 = 0.0083 ± 0.0318
f_v = 0.1360 ± 0.0145



k_1 = 0.4181 ± 0.0794
k_2 = 0.9064 ± 0.2663
k_3 = 0.0771 ± 0.0366
k_4 = 0.0272 ± 0.0244
f_v = 0.1153 ± 0.0207

Figure 12. Representative Three Compartment Simulation Fits.

Figure 13a. Three-Compartment Fits to Three-Compartment Data

Parameter k_1 = .3

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
0	.300		0.		
3	.294	.003	-.006	.010	.008
6	.303	.006	.003	.019	.017
9	.289	.011	-.011	.036	.025
12	.323	.017	.023	.055	.034
15	.311	.014	.011	.043	.045
18	.350	.061	.050	.193	.071

Parameter k_2 = .5

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
0	.501		0.		
3	.483	.007	-.016	.022	.025
6	.516	.020	.016	.063	.054
9	.497	.025	-.003	.078	.080
12	.577	.055	.077	.175	.110
15	.554	.046	.055	.146	.149
18	.800	.274	.300	.866	.273

Parameter k_3 = .05

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
0	.0499		-.0001		
3	.0490	.0016	-.0010	.0050	.0058
6	.0509	.0043	.0009	.0136	.0115
9	.0516	.0062	.0016	.0197	.0177
12	.0514	.0097	.0014	.0308	.0207
15	.0607	.0107	.0107	.0339	.0315
18	.0764	.0307	.0264	.0971	.0404

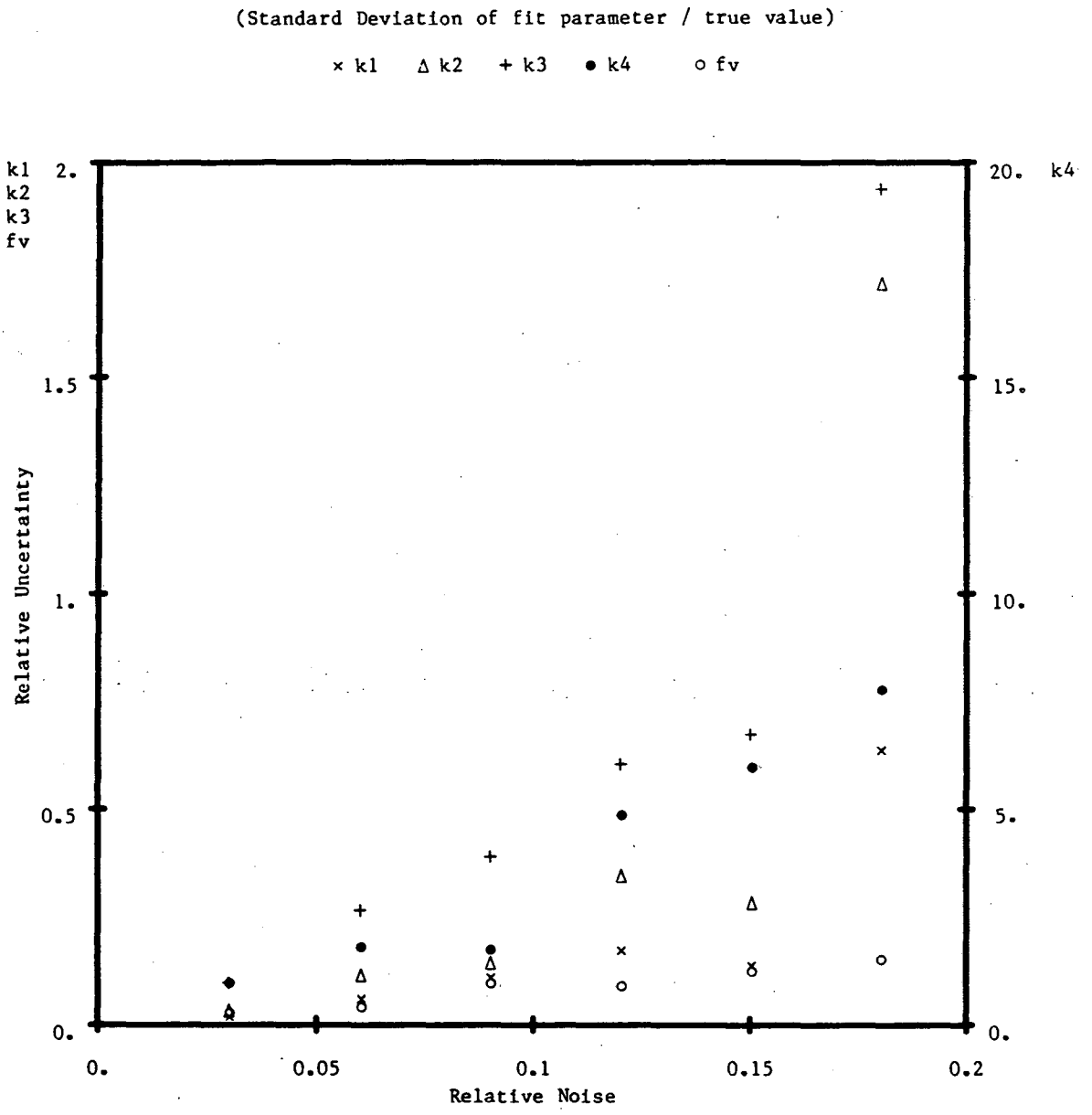
Parameter k_4 = .006

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
0	.00600		0.		
3	.0059	.0020	-.0001	.0062	.0056
6	.0052	.0036	-.0008	.0114	.0108
9	.0014	.0061	-.0046	.0109	.0163
12	.0015	.0094	-.0045	.0296	.0216
15	.0049	.0115	-.0011	.0362	.0288
18	.0108	.0149	.0048	.0470	.0438

Parameter f_v = .15

% Noise	Mean Fit Value	SEM	Error in Mean	Standard Deviation	Mean Estimated Uncertainty
0	.150		0.		
3	.150	.002	0.	.005	.003
6	.149	.002	-.001	.008	.007
9	.148	.005	-.002	.015	.010
12	.124	.005	-.026	.015	.012
15	.139	.006	-.011	.020	.016
18	.111	.008	-.039	.024	.018

Figure 13b. Relative Uncertainty vs. Noise
for Three-Compartment Simulations.

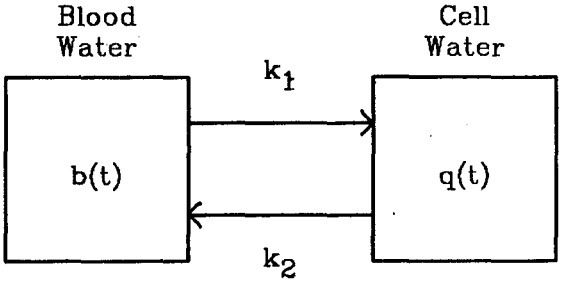


7. Applications to Experimental Data

Below we show examples of fitting compartmental models to actual data from animal experiments. These examples are intended only to demonstrate the program's ability to provide useful data for the investigation of physiological models.

7.1. O-15 Water

O-15 water ($H_2{}^{15}O$) is under consideration as an indicator of blood flow in the brain and heart. We find that O-15 water in the dog heart is well modeled by two compartments, one for blood and one for tissue.



Studies were carried out on mongrel dogs with O-15 water generated in the LBL 80-inch cyclotron. ECG-gated images were collected with the time intervals noted in the 2-compartment simulations in section 6.2, before and after raising myocardial blood flow by injection of Dipyridimole. Actual blood flow in the heart was measured simultaneously by the microsphere reference organ technique[11]. Regions of interest were drawn in the middle of the left ventricle for the input function, and in the left ventricular wall for the residue function. The data and fits are shown in Figure 14, and the results are summarized below.

	Before	6 min. After Dipyridimole	
Actual Flow	0.66	1.47	cc/gm/min
k_1	0.93 ± .14	1.68 ± .28	min^{-1}
k_2	1.01 ± .20	1.68 ± .31	min^{-1}
f_v	0.18 ± .04	0.35 ± .05	

The increase in blood flow was accompanied by a corresponding increase in k_1 and k_2 and f_v.

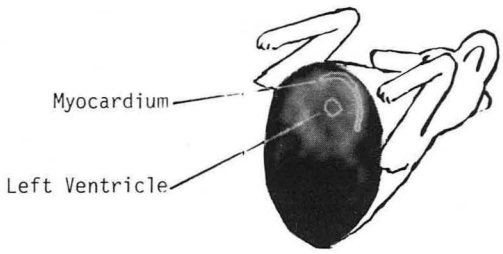
7.2. Fluorodeoxyglucose

[^{18}F] 2-fluoro-2-deoxy-D-glucose (FDG) is a tracer for regional glucose metabolism in the brain and heart [12]. Cell in these organs treat FDG like glucose through the first reaction in glycolysis,

$$\text{glucose} + \text{ATP} \underset{\text{glucose-6-phosphatase}}{\overset{\text{hexokinase}}{\rightleftharpoons}} \text{glucose-6-phosphate} + \text{ADP}$$

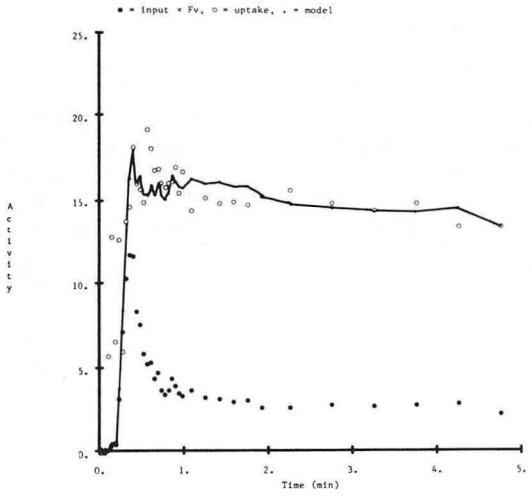
While glucose-6-phosphate is further metabolized, FDG-6-phosphate is not. However, the rates of transport and metabolic reactions of glucose and FDG are similar. A model for FDG kinetics in the brain and heart is

Figure 14. Oxygen-15 Water in the Canine Heart.



^{15}O-Water Image and
Regions of Interest

Before Dipyridimole



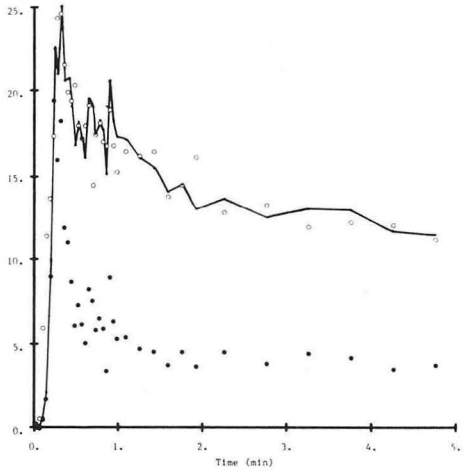
Before ----------

k1 = 9.3381E-01 ± 1.4215E-01
k2 = 1.0079E+00 ± 1.9753E-01
fv = 1.8356E-01 ± 3.7703E-02

Correlation Matrix:

	k1	k2
k2	0.949	
fv	-0.499	-0.453

After Dipyridimole



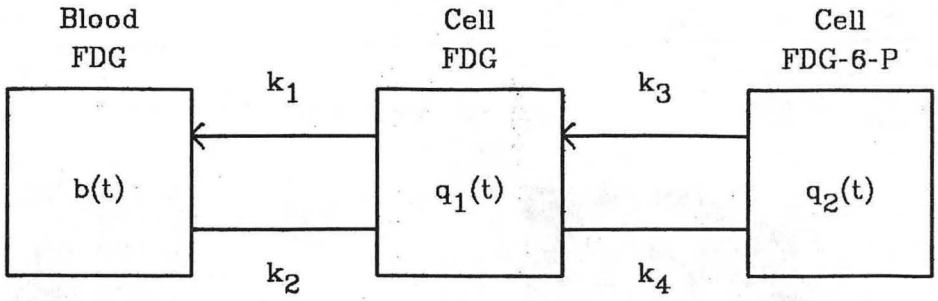
After ----------

k1 = 1.8086E+00 ± 2.8392E-01
k2 = 1.6809E+00 ± 3.0689E-01
fv = 3.4732E-01 ± 4.7471E-02

Correlation Matrix:

	k1	k2
k2	0.961	
fv	-0.428	-0.450

XBB 830-11016



where the first cell compartment represents free FDG and the second cell compartment represents phosphorylated FDG. Rate constants k_1 and k_2 account for the kinetics of glucose transport between the blood and the cell, and rate constants k_3 and k_4 account for the rate of the hexokinase and phosphatase reactions in the cell.

If we assume that the rate constants for glucose are proportional to the rate constants determined for FDG, then the glucose metabolic rate *GMR* in a region of interest is given by

$$GMR = \frac{[\mathrm{Glu}]_p}{LC} \frac{k_1 k_3}{k_2 + k_3} ,$$

where LC is the "lumped constant" which accounts for the difference between glucose and FDG rate constants, and $[\mathrm{Glu}]_p$ is the plasma glucose concentration [13]. We can thus estimate *GMR* from PET determined rate constants, a blood analysis for glucose, and *LC*.

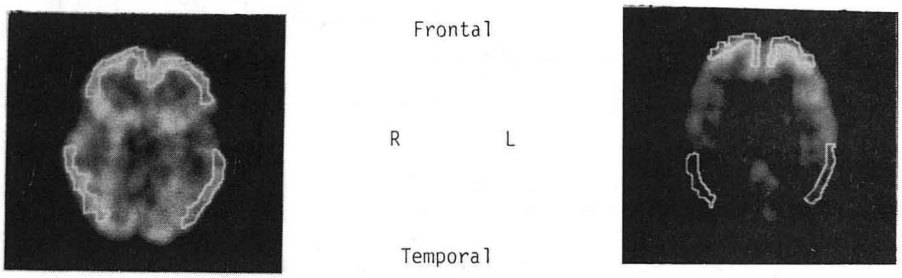
F-18 has a 112 minute half life. Data are collected for 45 minutes after injection, at the time intervals noted in the three compartment simulations in section 6.3. Data were obtained from the right frontal and temporal cortex of two human subjects, one healthy and one with diagnosed Alzheimer's Type Dementia (ATD). Images of FDG distribution after 45 minutes are shown in Figure 15, with the data and fits to the temporal cortex. The slices were obtained at slightly different levels of the brain. The ventricles seen as dark regions in the middle of the ATD brain are not observed in the normal brain; however, the cortex regions are approximately equivalent. Notice the reduced uptake in the temporal cortex of the ATD subject.

The determined rate constants are

	Normal	ATD	
$[\mathrm{Glu}]_p$	102.	98.	
Frontal	**Normal**	**ATD**	
k_1	.121 ± .004	.104 ± .006	min^{-1}
k_2	.198 ± .018	.256 ± .031	min^{-1}
k_3	.0875 ± .0084	.1148 ± .0108	min^{-1}
k_4	.0094 ± .0024	.0412 ± .0020	min^{-1}
f_v	.070 ± .004	.041 ± .005	
GMR·LC	3.78	3.16	mg/min/100g tissue
Temporal	**Normal**	**ATD**	
k_1	.136 ± .005	.062 ± .004	min^{-1}
k_2	.204 ± .018	.196 ± .029	min^{-1}
k_3	.0640 ± .0064	.0708 ± .0128	min^{-1}
k_4	.0012 ± .0028	.0339 ± .0050	min^{-1}
f_v	.075 ± .006	.034 ± .004	
GMR·LC	3.31	1.61	mg/min/100g tissue

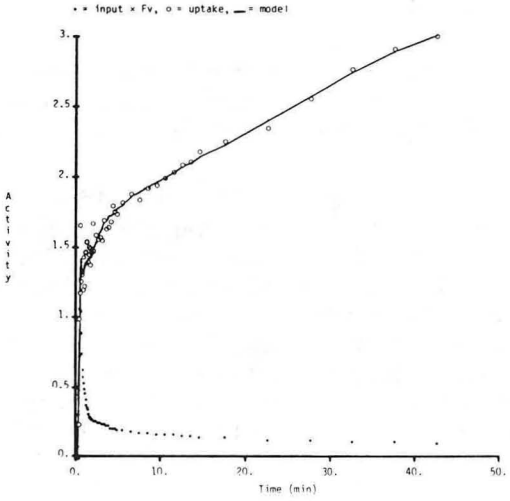
The ATD temporal cortex has substantially lower k_1 and *GMR·LC* values. However, we cannot draw conclusions about *GMR* because we cannot know whether *LC* is altered with Alzheimer's dementia. The ATD cortex also seems to have a lower vascular volume and a higher phosphatase (k_4) activity.

Figure 15. Fluorine-18 Fluorodeoxyglucose in the Human Brain.

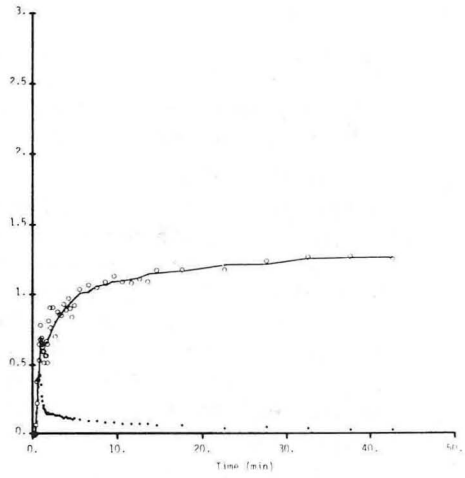


Right Temporal Cortex

Normal



Alzheimer's Type Dementia



Normal ----------

k1 = 1.3643E-01 ± 5.2825E-03
k2 = 2.0403E-01 ± 1.7846E-02
k3 = 6.4026E-02 ± 6.4200E-03
k4 = 1.1498E-03 ± 2.8383E-03
fv = 7.4523E-02 ± 5.5367E-03
t0 = -1.2071E+01 ± 7.5438E-01

Correlation Matrix:

	k1	k2	k3	k4	fv
k2	0.944				
k3	0.618	0.828			
k4	0.492	0.680	0.929		
fv	-0.613	-0.511	-0.218	-0.221	
t0	-0.294	-0.245	-0.095	-0.105	0.577

ATD ----------

k1 = 6.2098E-02 ± 4.0143E-03
k2 = 1.9607E-01 ± 2.9215E-02
k3 = 7.0846E-02 ± 1.2809E-02
k4 = 1.6094E-02 ± 5.0160E-03
fv = 3.3876E-02 ± 4.1648E-03
t0 = -6.8431E+00 ± 9.7430E-01

Correlation Matrix:

	k1	k2	k3	k4	fv
k2	0.941				
k3	0.667	0.867			
k4	0.460	0.660	0.911		
fv	-0.688	-0.541	-0.273	-0.193	
t0	-0.313	-0.237	-0.109	-0.081	0.560

XBB 830-11017

8. Summary

Program *fit* provides good estimates of two- and three-compartment model rate constants from input function and residue functions acquired by PET. It also provides reasonable estimates of the uncertainty and covariance of the fit rate constants.

Program features include

1. easy addition of new models,
2. interactive or batch use, and
3. easy interface with other programs.

Needed improvements in the program fall into four categories:

1. Speed. Computation of the uptake function and derivatives is currently quite inefficient. In the case of models 1, 3, and 5, the impulse response characteristic decay constants are computed at every invokation. Routines **mqchi**, **mqder**, **funup**, and **fimpls** should be rewritten to compute values for all N_w values at once. This rearrangement would also make possible the use of an array processor for further speed increases.
2. Linear-Interpolation Approximations. The piecewise linear approximations of the input and impulse response functions introduce errors in the computation of the residue function. These errors are small but unnecessary. *Funup* could be rewritten to analytically convolve the piecewise linear input function with a vector of n exponentials (n is generally the number of non-vascular compartments in the model). This would likely increase speed as well, as the method of **conv** is not terribly efficient.
3. Sampling. We currently ignore the fact that our measurements are averages over known time intervals (Equation 6.3). We can more accurately model our measurements by using the time-average of $w(t)$ over these intervals.
4. Data Statistics. We ignore the effect of input function noise and input-residue measurement correlation in our least-square function R. This can be corrected when the new ROI uncertainty and correlation estimation algorithm has been implemented in our data collection system. This will require the addition of a new data field in the ROI file format, for covariance with respect to a designated input function region.

References

[1] Bard, Y., *Nonlinear Parameter Estimation*, Academic Press, New York, New York, 1974.

[2] *Ibid*, p. 59.

[3] Marquardt, D.W., "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *J. Soc. Indust. Appl. Math.* 11(2): 431, 1963.

[4] Mason, S.J., Zimmermann, H.J., *Electronic Circuits, Signals, and Systems*, John Wiley & Sons, New York, New York, 1960.

[5] Kernighan, B.W. and Plauger, P.J., *Software Tools*, Addison-Wesley, Reading, Mass., 1976.

[6] Hall, D.E., Scherrer, D.D., and Sventek, J.S., "A Virtual Operating System," *Communications of the ACM* **23**(9): 495, 1980.

[7] Kernighan and Plauger, *op. cit.*, p. 3.

[8] Derenzo, S.E., Budinger, T.F., Huesman, R.H., Cahoon, J.L., and Vuletich, T., "Imaging Properties of a Positron Tomograph with 280 BGO Crystals," *IEEE Trans. Nucl. Sci.* **28**(1), 1981.

[9] Budinger, T.F., Gullberg, G.T., Huesman, R.H., "Emission Computed Tomography" in *Image Reconstruction from Projections, Implementation and Application*, G.T. Herman, ed., Springer-Verlag, 1979, pp 147-246.

[10] Huesman, R.H., "A New Fast Algorithm for the Evaluation of Regions of Interest and Statistical Uncertainty in Computed Tomography," Lawrence Berkeley Laboratory Report LBL-16521, August 1983 (to be published in *Phys. Med. Biol.*).

[11] Domenech, M.D., Hoffman, J.I.E., Noble, M.I.M., Saunders, K.B., Henson, J.R., Subijanto, S., "Total and Regional Coronary Blood Flow Measured by Radioactive Microspheres in Conscious and Anesthetized Dogs," *Circulation Res.* **25**: 581, 1969.

[12] Gallagher, B.M., Fowler, J.S., Gutterson, N.I., MacGregor, R.R., Wan, C., Wolf, A.P., "Metabolic Trapping as a Principle of Radiopharmaceutical Design: Some Factors Responsible for the Biodistribution of [^{18}F] 2-fluoro-2-deoxy-D-glucose," *J. Nucl. Med.* **19**(10): 1154, 1978.

[13] Phelps, M.E., Huang, S.C., Hoffman, E.J., Selin, C., Sokoloff, L., Kuhl, D.E., "Tomographic Measurement of Local Cerebral Glucose Metabolic Rate in Humans with [^{18}F] 2-fluoro-2-deoxy-D-glucose: Validation of Method," *Annals of Neurology* **6**(5): 388, 1979.

Acknowledgements

This work was supported by the Department of Energy under contract number DE-AC03-76SF00098.

I greatly appreciate the support and encouragement of my advisor, Dr. Thomas Budinger.

I want to thank the entire staff of the Donner Research Medicine Group for their friendship and supporting efforts, especially Dr. Ronald Huesman for the Marquardt code, his advice, and patience for my Unix proselytizing. I also extend my appreciation to the team responsible for the human and animal experiments mentioned in Section 7: Katie Brennan, Thomas Budinger, Robert Friedland, John Frisch, Marty Morimoto, Brian Moyer, Mohindar Singh, Julie Twitchell-Mathis, Donald Uber, and Yukio Yano.

```
# fit - fit compartmental models to ring data

DRIVER (fit)
    include datcom          # make sure these commons are in root overlay
    include namcom
    include parcom

    string usestr "Usage: fit [[-sscale] [file] -i[n]] [[-sscale] [file] -u[n]]"

    call query(usestr)

    call init               # set defaults
    call getdat             # read data files named on command line
    call getcmd             # process fit commands

    DRETURN
end
```

```
# fit.h - definitions for FIT

define (VERSION, "FIT V1.3")
                                        # dimensions for
define (MAXNAME,40)                     #    ... character strings
define (MAXLABEL,30)
define (ARGMAX,80)
define (MAXINPAR,6)                     #    ... parameter arrays
define (MAXUPPAR,8)
define (MAXFIT,8)
define (MAXDATA,80)                     #    ... measurement arrays

define (UNKNOWN,-1)                     # the types of variables we set
define (INPUTPAR,1)
define (UPTAKEPAR,2)
define (MARQPAR,3)
define (INPUTFUNCTION,4)
define (UPTAKEFUNCTION,5)
define (NSTEPS,6)

# special definitions for .ROI file routines (tinit,penter,pget)

define (TABLESIZE,1000)                 # dynamic storage: 2K bytes
define (T_SIZE,1)                       # size of table entry
define (T_POINTER,1)                    # string pointer offset in table info
```

```
# datcom - measured data common

real          tblood(MAXDATA)     # times of blood measurements
real          blood(MAXDATA)      # blood activity meas.
real          ublood(MAXDATA)     # uncertainty in blood meas.
integer       nblood              # number of tissue points
logical       btrue               # blood uncertainties correct

real          ttissu(MAXDATA)     # times of tissue measurements
real          tissu(MAXDATA)      # tissue activity meas.
real          utissu(MAXDATA)     # uncertainty of tissue meas.
integer       ntissu              # number of tissue points
logical       ttrue               # tissue uncertainties correct

character     bfile(MAXNAME)      # name of blood source file
real          bscale              # blood scale factor
integer       breg                # blood region number
character     blabel(MAXLABEL)    # label from blood region

character     tfile(MAXNAME)      # name of tissue source file
real          tscale              # tissue scale factor
integer       treg                # tissue region number
character     tlabel(MAXLABEL)    # label from tissue region

common /datcom/ tblood,blood,ublood,nblood,btrue
                ttissu,tissu,utissu,ntissu,ttrue
                bfile,bscale,breg,blabel,
                tfile,tscale,treg,tlabel
```



```
# namcom - names of parameters

                                # stored two characters in one word,
integer innam(MAXINPAR)         # input function model param names
integer  upnam(MAXUPPAR)        # residue function model param names

common /namcom/ innam, upnam
```

```
# parcom - model parameters

real      inpar(MAXINPAR)          # input function parameters
integer ninpar                     # number of input function parameters
integer infun                      # input function selector
integer minfun                     # maximum infun

real      uppar(MAXUPPAR)          # uptake model parameters
integer nuppar                     # number of uptake model parameters
integer upfun                      # uptake function selector
integer mupfun                     # maximum uptake function

integer idebug                     # fitting trace flag
integer nsteps                     # maximum number of iterations allowed

common /parcom/inpar,ninpar,infun,uppar,nuppar,upfun,minfun,mupfun,
                idebug,nsteps
```



```
# tablecom - roi parameter table memory

pointer   table                    # fake table declaration
common  /table/ table
```

```
;fffCATCH - catch terminal interrupt
        .title CATCH
        .ident /19JAN/
;
;       CATCH attaches the specified lun with an AST looking for
;       for CHAR.  When one is received, FLAG is set to fortran logical .TRUE.
;
;       Fortran calling sequence:
;
;       logical flag
;       integer lun
;       byte char
;
;       call catch(flag, [lun], [char])                 ! arm
;       call catch                                      ! disarm
;
;       The default for lun=5, for char=^C, for reatch=.true.
;
;       After catching a CHAR, catch detaches the lun. Catch may
;       also be forced to detach (disarm itself) by calling with no arguments.
;
;       If the attach fails, catch attempts to print a message on LUN.
;
; DEFAULT SETTINGS:
        LUNIT   =       5                       ; default unit for read
        BREAKC  =       3                       ; default break char ^C
;---

        NARGS   =       0                       ; offsets into arg block
        FLAG    =       2
        LUN     =       4
        CHAR    =       6.

        NOARG   =       -1                      ; address of null argument
        TRUE    =       -1                      ; fortran logical values
        FALSE   =       0

        .mcall qio$s, astx$s
        .globl CATCH, note

; pure code
        .psect $R.ROI,I,RO,CON,REL,LCL

;---------------------------------- CATCH ----------------------------

CATCH:  tstb    NARGS(r5)                       ; if no arguments
        ble     disarm                          ; disable thyself

        cmp     FLAG(r5), #NOARG                ; flag wasn't passed
        beq     disarm                          ; so this is a disable call
        mov     FLAG(r5), flagp                 ; get pointer to flag
        mov     #FALSE, @flagp                  ; clear it

        cmp     NARGS(r5), #2                   ; see if LUN was passed
        blt     doqio                           ; no lun and no char - go qio
        cmp     LUN(r5), #NOARG                 ; see if LUN was passed
        beq     2$
        mov     @LUN(r5), luntt                 ; save the unit number

2$:     cmp     NARGS(r5), #3                   ; see if char arg is there
        blt     doqio
        cmp     CHAR(r5), #NOARG
        beq     doqio
        movb    @CHAR(r5), break                ; store break character
```

```
doqio:  cmp     armed, #TRUE                    ; see if we have to do this
        beq     done                            ; if it's done, don't redo it
        mov     #TRUE, armed

        jsr     pc, ttydet                      ; detach tty

        QIO$S   #IO.ATA,luntt,,,#isb,,<#gotch>  ; attach keyboard
        bcs     bad
        cmp     isb, #IS.SUC
        beq     done

bad:    mov     #badmsg, -(sp)                  ; print error message
        mov     #1, -(sp)                       ; with call note("[CATCH]...")
        mov     sp, r5
        jsr     pc, note
        tst     (sp)+
        tst     (sp)+

;------------------------------------ DISARM ------------------------------
; DISARM - undo catchc

disarm: cmp     armed, #FALSE                   ; if not armed, just return
        beq     done

        mov     #FALSE, armed
        QIO$S   #IO.DET,luntt                   ; issue detach from lun
        call    ttyatt
done:   rts     pc

;------------------------------------ GOTCH -------------------------------
; gotch - ast routine called when a character comes in

GOTCH:  bicb    #200, (sp)                      ; clear parity
        cmpb    (sp), break                     ; see if it's what we want
        bne     1$                              ; discard if not
        mov     #TRUE, @flagp                   ; yes - set users flag
        jsr     pc, disarm                      ; issue detach from lun
1$:     tst     (sp)+                           ; discard character
        ASTX$S                                  ; exit ast routine
        halt                                    ; should never happen

;------------------------------------ DATA --------------------------------
; impure data
        .psect $r.rwd,D,RW,CON,REL,LCL

armed:  .word   FALSE                           ; flag indicating pending read
flagp:  .word   0                               ; address of user's flag
luntt:  .word   LUNIT                           ; logical unit to read
isb:    .blkw   2                               ; QIO success buffer
break:  .byte   BREAKC                          ; character we're looking for
        .even


; pure data
        .psect $r.rod,D,RO,CON,REL,LCL

badmsg: .asciz  /[CATCH] Attach LUN failed/
        .even

        .end
```

```
# con - convolution of two sampled functions.

real function con (a, b, ta, tb, time, a0, b0, ta0, tb0, na, nb)
integer na, nb
real a(na), b(nb), ta(na), tb(nb), time, a0, b0, ta0, tb0
#
#       Perform onvolution of functions a and b
#
#       Evaluates a*b(time) where a(s) = linear interpolation
#       of points (ta0,a0), (ta(i),a(i))  and b(s) = linear interpolation
#       of points (tb0,b0), (tb(i),b(i)).  Any ta(i) less than ta0 or
#       tb0 less than tb0 are ignored a is the maximum index in a, ta.
#       Nb is the maximum index in b, tb.
#
#       We integrate by summing a series of trapezoidal panels, delimited
#       by the known time points (ta0, ta(1), ... ta(na)) and
#       (time-tb0, time-tb(1), ... time-tb(nb)
#       The time interval of a panel is t = ts to te.
#       For function a the panel is (ts,as), (te,ae).  For function
#       b the panel is (ts,bs), (te,be).  The logical flags tell
#       whether we know the functions exactly at the start and end points.
#       If we don't know, we interpolate.

        real tend, ts, te, as, ae, bs, be, sum
        logical enda, endb
        integer ia, ib

# begin con ------------------------------------------------------------
        tend = time - tb0                       # upper limit of integration
        sum = 0.                                # initialize integral sum
        ia  = 1                                 # starting pointers to functions
        ib  = nb
        ts = ta0                                # starting time
        as = a0                                 # we know start of a

        while ((time-tb(ib)) < ts & ib > 1)     # find where to start b
            ib = ib - 1
        if (ib < nb)                            # extrapolate from first point
            ib = ib + 1                         # before ta0
        bs = b(ib) + (b(ib-1)-b(ib))*(ts-(time-tb(ib)))/(tb(ib)-tb(ib-1))

        while (ts < tend) [                     # INTEGRAL LOOP
            while (ta(ia) <= ts & ia < na)      # find first time after current
                ia = ia + 1                     # time, in both a and b series
            while ((time - tb(ib)) <= ts & ib > 1)
                ib = ib - 1

            te = time - tb(ib)                  # take smallest of these
            if (ta(ia) < te | te <= ts)         # as panel limit
                te = ta(ia)
            if (te <= ts)                       # but make sure we advance
                te = tend

            enda = te == ta(ia)                 # find out which we know
            endb = te == (time - tb(ib))

            if (enda)                           # know it directly
                ae = a(ia)
            else if (ia == 1)                   # interpolate from (ta0,a0)
                ae = a0 + (a(1)-a0)*(te-ta0)/(ta(1)-ta0)
            else                                # interpolate between 2 points
                ae = a(ia-1) + (a(ia)-a(ia-1))*(te-ta(ia-1))/(ta(ia)-ta(ia-1))
```

```
            if (endb)                                   # know it directly
                be = b(ib)
            else if (te < (time - tb(1)))               # interpolate between 2 points
                be = b(ib+1) +_
                    (b(ib)-b(ib+1))*(te-(time-tb(ib+1)))/(tb(ib+1)-tb(ib))
            else if (te == tend)                        # it's the endpoint
                be = b0
            else                                        # interpolate from (tb0, b0)
                be = b(1) + (b0-b(1))*(te-(time-tb(1)))/(tb(1)-tb0)

                                                        # integral of panel
            panel=(te-ts)*(1./3.*(ae-as)*(be-bs) + 0.5*(bs*(ae-as)+_
                as*(be-bs)) + as*bs)

            sum = sum + panel                           # add panel to sum
            as = ae
            bs = be
        ]

        return(sum)
end
```

```
# datlin - read ROI data line

integer function datlin (line, fdes, size)
character line(ARB)
filedes fdes
integer size(2)
#
# reads a data line from archive file 'fdes'.  Ignores blank lines and comments,
# and enters parameter-setting comment lines into the table.
# Returns EOF on end of file.  This function is just like getlin except
# it will not return blank or comment lines.
#
        ext_func integer length, agtlin
        ext_subr skipbl, penter
        character name(MAXNAME)
        integer info(T_SIZE), i, j, last

        while (agtlin(line, fdes, size)!=EOF) [ # try reading a line
?       call fprint(STDERR, "[DATLIN] read: %s", line)

            last = 0
            for (j=1; line(j) != EOS; j=j+1)        # find last nonwhite character
                if (line(j)!=BLANK & line(j)!=NEWLINE & line(j)!=TAB)
                    last = j

            line(last+1) = EOS                      # trim trailing whitespace

            j = 1
            call skipbl(line, j)                    # look at first nonblank char
            if (line(j) == EOS)                     # blank line - ignore it
                next
            if (line(j) != '#') [                  # not a comment:
?               call fprint(STDERR, "[DATLIN] *@n")
                return(OK)                          # return triumphant.
            ]

            j = j + 1                               # look at next nonblank char
            call skipbl(line, j)
            if (line(j) != '%')                     # this is just a comment
                next

            i = 1
            for (j=j+1; line(j) != '%'; j=j+1) [
                if (line(j) == EOS | i == MAXNAME)
                    next 2                          # no closing %, forget it
                name(i) = line(j)                   # extract the parameter name
                i = i + 1
            ]
            name(i) = EOS

            j = j + 1
            call skipbl(line, j)            # find beginning of definition
            call penter(name, line(j))      # enter definition
        ]
        return(EOF)
end
```

```
# dofit - perform fit on parameters

subroutine dofit (line, j)
character line(ARB)
integer j
#
#       line(j...) is a list of parameters to fit.  It is picked apart
#       by setmap.  We print initial values, call marquardt, print
#       results.
#
        ext_func filedes open
        ext_func integer isatty
        character val(MAXNAME)
        integer map(MAXFIT), nparm, which, flags(MAXFIT)
        real uncert(MAXFIT), cov(MAXFIT*MAXFIT)
        filedes ttydes
        external funin, funup
        logical quit                # set by typing control-C
        logical true                # true uncertainties for our fit?
        common /quit/ quit

        include datcom
        include parcom

        call setmap(line(j), map, nparm, which)

        if (nparm == 0) [
            call fprint(STDERR,"No paramaters to fit@n")
            return
        ]

        call fprint(STDOUT, "#Initial Conditions:@n@n")
        call shopar(STDOUT, NO, uncert, NO, flags)
        call fprint(STDOUT, "@n# Fit %s", line(j))

        chi   = 0.
        chi0  = 0.
        istep = 0
        quit  = .false.

        if (isatty(STDIN) == YES)               # lun for control-C is terminal
            ttydes = STDIN
        else
            ttydes = open("TI:", READ)

        call catch(quit, ttydes)

        if (which == INPUTPAR)
            call marq(funin,ninpar,inpar,uncert,cov,nparm,map,nblood,
                tblood,blood,ublood,nsteps,chi0,istep,chi,ierr,flags,idebug)
        else
            call marq(funup,nuppar,uppar,uncert,cov,nparm,map,ntissu,
                ttissu,tissu,utissu,nsteps,chi0,istep,chi,ierr,flags,idebug)

        call catch
        if (ttydes != STDIN)
            call close(ttydes)

        call rtoe(chi0, val, 11, 4)
        call fprint(STDOUT, "@n# Results:@n# Initial chi square: %s@n", val)
        call rtoe(chi, val, 11, 4)
        call fprint(STDOUT, "# Final   chi square: %s  after %d iterations@n@n",
          val, istep)

        if (ierr != 0) [
```

```
            call fprint(STDOUT, "# *** Marquardt error %d: ", ierr)
            select (ierr) [
                case -1:
                    call fprint(STDOUT, "Parameter setup@n")
                case 1:
                    call fprint(STDOUT, "Too many iterations@n")
                case 2:
                    call fprint(STDOUT, "Matrix invert while stepping@n")
                case 3:
                    call fprint(STDOUT, "Matrix invert after convergence@n")
                case 4:
                    call fprint(STDOUT, "Terminated by user before convergence@n")
                default:
                    call fprint(STDOUT, "?@n")
            ]
        ]

        if (which == INPUTPAR) [
            npar = ninpar
            ndat = nblood
            true = btrue
        ]
        else [
            npar = nuppar
            ndat = ntissu
            true = ttrue
        ]

        ndf = ndat - nparm                  # degrees of freedom
        call fprint(STDOUT, "# Number of degrees of freedom: %d", ndf)

        if (!true) [                                    # have to fudge uncertainties
            sc  = sqrt(chi / float(ndf))                  # pretend model fit: chi=ndf
            do i = 1, npar                                # scale uncert, cov
                uncert(i) = uncert(i) * sc
            do i = 1, npar**2
                cov(i) = cov(i) * sc
            call fprint(STDOUT,"  -- Uncertainties fudged@n")
        ]
        else
           call putch(NEWLINE, STDOUT)

        call shopar(STDOUT, which, uncert, which, flags)
        call shocov(cov, npar, which)

        return
end
```

```
C DOT - Compute Dot-Product of Vectors

      FUNCTION DOT(A,B,N)
      DIMENSION A(1),B(1)

      D = 0.
      DO 10  I = 1,N
          D = D + A(I)*B(I)
10    CONTINUE
      DOT = D

      RETURN
      END
```

```
# dowrit - write input or uptake data & model to file

subroutine dowrit (line, j)
character line(ARB)
integer j
#
#       line(j...) is a write-data command.  Format is:
#               [INput] [,] [UPtake] [> file]
#
#
        ext_func integer gettok, equal
        ext_func filedes open
        character var(MAXNAME), filnam(MAXNAME)
        filedes fdes
        integer which, mode, ndat
        real time, meas, uncert, model, inp, par
        external funin, funup

        include parcom
        include datcom

        fdes = STDOUT
        mode = WRITE

        if (gettok(var, line, j) == EOF) [
100         call fprint(STDERR, "*** Usage: write in|up [>|>> file]@n")
            return
        ]

        if (var(1) == 'i' & var(2) == 'n') [
            which = INPUTPAR
            npar = ninpar
        ]
        else if (var(1) == 'u' & var(2) == 'p') [
            which = UPTAKEPAR
            npar = nuppar
        ]
        else
            goto 100

        call gettok(var, line, j)
        if (var(1) == '>') [                                    # redirection
            if (line(j) == '>') [
                mode = APPEND
                j = j + 1
            ]

            filnam(1) = EOS
            while (gettok(var, line, j) != EOF)                 # remaining tokens are
                call concat(filnam, var, filnam)                # tacked onto file name

            if (length(filnam) <= 0)                            # missing name
                goto 100

            fdes = open(filnam, mode)
            if (fdes == ERR) [
                call fprint(STDERR, "Can't write to %s@n", filnam)
                return
            ]
        ]

        call rtoe(bscale, var, 10, 3)
        call fprint(fdes,"Input: %s - %s (reg. %d * %s Model %d@n",
            bfile, blabel, breg, var, infun)
```

```
if (which == UPTAKEPAR) [
    call rtoe(tscale, var, 10, 3)
    call fprint(fdes,"Uptake: %s - %s (reg. %d * %s)  Model %d@n",
        tfile, tlabel, treg, var, upfun)
]
else
    call putch('@n', fdes)

for (i = 1; i <= npar; i=i+1) [
    call getnam(var, which, i)                    # parameter name
    call fprint(fdes, "%s = ", var)
    if (which == INPUTPAR)
        par = inpar(i)
    else
        par = uppar(i)
    call rtoe(par, var, 11, 3)                    # and value
    call putlin(var, fdes)

    if (mod(i,4) == 0)
        call putch('@n', fdes)
    else
        call putlin("  ", fdes)
]
if (mod(i,4) != 1)
    call putch('@n', fdes)

if (which == INPUTPAR) [
    call fprint(fdes,"@n time       input       uncert      in_model@n")
    ndat = nblood
]
else [
    call fprint(fdes,
      "@n time    uptake      uncert      up_model    input@n")
    ndat = ntissu
]

do i = 1, ndat [
    if (which == INPUTPAR) [
        time   = tblood(i)
        meas   = blood(i)
        uncert = ublood(i)
        call funin(.false.,inpar,time,model)
    ]
    else [
        time   = ttissu(i)
        meas   = tissu(i)
        uncert = utissu(i)
        call funup(.false.,uppar,time,model)
        call funin(.false.,inpar,time-uppar(6),inp)
    ]

    call rtof(time, var, 7, 1)                    # time
    call putlin(var, fdes)

    call rtoe(meas, var, 11, 3)                   # measured value
    call putlin(var, fdes)

    call rtoe(uncert, var, 11, 3)                 # uncertainty
    call putlin(var, fdes)

    call rtoe(model, var, 11, 3)                  # model value
    call putlin(var, fdes)
```

```
            if (which == UPTAKEPAR) [                    # show input function
                call rtoe(inp, var, 11, 3)               # if doing uptake
                call putlin(var, fdes)
            ]

            call putch('@n', fdes)
        ]

        if (fdes != STDOUT) [
            call close(fdes)
            call fprint(STDOUT,"# Wrote data/fit list to file '%s'@n", filnam)
        ]

        return
end
```

```
real function fimpls(par, t)
real par(ARB), t
#
#       Impulse function for compartmental models
#
#       t = time of evaluation in terms of input function time, in sec.
#       This routine evaluates several different model input functions:
#       upfun = 1       three compartments in a row e.g. (FDG)
#               2       three exponentials (bastard function for 3, below)
#               3       four compartments
#               4       two compartments, k1 both ways
#               5       four compartments, kb held equal to ka

        real ka6, k16, p0, p1, p2, a1, a2, a3, f1, f2, f3, time
        include parcom

        define (K1,par(1))              define (Fv,par(5))
        define (K2,par(2))              define (KA,par(7))
        define (K3,par(3))              define (KB,par(8))
        define (K4,par(4))              define (Fe,par(9))

#       inline functions for case 3: numerator polynomials
        anume(s)  = ka6*(s**2 + (K2+K3+K4)*s + K2*K4)
        anum12(s)  = ka6*K1*(s + K3 + K4)

        if (t < 0.)                             # return 0 for t < 0
            return(0.)

        time = -t/60.
        k16  = K1*(1.-Fv)/60.                   # a frequently needed number

        select (upfun) [
            case 1:
#       par(1) = k1 (blood fdg <=> tissue fdg)
#       par(2) = k2
#       par(3) = k3 (tissue fdg <=> phosphorylated)
#       par(4) = k4
#       par(5) = Fv (vascular partial volume)

                if (time == 0.)
                    return(k16)

                beta1 = K2 + K3 + K4
                beta2 = beta1**2 - 4.*K2*K4
                if (beta2 <= 0.) [
                    call fprint(STDERR,"***Unable to solve roots@n")
                    return(0.)
                ]

                beta2 = sqrt(beta2)
                alpha1 = (beta1 - beta2)*.5
                alpha2 = (beta1 + beta2)*.5
                f1 = k16 * (K3+K4-alpha1)/beta2
                f2 = k16 - f1

                return(f1*exp(time*alpha1) + f2*exp(time*alpha2))

            case 2:
#               par(1) = f1  coefficient for first exponential
#               par(2) = k1  rate constant for first exponential
#               par(3) = f2  (second exp)
#               par(4) = k2
```

```
#               par(7) = f3  (third exp)
#               par(8) = k3
#               par(5) = Fv  (vascular partial volume)

                if (time == 0.)
                    return((par(1)+par(3)+par(7))*(1.-Fv)/60.)

                return(  (par(1)*exp(time*par(2))+_
                          par(3)*exp(time*par(4))+_
                          par(7)*exp(time*par(8))  ) *(1.-Fv)/ 60.)

            case 3:
            case 5:
#               par(1) = k1 (extracellular space <=> tissue)        \
#               par(2) = k2                                         } old k's
#               par(3) = k3 (tissue fdg <=> phosphorylated)         /
#               par(4) = k4
#               par(7) = ka (blood <=> extracellular space)         \
#               par(8) = kb (always equal to ka in model 5)         /  new k's
#               par(5) = Fv: vascular partial volume
#               par(9) = Fe: extracellular partial volume

#               take care of time unit dependence of Ka: convert 1/min to 1/sec
                ka6 = KA / 60.

                if (time == 0.)
                    return(ka6*Fe)
                time = -time                        # a's are already < 0
                if (upfun == 5)                     # model 5: ka and kb equal
                    KB = KA

                p2 = KB+K1+K2+K3+K4                 # coefficients of characteristic eqn
                p1 = (K1+KB)*(K3+K4)+K2*(KB+K4)
                p0 = KB*K2*K4
                call rt3(p2,p1,p0,a1,a2,a3,ierr)# find its roots

                if (ierr < 0) [
                    call remark("***Unable to solve roots in fimpls")
                    return(1.0e+10)
                ]
                if (ierr > 0)
                    call remark("***Equal roots, hope that's ok")

                ea1 = exp(time*a1)
                ea2 = exp(time*a2)
                ea3 = exp(time*a3)

                Fc = 1. - Fv - Fe                   # cell volume
                f1 = (Fe*anume(a1)+Fc*anum12(a1))/((a1-a2)*(a1-a3))
                f2 = (Fe*anume(a2)+Fc*anum12(a2))/((a2-a1)*(a2-a3))
                f3 = (Fe*anume(a3)+Fc*anum12(a3))/((a3-a1)*(a3-a2))
                return(f1*ea1+f2*ea2+f3*ea3)

            case 4:                                 # two compart, k1 both directions
                if (time == 0.)
                    return(k16)
                return(k16*exp(time*K1))

            default:
                call error("*** in fimpls, can't happen")
        ]
end
```

```
# finit - initialize parameter names and values

subroutine finit

# this routine defines the names of the parameters and the maximum number
# of parameters.  Initialization must be done by assignment statements.
# The routine is called whenever the input or uptake function is changed.

        include datcom
        include parcom
        include namcom

        minfun = 4
        mupfun = 5

        ninpar    = 5
%       innam(1) = 'a1'
%       innam(2) = 'm1'
%       innam(3) = 'a2'
%       innam(4) = 'm2'
%       innam(5) = 'ti'
        innam(6) = 0

%       upnam(5) = 'fv'
%       upnam(6) = 't0'

        select (upfun) [
            case 1:
                nuppar    = 6
%               upnam(1) = 'k1'
%               upnam(2) = 'k2'
%               upnam(3) = 'k3'
%               upnam(4) = 'k4'
            case 2:
                nuppar    = 8
%               upnam(1) = 'f1'
%               upnam(2) = 'k1'
%               upnam(3) = 'f2'
%               upnam(4) = 'k2'
%               upnam(7) = 'f3'
%               upnam(8) = 'k3'
            case 3:
            case 5:
                nuppar    = 9
%               upnam(1) = 'k1'
%               upnam(2) = 'k2'
%               upnam(3) = 'k3'
%               upnam(4) = 'k4'
%               upnam(7) = 'ka'
%               upnam(8) = 'kb'
%               upnam(9) = 'fe'
            case 4:
                nuppar    = 6
%               upnam(1) = 'k1'
%               upnam(2) = '  '
%               upnam(3) = '  '
%               upnam(4) = '  '
            default:
                call error("*** in finit, can't happen")
        ]
        return
end
```

```
# funin - compute input function (& maybe derivatives)

subroutine funin (tderiv, par, t, y, tder, dy)
logical tderiv, tder(ARB)
real par(ARB), t, dy(ARB)
#
#       input function for fitting
#       choices are:
#               1 biexponential         a1   exp(-m1 T)    + a2  exp(-m2 T)
#               2 time*biexponential    a1 T exp(-m1 T)    + a2 T exp(-m2 T)
#               3 gamma variate         a1 T exp(-m2 T**2) + a2 T exp(-m2 T**2)
#               4 linear interpolation from input measurement
#       where T = (t-Ti)/60.  This Ti shift does not affect model 4.
#
#       units: a1, a2 - units of the input measurements (cts/min/cc)
#              m1, m2 - 1/min
#              t, Ti  - seconds
#
        include parcom
        include datcom
        real time, tk, em1, em2
        integer lo, hi, try

        define (A1, 1)
        define (M1, 2)
        define (A2, 3)
        define (M2, 4)
        define (Ti, 5)

        if (infun == 4) [                       # linear interpolation
            if (t < 0.)
                y = 0.
            else [
                lo = 1; hi = nblood             # binary search for nearest time
                while (lo < hi) [
                    try = (lo + hi) / 2
                    if (tblood(try) < t)
                        lo = try + 1
                    else
                        hi = try - 1
                ]

                if (tblood(lo) > t)                     # we want to interpolate
                    lo = lo - 1                         # between (lo) and (lo+1) so
                if (lo <= 0)                            # make sure lo points where it
                    lo = 1                              # should.
                else if (lo >= nblood)
                    lo = nblood - 1
                                                        # interpolate
                y = blood(lo) + (t-tblood(lo)) * (blood(lo+1)-blood(lo))_
                    / (tblood(lo+1)-tblood(lo))
                if (y < 0.)
                    y = 0.                              # no negative numbers
            ]

            if (tderiv)
                do i = 1, ninpar
                    if (tder(i))
                        dy(i) = 0.

            return
        ]
```

```
                                        # other models:
        time = (t-par(Ti)) / 60.        # convert to minutes
        if (time < 0.) [                # and handle t-ti < 0
            y = 0.
            if (tderiv)
                do i = 1, ninpar
                    if (tder(i))
                        dy(i) = 0.
            return
        ]

        if (infun == 1 | infun == 2) [          # exp, t*exp
            em1 = exp(-time*par(M1))
            em2 = exp(-time*par(M2))
            y = par(A1)*em1 + par(A2)*em2
            if (tderiv) [
                if (tder(A1)) dy(A1) = em1
                if (tder(A2)) dy(A2) = em2
                if (tder(M1)) dy(M1) = -time*par(A1)*em1
                if (tder(M2)) dy(M2) = -time*par(A2)*em2
                if (tder(Ti)) dy(Ti) = (par(M1)*par(A1)*em1 +_
                                        par(M2)*par(A2)*em2) / 60.
            ]

            if (infun == 2) [
                y = y * time
                if (tderiv) [
                    do i = 1, ninpar
                        if (tder(i))
                            dy(i) = dy(i) * time
                    if (tder(Ti))
                        dy(Ti) = dy(Ti)-(par(A1)*em1+par(A2)*em2)/60.
                ]
            ]
        ]
        else [                          # infun == 3: t exp -t*2
            em1 = exp(-par(M1)*time**2)
            em2 = exp(-par(M2)*time**2)
            y = par(A1)*time*em1 + par(A2)*time*em2
            if (tderiv) [
                if (tder(A1)) dy(A1) = time*em1
                if (tder(A2)) dy(A2) = time*em2
                if (tder(M1)) dy(M1) =-par(A1)*time**3*em1
                if (tder(M2)) dy(M2) =-par(A2)*time**3*em2
                if (tder(Ti)) dy(Ti) = _
                    ((2.*par(M1)*time**2-1.)*par(A1)*em1 + _
                     (2.*par(M2)*time**2-1.)*par(A2)*em2) / 60.
            ]

        ]
        return
end
```

```
# funup - compute uptake (residue) function (& maybe derivatives)

subroutine funup (tderiv, par, t, y, tder, dy)
logical tderiv, tder(ARB)
real par(ARB), t, y, dy(ARB)
#
#       uptake function for fitting.  Returns tissue activity level
#       at time t according to parameters 'par' (and implicitly, the input
#       function and its parameters).  If tderiv is true, for every true
#       tder(i) we return dy(i) = dy / dpar(i).
#
#       parameters:
#               par(1-4,7,8) = k1,k2,k3,k4  FDG rate constants
#               par(5)       = fv  fractional blood volume
#               par(6)       = t0  time shift between input and tissue blood
#
#       This routine uses actual blood measurements and times for
#       convolution when possible (infun==4).
#
        ext_func real fimpls, con
        real time(MAXDATA), f(MAXDATA,MAXFIT), b(MAXDATA), tlast
        integer k
        data tlast /1.0e+20/

        include parcom                                  # need to see infun
        include datcom                                  # need to see blood

        if (t < tlast) [                                # start new run-through
            k = 0
            call funin(.false., inpar, 0., b0)
        ]

        k = k + 1
        time(k) = t                                     # store current point
        tlast = t                                       # remember last seen

        call funin(.false., inpar, t-par(6), bhere)     # add one more input
        f(k, 1) = fimpls(par, t)                        # and impulse point

        if (infun == 4)
            y = con(f(1,1),blood,time,tblood,t-par(6),
                fimpls(par,0.),b0,0.,0.,k,nblood)
        else [
            call funin(.false.,inpar,t,b(k))
            y = con(f(1,1),b,       time,time,   t-par(6),
                fimpls(par,0.),b0,0.,0.,k,k)
        ]

        y = y + par(5)*bhere

        if (tderiv) [
            jder = 1
            do i = 1, nuppar
                if (tder(i)) [
                    jder = jder + 1
                    opar = par(i)
                    if (i == 6) [
                        h = abs(.01 * par(i)) + 1.0e-3
                        par(6) = par(6) + h
                        call funin(.false.,inpar,t-par(6),b1here)

                        if (infun == 4)
                            y1 = con(f(1,1),blood,      time,tblood,t-par(6),
                            fimpls(par,0.),b0,0.,0.,k,nblood)
                        else
```

```
                        y1 = con(f(1,1),b,time,time,t-par(6),
                        fimpls(par,0.),b0,0.,0.,k,k)

                    y1 = y1 + par(5)*b1here
                ]
                else [
                    h = abs(.005 * par(i)) + 1.0e-6
                    par(i) = par(i) + h
                    f(k, jder) = fimpls(par, t)

                    if (infun == 4)
                        y1 = con(f(1,jder),blood,time,tblood,t-par(6),
                        fimpls(par,0.),b0,0.,0.,k,nblood)
                    else
                        y1 = con(f(1,jder),b,     time,time,      t-par(6),
                        fimpls(par,0.),b0,0.,0.,k,k)

                    y1 = y1 + par(5)*bhere
                ]
                par(i) = opar
                dy(i) = (y1-y)/h
            ]
        ]
        return
end
```

```
# getbld - get time-activity-uncertainty data from .JOB format file

subroutine getbld (file,ndat,time,value,uncert,scale,true)
character file(ARB)
integer ndat
real time(ARB), value(ARB), uncert(ARB),scale
logical true

#       Reads activity vs. time from the (archived) blood file named 'file'
#       Sets number of values 'ndat' and fills arrays time (collection time
#       (in sec. after injection), value (counts/min/gm), uncert.
#
#       Function returns OK if successful, ERR if file was not found or
#       a read error was encountered.  The the filename can be of the form
#       name, archivename`filename, archivename`subarchive`filename, etc.
#
#       Scale is a scale factor to apply to the data and uncertainties.
#
        ext_func integer aopen, agtlin, ctoi
        ext_func real ctor
        integer fd, size(2)
        character line(ARGMAX)

        define (HEADERLINES,8)                          # blood file junk

        ndat = 0
?       call fprint(STDERR, "[GETBLD] file = %s@n", file)

        if (aopen(file, fd, size) == ERR)
            call cant(file)

        for (i = 1; i <= HEADERLINES; i = i + 1) [      # skip header
            if (agtlin(line, fd, size) == EOF)
                goto 100
?           call remark(line)
        ]

        while (agtlin(line, fd, size) != EOF) [
            j = 1
            i = ctoi(line, j)               # sample number
            t = ctor(line, j)               # time
            x = ctor(line, j)               # weight
            x = ctor(line, j)               # counts/min
            v = ctor(line, j)               # corrected counts/min/gm

            if (ndat >= 1)
                if (t < time(ndat))         # check for early junk at end
                    break

            ndat = ndat + 1
            time(ndat)   = t
            value(ndat) = v*scale
            uncert(ndat)= 1.                # a kludge for now
        ]

        true = .false.                      # ... uncertainties are bad
        call close(fd)
        return

100     call sprint(line, "Error - bad format in blood file %s@n", file)
        call putlin(line, STDOUT)
        call error(line)
end
```

```
# getcmd - read commands

subroutine getcmd

        ext_func integer prompt, gettok, equal, setvar, index
        character line(ARGMAX), var(MAXNAME)
        real ctor
        real val
        string prstr ": "
        include parcom

?       call fprint(STDERR, "[GETCMD]@n")

        while(prompt(prstr, line, STDIN) != EOF) [          # get instruction
            call fold(line)                                 # force lower case

            j = index(line, '#')                            # clip comments
            if (j > 1)
                line(j) = EOS

?           call fprint(STDERR,"command = '%s'@n", line)

            j = 1
            if (gettok(var, line, j) == EOF)
                next                                        # ignore empty lines

?           call fprint(STDERR,"var = '%s'@n", var)

            call skipbl(line, j)
            if (line(j) == '=') [                           # it is an assignment
                j = j + 1                                   # pick up value to assign
                val = ctor(line, j)

                if (setvar(var, val) == ERR)                # set it
                    call fprint(STDERR,"*Error - couldn't set %s@n", var)
            ]

            else if (equal(var, "write") == YES)
                call dowrit(line, j)

            else if (equal(var, "fit") == YES)
                call dofit(line, j)

            else if (equal(var, "debug") == YES) [
                idebug = STDERR
                while (gettok(var, line, j) != EOF)
                    if (equal(var, "verbose") == YES)
                        idebug = -STDERR
            ]

            else if (equal(var, "nodebug") == YES)
                idebug = 0

            else call fprint(STDERR,"*Error - illegal command: %s@n", var)
        ]

        return
end
```

```
# getdat - read data as directed by command line arguments

subroutine getdat

        ext_func integer getarg, getepi, getbld, ctoi
        ext_func real ctor
        real scale
        character arg(ARGMAX), file(MAXNAME)

        include parcom
        include datcom

        scale = 1.
        call strcpy("No file specified", file)

        for (i = 1; getarg(i, arg, ARGMAX) != EOF; i = i + 1) [
            call fold(arg)

            if (arg(1) == '-')
                select (arg(2)) [
                    case 's':
                        j = 3
                        scale = ctor(arg, j)
                        if (scale <= 0.)
                            call error("Bad scale factor")

                    case 'i':
                        j = 3
                        breg = ctoi(arg, j)
                        bscale = scale
                        call strcpy(file, bfile)
                        call getfun(bfile,breg,bscale,nblood,tblood,blood,
                                ublood,blabel,btrue)
                        scale = 1.

                        call rtoe(bscale,arg,1,4)
                        call fprint(STDOUT,
                        "# Input:  %s - %s (region %d) Scale =    %s  %d points@n",
                          bfile, blabel, breg, arg, nblood)

                    case 'u':
                        j = 3
                        treg = ctoi(arg, j)
                        tscale = scale
                        call strcpy(file, tfile)
                        call getfun(tfile,treg,tscale,ntissu,ttissu,tissu,
                                utissu,tlabel,ttrue)
                        scale = 1.

                        call rtoe(tscale,arg,1,4)
                        call fprint(STDOUT,
                        "# Tissue: %s - %s (region %d) Scale =    %s  %d points@n",
                          tfile, tlabel, treg, arg, ntissu)

                    default:
                        call error("Unknown flag")
                ]
            else
                call strcpy(arg, file)
        ]
        return
end
```

```
# getfun - read input or uptake function from blood file or epi file

subroutine getfun (file,region,scale,ndat,time,value,uncert,label,true)
character file(ARB), label(ARB)
integer region, ndat
real scale, time(ARB), value(ARB), uncert(ARB)
logical true                    # uncertainties true?

        ext_func integer index
        real t1(MAXDATA)                                # holds end times

                                                # if no ., append subfile
        if (index(file, '.') <= 0)              # to archive name
            if (region > 0)
                call concat(file, "`roi", file)
            else
                call concat(file, "`blood", file)

?       call fprint(STDERR,"[GETFUN] file = %s@n", file)

        if (region <= 0) [                              # read from blood file
            call getbld(file,ndat,time,value,uncert,scale,true)
            call strcpy("Blood draws", label)
        ]
        else                                            # read from roi file
            call getroi(file,region,ndat,time,value,uncert,label,scale,true)

        return
end
```

```
# getnam - get parameter name by type.  Inverse of whopar.

subroutine getnam (var, kind, index)
character var(ARB)
integer kind, index

        byte name(2)
        integer iname
        integer in, up, ns
        integer mqnam(NQPAR)
        equivalence (iname, name)

        include namcom
        include parcom
        common /mqnam/ mqnam, in, up, ns

%       iname = '??'

        if (index >= 1)
            select (kind) [
                case INPUTPAR:
                    if (index <= ninpar)
                        iname = innam(index)
                case UPTAKEPAR:
                    if (index <= nuppar)
                        iname = upnam(index)
                case MARQPAR:
                    if (index <= NQPAR)
                        iname = mqnam(index)
                case INPUTFUNCTION:
                    iname = in
                case UPTAKEFUNCTION:
                    iname = up
                case NSTEPS:
                    iname = ns
            ]

        var(1) = name(1)
        var(2) = name(2)
        var(3) = EOS

        return
end
```

```
# getroi - read time-activity-uncertainty data from .ROI format file

subroutine getroi (file,region,ndat,time,value,uncert,label,scale,true)
character file(ARB), label(ARB)
integer region, ndat
real time(ARB), value(ARB), uncert(ARB), scale
logical true

# Set ndat, time, value, and uncert, label, true.  If the file or the specified
# region does not exist we print an error message and exit.

        character line(134)
        filedes fdes
        real t0, t1
        ext_func integer datlin, aopen, pget
        integer size(2)

        call tinit                                  # initialize data table
        if (aopen(file, fdes, size) == ERR)         # attempt to open roi file
            call cant(file)

        if (datlin(line, fdes, size) == EOF)        # read up to first data line
            call error("No data in ROI file")       # there's nothing there?

        if (pget("NTIMES", 'd', ndat) != YES)       # get counts
            call error("NTIMES not defined")

        if (pget("NREGIONS", 'd', novl) != YES)
            call error("NREGIONS not defined")

        if (region < 1 | region > novl)
            call error("Region out of range")
                                                    # get times, average start&stop
        for (i = 1; i <= ndat; i = i + 1) [         # note that we have first line
            j = 1
            t0 = ctor(line, j)                      # start & stop times
            t1 = ctor(line, j)
            time(i) = (t0+t1)/2.                    # take middle as measurement time
            if (i < ndat)                           # get next line
                call datlin(line, fdes, size)
        ]

        for (i = 1; i < region; i = i + 1)          # skip other regions
            for (j = 1; j <= ndat; j = j + 1)
                if (datlin(line, fdes, size) == EOF)
1                   call error("Out of data in ROI file")

        for (i = 1; i <= ndat; i = i + 1) [         # get requested region
            if (datlin(line, fdes, size) == EOF)
                goto 1
            j = 1
            value(i) = ctor(line, j); uncert(i) = ctor(line,j )
            if (scale != 1.) [                      # apply scaling factor
                value(i)  = value(i)*scale
                uncert(i) = uncert(i)*scale
            ]
        ]

        if (pget("LABEL", 's', label) == NO)
            call strcpy("(No LABEL)", label)

        true = .false.                  # true only if TRUE_UNCERT == 1
        if (pget("TRUE_UNCERT", 'd', i) == YES)
            true = i == 1
            call close(fdes)
            return
    end
```

```
# gettok - extract alphanumeric or punc. token from string

integer function gettok (tok, str, j)
character tok(ARB), str(ARB)
integer j
#
#       extracts a token from str starting at j.  Skips blanks and
#       takes a string consisting of all alphanumeric or one punctuation
#       character.
#       Returns EOF when there no such tokens to find, OK otherwise.
#
        ext_func integer type                           # function returns LETTER
                                                        # or DIGIT or char.

        while (str(j) == ' ' | str(j) == '@t' | str(j) == '@n')
            j = j + 1
        if (str(j) == EOS) [                            # detect no token
?           call fprint(STDERR,"[GETTOK] EOF@n")
            return(EOF)
        ]

        iout = 2
        tok(1) = str(j)                                 # take first one anyhow
        j = j + 1

        if (type(tok(1)) != tok(1))                     # first is LETTER|DIGIT
            while (type(str(j)) != str(j)) [            # while rest are,
                tok(iout) = str(j)                      # copy them in
                iout = iout + 1
                j = j + 1
            ]

        tok(iout) = EOS
?       call fprint(STDERR,"[GETTOK] '%s'@n", tok)

        return(OK)
end
```

```
# init - initialize data variables, parameters, etc.

subroutine init
        integer now(9)
        character dat(10), tim(9)
        include datcom
        include parcom

        data inpar /1.0, 1.0, 1.0, .01,      0., 0./
        data uppar / .1, 0.1, 1.0, .001, .1, 0., 0., 0./

        infun = 1
        upfun = 1
        nsteps = 1000
        call finit                          # initialize function stuff

        call strcpy("No file specified", bfile)              # defaults
        call strcpy(bfile, tfile)
        blabel(1) = EOS
        tlabel(1) = EOS

        nblood = 0
        ntissu = 0
        breg   = 0
        treg   = 0
        tscale = 1.
        bscale = 1.

        call errset(72,.true.,.false.) #ignore     floating overflow
        call errset(73,.true.,.false.) #           zero divide
        call errset(74,.true.,.false.) #           underflow
        call errset(75,.true.,.false.) #           float to integer ofl.
        call errset(84,.true.,.false.) #           sqrt(<0)

        call getnow(now)
        call fmtdat(dat, tim, now, LETTER)

        call fprint(STDOUT,"# %s     %s %s@n", VERSION, dat, tim)

        return
end
```

```
C MARQ - Marquart Least-Squares fit

      SUBROUTINE MARQ (FUN,NPAR,PAR,DPAR,COV,NPARM,MAP,
     1                 NDAT,T,DAT,ERR,
     2                 NSTEP,CHI0,ISTEP,CHI,IERR,JERR,IDEBUG)

      EXTERNAL FUN
      INTEGER NPAR, NPARM, MAP(1), NDAT, NSTEP, ISTEP, IERR, JERR(1)
      REAL PAR(1), DPAR(1), COV(1), T(1), DAT(1), ERR(1), CHI0, CHI
C
C     Subroutine MARQ finds the set of parameters of function
C     FUN which minimizes chi-squared for the set of measurements
C     provided.  In the arugument descriptions below, [I] means
C     that the argument is an input (used by the subprogram), [O] means
C     that the argument is an output (set by the subprogram), [IO] means
C     that it is both used and set.
C
C     FUN     [I] - Function and derivative routine supplied by user
C                   SUBROUTINE FUN (TDERIV,PAR,TIME,Y,TDER,DY)
C                   LOGICAL TDERIV,TDER(1)
C                   REAL PAR(1),TIME,Y,DY(1)
C                     TDERIV  [I]- If .true., compute derivatives.  If
C                                  .false., do not return any derivatives in DY
C                     PAR()   [I]- Parameters of function
C                     TIME    [I]- Value of independent variable
C                     Y       [O]- Value of the function at TIME
C                     TDER()  [I]- Logical array: if TDER(i) then compute
C                                  DY(i) = dFUN/dPAR(i)
C                     DY()    [O]- Array of derivatives
C
C     NPAR    [I] - Length of parameter array PAR
C     PAR()   [IO]- Parameter array
C     DPAR()  [O] - uncertainties of fit parameters (0 if not fit)
C     COV()   [O] - Covariance matrix:
C                   COV((I-1)*NPARM + J) = cov(par(i),par(j))
C                   if par(i) and par(j) were fit, 0 otherwise
C
C     NPARM   [I] - Number of parameters in PAR to fit
C     MAP()   [I] - List (indices) of which parameters in PAR to fit
C
C     NDAT    [I] - Length of data array
C     T()     [I] - Values of the independent variable
C     DAT()   [I] - Data array
C     ERR()   [I] - Error array (uncertainties in DAT)
C
C     NSTEP   [I] - Maximum number of steps to take
C     CHI0    [O] - Initial chi-squared
C     ISTEP   [O] - Number of steps taken
C     CHI     [O] - Final chi-squared
C
C     IERR    [O] - Error flag:
C                     -1,     Error in parameter setup
C                      1,     Too many iterations
C                      0,     No errors detected
C                      2,     Failed to invert matrix while stepping
C                      3,     Failed to invert matrix after convergence
C                      4,     Fit interrupted by QUIT before convergence
C
C     JERR    [O] - Parameter error flags:
C                     -1,     Parameter not fit
C                      0,     Normal parameter fit
C                      1,     Parameter insensitive
C                      2,     Parameter correlated
C
```

```
C       IDEBUG  [I] - file descriptor for reporting debug information:
C                       <0      large amount of info on unit iabs(idebug)
C                        0      no debugging information
C                       >0      iterations reported on unit idebug
C
C       MARQ will exit prematurely with exit status 4 if the
C       logical flag QUIT in common /QUIT/ is set true.
C---
C
C       Variables internal to this routine:
C
C       A       - Second derivative matrix in various forms:
C                   originally calculated in upper triangle,
C                   normalized into lower triangle,
C                   brought to upper triangle and inverted in place.
C                 This is the matrix 'B' in the Marquardt algorithm.
C       D       - Step
C       G       - Gradient.  This is the vector 'E' in the Marquardt algorithm.
C       GS      - Normalized gradient
C       JFLAG   - Flags from SPDINV
C                   JFLAG = 0,      Normal
C                   JFLAG = 1,      Insensative parmeter
C                   JFLAG = 2,      Correlated parameter
C       PARM    - Mapped parameters
C       RTID    - Normalization factors (square-root of inverse
C                   of diagonals of A)
C       TAR     - Temporary parameters
C       TDER    - Logical array indicating which derivatives
C                   to return
C       DEBUG,VERBOS - logical debug printing flags
C       TEST    - used in debug printout; true if just had a bad step
C
        PARAMETER maxp = 10  ! max # of paramters (See also MQDER)
        PARAMETER mpsq = 100 ! and squared
C
        DIMENSION JFLAG(maxp),COM(3)
        DIMENSION PARM(maxp),G(maxp),GS(maxp),RTID(maxp),TAR(maxp)
        DIMENSION D(maxp),A(mpsq)
        LOGICAL TDER(maxp), debug, verbos, quit, test
        COMMON/MARQ/TCON,ECON,ZLAM,VLAM,COZ,VCONST,EPS
        COMMON/LST/LS(maxp)
        common/quit/quit
        common /mdebug/ debug, verbos, ldebug
        DATA EPS/1.E-6/

C       Convergence parameters

        DATA TCON,ECON/1.E-5,1.E-4/

C       Diagonal increment, factor to change it by,
C       limiting cosine of angle from the gradient,
C       factor to cut step size.

        DATA ZLAM,VLAM,COZ,VCONST/0.1,10.,0.8,0.5/

C ------------------------------------------------------------------
C                       SETUP
C ------------------------------------------------------------------

C       Check some input parameters.

        IERR = -1
        DEBUG  = IDEBUG .NE. 0          ! flags for switching debug info
        VERBOS = IDEBUG .LT. 0
        LDEBUG = IABS(IDEBUG)
```

```
      IF (NPAR.LT.1 .OR. NPAR.GT.MAXP .OR.                ! out of range
     1    NPARM.LT.1 .OR. NPARM.GT.NPAR) RETURN

      DO 12  I = 1,NPARM
          J = MAP(I)
          IF (J.LT.1 .OR. J.GT.NPAR) RETURN               ! out of range
          IF (I .GT. 1) THEN
              II = I - 1
              DO 10  JJ = 1,II
                  IF (J .EQ. MAP(JJ)) RETURN              ! duplicate
10            CONTINUE
          ENDIF
12    CONTINUE

      IERR = 0

C     Setup virtual row origins
C     (For a square matrix, because we will use both upper
C     and lower triangles)

      LS(1) = 0
      DO 14  I = 2,NPARM
14        LS(I) = LS(I-1) + NPARM

C     Setup derivative flags for variable parameters.

      DO 16  I = 1,NPAR
16        TDER(I) = .FALSE.
      DO 18  I = 1,NPARM
          J = MAP(I)
18        TDER(J) = .TRUE.
C
C     Map parameters and calculate initial chi-squared.
C
      CALL MQMAP (1,NPAR,PAR,NPARM,MAP,PARM)
      CALL MQCHI (FUN,NPAR,PAR.NPARM,MAP,PARM,
     1            NDAT,T,DAT,ERR,CHI)
      CHI0 = CHI

C.....DEBUG PRINTOUT

      IF (DEBUG) THEN
          CALL RTOE(CHI,A,11,3)                  ! use A as string scratch
          CALL FPRINT(LDEBUG,'Entering MARQ.  Chi = %s@n', A)
      ENDIF

C     Setup initial values for stepping.

      XLAM = ZLAM !starting value of diagonal increment
      ISTEP = 0   !initialize step number

C -------------------------------------------------------------------
C                     TOP OF STEPPING LOOP
C -------------------------------------------------------------------
C
C     Stay within step limit.

30    IF (ISTEP .GE. NSTEP) THEN
          IERR = 1
          RETURN
      ENDIF
      IF (QUIT) THEN
          IERR = 4
          GOTO 81
      ENDIF
```

```
          ISTEP = ISTEP + 1
          CONST = 1.                    !keep track of step cut factor

C         Get gradient (G) and second derivative matrix (A);
C         second derivatives go to upper triangle.

          CALL MQDER (FUN,NPAR,PAR,NPARM,MAP,PARM,
     1                NDAT,T,DAT,ERR,TDER,G,A)

C         Calculate normalization factors.

          DO 32  I = 1,NPARM
              LI = LS(I)
              RTID (I) = 0.
32            IF (A(I+LI) .GT. 0.) RTID(I) = 1./SQRT(A(I+LI))

C         Normalize gradient (G) and second derivative matrix (A);
C         normalized gradient goes to GS, and
C         normalized second derivatives go to lower triangle.

          DO 34  J = 1,NPARM
               LJ = LS(J)
               GS(J) = G(J)*RTID(J)

               DO 34  I = J,NPARM
                  LI = LS(I)
34                A(J+LI) = A(I+LJ)*RTID(I)*RTID(J)

C         Cut XLAM if not too small already.

          IF (XLAM .GT. EPS) XLAM = XLAM/VLAM

C ---------------------------------------------------------------------
C                         ADJUST DIAGONALS AND INVERT
C ---------------------------------------------------------------------
C
C         Put XLAM + 1. on the diagonal (same as adding XLAM; we've
C         normalized the diagonal to one, and bring the normalized matrix
C         to the upper triangle.

40        DO 42  J = 1,NPARM
              LJ = LS(J)
              IF (RTID(J) .GT. 0.) THEN
                  A(J+LJ) = XLAM + 1.
              ELSE
                  A(J+LJ) = XLAM
              ENDIF
C
              DO 42  I = J,NPARM
                  LI = LS(I)
42                A(I+LJ) = A(J+LI)

C         Invert the matrix.

          CALL SPDINV (A,NPARM,IFLAG,JFLAG)
          IF (IFLAG .NE. 0) THEN
              IERR = 2
              RETURN
          ENDIF

C         Matrix multiply and unnormalize to get new parameters.

          DO 50  I = 1,NPARM
              LI = LS(I)
```

```
                D(I) = 0.
                IF (JFLAG(I) .EQ. 0) THEN
                    DO 52  J = 1,I
                        LJ = LS(J)
52                      D(I) = D(I) + A(I+LJ)*GS(J)

                    IF (I .NE. NPARM) THEN
                        JJ = I + 1
                        DO 54  J = JJ,NPARM
54                          D(I) = D(I) + A(J+LI)*GS(J)
                    ENDIF
                ENDIF
                D(I) = D(I)*RTID(I)
                TAR(I) = PARM(I) + D(I)
50          CONTINUE

C ------------------------------------------------------------------------
C                           TEST STEP
C ------------------------------------------------------------------------
C
C           Test for a good step.

            CALL MQCHI (FUN,NPAR,PAR,NPARM,MAP,TAR,
     1                  NDAT,T,DAT,ERR,TCHI)
            IF (TCHI .LE. CHI) GO TO 70
            TEST = .TRUE.
            IF (VERBOS) GOTO 73             ! go do debug printout first

C           Not a good step; see if we're near the gradient.

51          COSINE = DOT(G,D,NPARM)/
     1        SQRT(DOT(G,G,NPARM)*DOT(D,D,NPARM))
            IF(COSINE .GT. COZ) GOTO 60

C           Increase XLAM and try again.

            XLAM = XLAM*VLAM
            IF (QUIT) THEN
                IERR = 4
                GOTO 81
            ENDIF
            GO TO 40

C           Bad step but right direction;
C           reduce step size until chi-squared is ok.

60          CONST = CONST*VCONST
            DO 62  I = 1,NPARM
                D(I) = D(I)*VCONST
62              TAR(I) = PARM(I) + D(I)
            CALL MQCHI (FUN,NPAR,PAR,NPARM,MAP,TAR,
     1                  NDAT,T,DAT,ERR,TCHI)
            IF (TCHI .LE. CHI) GO TO 70
            IF (QUIT) THEN
                IERR = 4
                GOTO 81
            ENDIF
            GOTO 60

C           Good step; update chi-squared and parameters.

70          TEST = .FALSE.
            CHI = TCHI
            DO 72  I = 1,NPARM
72              PARM(I) = TAR(I)
```

```
C
C.....      DEBUG PRINTOUT
C
            IF (DEBUG) THEN                          ! sorry, this is sooooo grody
73              CALL RTOE(CHI,COV,11,3)              ! use COV as string scratch
                CALL FPRINT(LDEBUG,'@nIteration %d  Chisqr = %s', ISTEP, COV)
                CALL RTOE(XLAM,COV,11,3)
                CALL FPRINT(LDEBUG,' Lam = %s', COV)
                CALL RTOE(CONST,COV,11,3)
                CALL FPRINT(LDEBUG,' Const = %s@n',COV)
                DO 94 I = 1, NPARM
                    CALL RTOE(PARM(I),COV,11,3)
                    CALL FPRINT(LDEBUG,' Par %2d  %s', I, COV)
                    CALL RTOE(D(I), COV, 11, 3)
                    CALL FPRINT(LDEBUG,' ; %s   %d@n', COV, JFLAG(I))
94              CONTINUE
                IF (TEST) GOTO 51    ! we were just testing this set of params
            ENDIF                    ! ... but you've seen worse so don't complain

C           Test for convergence; if not, go take another step.

            DO 74  I = 1,NPARM
                IF (ABS(D(I))/(TCON+ABS(PARM(I))) .GT. ECON) GO TO 30
74          CONTINUE

C ------------------------------------------------------------------------
C                           CONVERGENCE
C ------------------------------------------------------------------------
C
C           Put 1. on the diagonal, and
C           bring the normalized matrix to the upper triangle.

            IF (DEBUG) CALL FPRINT(LDEBUG,'@nConvergence!!!@n')
81          CONTINUE
            DO 80  J = 1,NPARM
                LJ = LS(J)
                IF (RTID(J) .GT. 0.) THEN
                    A(J+LJ) = 1.
                ELSE
                    A(J+LJ) = 0.
                ENDIF

                DO 80  I = J,NPARM
                    LI = LS(I)
80                  A(I+LJ) = A(J+LI)

C           Invert the matrix.

            CALL SPDINV (A,NPARM,IFLAG,JFLAG)
            IF (IFLAG .NE. 0) THEN
                IERR = 3
                RETURN
            ENDIF

C           Unnormalize the inverted matrix (gives covariance matrix).

            DO 84  J = 1,NPARM
                LJ = LS(J)
                DO 84  I = J,NPARM
84                  A(I+LJ) = A(I+LJ)*RTID(I)*RTID(J)

C           Extract uncertainties, and symetrize
C           covariance matrix to lower triangle.

            DO 86  J = 1,NPARM
```

```
            LJ = LS(J)
            D(J) = SQRT(A(J+LJ))

            DO 86  I = J,NPARM
                LI = LS(I)
86              A(J+LI) = A(I+LJ)

C       Unmap parameters, errors, and covariance matrix.

        CALL MQMAP (-1,NPAR,PAR,NPARM,MAP,PARM)
        CALL MQMAP (-101,NPAR,DPAR,NPARM,MAP,D)
        CALL MQMAP (-102,NPAR,COV,NPARM,MAP,A)

C       Set JERR equal to JFLAG when fit; set to -1 when not.

        DO 88  I = 1,NPAR
88          JERR(I) = -1
        DO 89  I = 1,NPARM
            J = MAP(I)
89          JERR(J) = JFLAG(I)

        RETURN
        END
```

```
C MQCHI - Compute Chi-squared by calling user's FUN.

      SUBROUTINE MQCHII (FUN,NPAR,PAR,NPARM,MAP,PARM,
     1                   NDAT,T,DAT,ERR,CHI)

      EXTERNAL FUN
      INTEGER NPAR, NPARM, MAP(1), NDAT
      REAL PAR(1), PARM(1), T(1), DAT(1), ERR(1), CHI

      BYTE VAL(40)
      LOGICAL TDERIV
      DATA TDERIV /.FALSE./
      DATA BIG /1.0E+20/              ! upper limit on chi square
      LOGICAL DEBUG, VERBOS, TEST
      COMMON /MDEBUG/ DEBUG, VERBOS, LDEBUG

C     Unmap parameters before calling FUN.

      CALL MQMAP(-1,NPAR,PAR,NPARM,MAP,PARM)

      IF (VERBOS) THEN
          CALL FPRINT(LDEBUG,'[CHI] Params:@n')
          DO 1 K = 1, NPARM
              CALL RTOE(PARM(K),VAL,11,3)
              CALL FPRINT(LDEBUG,'%d) %s  ', K, VAL)
1         CONTINUE
          CALL FPRINT(LDEBUG,'@n')
      ENDIF

C     Calculate the chi-squared.

      CHI= 0.
      DO 10  K = 1,NDAT
          CALL FUN (TDERIV,PAR,T(K),Y)
          CHI = CHI + ((DAT(K)-Y)/ERR(K))**2
          IF (CHI .GE. BIG) GOTO 11
10    CONTINUE
11    CONTINUE

      IF (VERBOS) THEN
          CALL RTOE(CHI, VAL, 11, 3)
          CALL FPRINT(LDEBUG, '>>> Chi = %s@n', VAL)
      ENDIF
      RETURN
      END
```

```
C MQDER - Compute Derivative Matrix from user's FUN.

      SUBROUTINE MQDER (FUN,NPAR,PAR,NPARM,MAP,PARM,
     1                  NDAT,T,DAT,ERR,TDER,G,A)

      EXTERNAL FUN
      LOGICAL TDER(1)
      INTEGER NPAR, NPARM, MAP(1), NDAT
      REAL PAR(1), PARM(1), T(1), DAT(1), ERR(1), G(1), A(1)

      PARAMETER maxp = 10                    ! max # of parameters

      REAL DY(maxp),DYM(maxp)
      COMMON/LST/LS(1)
      LOGICAL TDERIV, DEBUG, VERBOS
      COMMON /MDEBUG/ DEBUG, VERBOS, LDEBUG
      DATA TDERIV /.TRUE./

      DO 11  J = 1,NPARM                     ! clear gradient & deriv products
          G(J) = 0.
          LJ = LS(J)
          DO 10  I = J,NPARM
              A(I+LJ) = 0.
10        CONTINUE
11    CONTINUE

      CALL MQMAP(-1,NPAR,PAR,NPARM,MAP,PARM) ! Unmap the paramters before calling FN

C     Calculate the gradient (G) and the second derivative matrix (A).

      DO 23  K = 1,NDAT
          CALL FUN (TDERIV,PAR,T(K),Y,TDER,DY)
          ERSQI = 1./ERR(K)**2
          CALL MQMAP(1,NPAR,DY,NPARM,MAP,DYM)        ! map derivatives

          DO 20  I = 1,NPARM
              G(I) = G(I) + DYM(I)*(DAT(K) - Y)*ERSQI
20        CONTINUE

          DO 22  J = 1,NPARM
              LJ = LS(J)
              DO 21  I = J,NPARM
                  A(I+LJ) = A(I+LJ) + DYM(I)*DYM(J)*ERSQI
21            CONTINUE
22        CONTINUE

          IF (VERBOS) THEN
              CALL RTOF(T, VAL, 6, 1)
              CALL FPRINT(LDEBUG,'Der> @t=%s',VAL)
              CALL RTOE(Y, VAL, 11, 3)
              CALL FPRINT(LDEBUG,' y=%s@n', VAL)
              DO 1 J = 1, NPARM
                  CALL RTOE(DYM(J),VAL, 13, 3)
                  CALL FPRINT(LDEBUG,'d%d=%s ',J, VAL)
1             CONTINUE
              CALL FPRINT(LDEBUG,'@n')
          ENDIF
23    CONTINUE
      RETURN
      END
```

```
C MQMAP - Map/Unmap parameter array or matrix

      SUBROUTINE MQMAP (N,NPAR,X,NPARM,MAP,XM)

C     Map (or unmap if N<0) NPARM (of the NPAR) values of X
C     into XM according to MAP.
C     MAP contains the indices to the unmapped array.
C     N indicates the dimension of the array (2 or less),
C     and legal values are 1, 2, -1, -2, -101, -102.
C     Negative values of N indicate unmapping, and for
C     those less than -100, X is zeroed before the transfer.

      INTEGER N, NPAR, NPARM, MAP(1)
      REAL X(1), XM(1)

      LOGICAL ZERO

      NABS = IABS(N)
      ZERO = NABS .GT. 100
      IF (ZERO) NABS = NABS - 100
      IF (N .GT. 0) GO TO ( 10, 20) NABS
      IF (N .LT. 0) GO TO (110,120) NABS
      RETURN

10    DO 12  IM = 1,NPARM
          I = MAP(IM)
12        XM(IM) = X(I)
      RETURN

20    IJM = 0
      DO 22  JM = 1,NPARM
          J = NPAR*(MAP(JM)-1)
          DO 22  IM = 1,NPARM
              IJM = IJM + 1
              IJ = J + MAP(IM)
              XM(IJM) = X(IJ)
22    CONTINUE
      RETURN

110   IF (ZERO) THEN
          DO 112  I = 1,NPAR
112           X(I) = 0.
      ENDIF

114   DO 116  IM = 1,NPARM
          I = MAP(IM)
116       X(I) = XM(IM)
      RETURN

120   IF (ZERO) THEN
          NPSQ = NPAR**2
          DO 122  I = 1,NPSQ
122           X(I) = 0.
      ENDIF

124   IJM = 0
      DO 126  JM = 1,NPARM
          J = NPAR*(MAP(JM)-1)
          DO 126  IM = 1,NPARM
              IJM = IJM + 1
              IJ = J + MAP(IM)
              X(IJ) = XM(IJM)
126   CONTINUE
      RETURN
      END
```

```
# penter - enter definition into symbol table

subroutine penter (name, par)
character name(ARB), par(ARB)
#
#       enters defintion 'par' of parameter 'name'.

        pointer point
        integer info(T_SIZE)
        ext_func pointer sdupl
        ext_func integer lookup, enter
        ext_subr strcpy, dsfree, error
        include tablecom

        if (lookup(name, info, table) == YES) [ # there's an old definition
            call dsfree(info(T_POINTER))          # so free its string space
        ]

        point = sdupl(par)                        # enter defn into data storage
        if (point == LAMBDA)
            call error("Couldn't allocate string space for parameter")

        info(T_POINTER) = point                   # this is the stuff to store
        if (enter(name, info, table) == ERR)
            call error("Couldn't add definition to table")

?       call fprint(STDERR, "[PENTER] '%s' '%s'@n", name, par)
        return
end
```

```
# pflag - print strings for flags MARQ returns

subroutine pflag (fdes, flag)
integer fdes, flag
#
# prints a chararacter string on fdes
#
        select(flag) [
            case -1:
                call putlin("  (not fit)", fdes)
            case 0:
                ;                                   # nothing - fit ok
            case 1:
                call putlin("  insensitive", fdes)
            case 2:
                call putlin("  correlated", fdes)
            default:                                # unknown flag
                call putlin("  ??? flag ???", fdes)
        ]
        return
end
```

```
# pget - get ROI symbol definition in desired format

integer function pget(name, fmt, par)
character name(ARB), fmt
integer par(2)
#
# fetches parameter 'name' into 'par' interpreted in format 'fmt':
# 's' = string, 'd' = decimal integer, 'o' = octal integer, 'f' =
# floating point (single precision).
# If the parameter was defined, returns YES.  If not, doesn't alter par
# and returns NO.

        character cmem(1)
        ext_func integer lookup, ctoi
        ext_func real ctor
        ext_subr strcpy
        integer info(T_SIZE)
        real rpar
        integer ipar(2)
        equivalence (rpar, ipar(1))                 # union {real, integer(2)}
        include tablecom
        common /cdsmem/ cmem

?       call fprint(STDERR,"[PGET] '%s' '%c'", name, fmt)

        if (lookup(name, info, table) != YES) [ # well, that's that.
?           call fprint(STDERR, " *@n")
            return(NO)
        ]

        j = cvt_to_cptr(info(T_POINTER))            # convert integer array index to
                                                    # character array index

?       call fprint(STDERR, " = '%s'@n", cmem(j))

        select (fmt) [
            case 's':                               # s: copy string
                call strcpy(cmem(j), par)

            case 'd':                               # d: decode as integer
                par(1) = ctoi(cmem, j)

            case 'r':                               # r: decode as real
                rpar = ctor(cmem, j)
                par(1) = ipar(1)                    # we assume a real is same
                par(2) = ipar(2)                    # size as two integers

            case 'o':                               # o: decode as octal
                par(1) = 0
                while (cmem(j) >= '0' & cmem(j) <= '7') [
                    par(1) = par(1)*8 + cmem(j)-'0'
                    j = j + 1
                ]

            default:
                call error("PGET with undefined format")
        ]
        return(YES)
end
```

```
# rt3 - find roots of cubic polynomial with three real roots.

subroutine rt3 (p, q, r, alpha, beta, gamma, ierr)
real p, q, r, alpha, beta, gamma
integer ierr
#
#       Finds the 3 real roots of the cubic equation:
#
#       y**3 + P y**2 + Q y + R = 0 .
#
#       The roots are returned in alpha, beta, and gamma.
#
#       If three real unequal roots exist, ierr=0 is returned;
#       if three real roots exist, at least two of which are
#       equal, ierr=1 is returned;  otherwise ierr=-1 is returned,
#       and alpha, beta, and gamma are meaningless.
#
#---
        real = -(2.*p**3 - 9.*p*q + 27.*r) / 54.
        sqimag = -(4.*p**3*r - (p*q)**2 - 18.*p*q*r + 4.*q**3 + 27.*r**2)/108.

        if (sqimag < 0.) [
            ierr = -1
            return
        ]

        if (sqimag == 0.)
            ierr = 1
        else
            ierr = 0

        theta3 = atan2(sqrt(sqimag), real) / 3.
        abs3   = (real**2 + sqimag)**(1./6.)

        real3  =  abs3 * cos(theta3)
        aimag3 = abs3 * sin(theta3)
        sqrt3  = sqrt(3.)

        alpha  = 2.*real3 - p/3.
        beta   = - real3 - sqrt3*aimag3 - p/3.
        gamma  = - real3 + sqrt3*aimag3 - p/3.

        return
end
```

```
# setmap - determine which parameters to fit

subroutine setmap (line, map, nparm, which)
character line(ARB)
integer map(ARB), nparm, which
#
#       line() is a list of parameters to fit.      It is picked apart
#       into a list (map) of parameters to fit.

        ext_func integer type, gettok, equal
        character var(MAXNAME)
        integer set, i, ind
        logical dupe

        nparm = 0
        idebug = 0
        j = 1

        while (gettok(var, line, j) != EOF) [
            if (equal(var, ",") == YES)
                next

            call whopar(var, set, ind)
?           call fprint(STDERR,"* parameter set ind %s %d %d@n", var, set, ind)

            if (set == UNKNOWN) [
                call fprint(STDERR,"Unknown parameter name %s@n", var)
                nparm = 0
                return
            ]

            if (set != INPUTPAR & set != UPTAKEPAR) [
                call fprint(STDERR,"Can only fit input or uptake parameters@n")
                return
            ]

            if (nparm == 0)
                which = set
            else [
                if (set != which) [
                    call fprint(STDERR,"Can't mix in/up params in one fit@n")
                    nparm = 0
                    return
                ]

                dupe = .false.
                do i = 1, nparm
                    if (map(i) == ind) [
                        call fprint(STDERR,"%s duplicated in parameter list@n")
                        dupe = .true.
                    ]
            ]

            if (! dupe) [
                nparm = nparm + 1
                map(nparm) = ind
            ]

            call skipbl(line, j)
        ]

        return
end
```

```
# setvar - set parameter by name

integer function setvar (var, val)
character var(ARB)
real val

        real qpar(1)
        ext_func integer index
        integer kind, status

        include parcom
        common /marq/ qpar

        status = OK

        call whopar(var, kind, index)
?       call fprint(STDERR,"[SETVAR] %s = %d %d@n", var, kind, index)

        select (kind) [
            case INPUTPAR:
                inpar(index) = val
            case UPTAKEPAR:
                uppar(index) = val
            case MARQPAR:
                qpar(index)  = val
            case INPUTFUNCTION:
                i = int(val)
                if (i < 1 | i > minfun)
                    call fprint(STDERR,"*Bad input function number %d@n",i)
                else [
                    infun = i
                    call finit
                ]
            case UPTAKEFUNCTION:
                i = int(val)
                if (i < 1 | i > mupfun)
                    call fprint(STDRR,"*Bad uptake function number %d@n",i)
                else [
                    upfun = i
                    call finit
                ]
            case NSTEPS:
                nsteps = int(val)
            default:
                status = ERR
        ]

        return(status)
end
```

```
# shocov - print covariance matrix

subroutine shocov(cov, npar, which)
real cov(ARB)
integer npar, which
#
#       prints covariance matrix and correlations for the parameters.
#       we print covariances in the upper triangle and correlations
#       in the lower triangle.
#
#       version 2: only prints correlations (lower triangle).

        character var(MAXNAME)
        define(COV,cov(($1-1)*npar+$2))

        call fprint(STDOUT, "@nCorrelation Matrix:@n        ")

        do i = 1, npar [
            call getnam(var, which, i)
            call fprint(STDOUT, "      %s     ",var)
        ]
        call putch('@n', STDOUT)

        do i = 2, npar [
            call getnam(var, which, i)
            call fprint(STDOUT, "%s  ", var)

            for (j = 1; j < i; j = j + 1) [
                cor = COV(i,i)*COV(j,j)
                if (cor > 0.)
                    cor = COV(i,j)/sqrt(cor)
                else
                    cor = 0.
                call rtof(cor, var, 7, 3)
                call fprint(STDOUT, "  %s  ", var)
            ]
?           for (; j <= npar; j = j + 1) [                  # only print cov's when
?               call rtoe(COV(i,j), var, 11, 3)             # debugging
?               call putlin(var, STDOUT)
?           ]
            call putch('@n', STDOUT)
        ]
        return
end
```

```
# shopar - print parameters on file

subroutine shopar (fdes, dounc, uncert, doflag, flags)
integer fdes, dounc, doflag, flags(ARB)
real uncert(ARB)
#
#       shopar prints the input and uptake function nubmers and
#       parameters on 'fdes'.  If dounc is INPUTPAR or UPTAKEPAR
#       we print uncertainties next to the appropriate parameters.
#       Likewise, we print the flag labels next to
#       the parameters if doflag = INPUTPAR or UPTAKEPAR:
#       for param(i), we print
#               (not fit)                   if flags(i) = -1
#                                                         0
#               correlated                                1
#               insensitive                               2
#

        character name(3), val(MAXNAME)
        include parcom

        call fprint(fdes, "Input_function %d@n", infun)

        do i = 1, ninpar [
            call getnam(name, INPUTPAR, i)
            call rtoe(inpar(i), val, 11, 4)
            call fprint(fdes,"%s = %s", name, val)
            if (dounc == INPUTPAR) [
                call rtoe(uncert(i), val, 11, 4)
                call fprint(fdes, "  +-     %s", val)
            ]
            if (doflag == INPUTPAR)
                call pflag(fdes, flags(i))
            call putch('@n', fdes)
        ]

        call fprint(fdes, "@n@Uptake_function %d@n", upfun)

        do i = 1, nuppar [
            call getnam(name, UPTAKEPAR, i)
            call rtoe(uppar(i), val, 11, 4)
            call fprint(fdes,"%s = %s", name, val)
            if (dounc == UPTAKEPAR) [
                call rtoe(uncert(i), val, 11, 4)
                call fprint(fdes, "  +-     %s", val)
            ]
            if (doflag == UPTAKEPAR)
                call pflag(fdes, flags(i))
            call putch('@n', fdes)
        ]

        return
end
```

```
C SPDINV - Invert Symmetric PosDev Matrix.

      SUBROUTINE SPDINV(S,N,IFLAG,JFLAG)

C     INVERTS SYMMETRIC POSITIVE-DEFINITE MATRIX "S" IN PLACE
C     USING ONLY THE UPPER TRIANGLE.USER PROVIDES ARRAY "LS"
C     OF POINTERS TO THE VIRTUAL ROW ORIGINS OF "S".
C
C     S ( I + LS(J) ) IS THE (I,J) ELEMENT OF THE MATRIX
C                FOR 1 .LE. J .LE. N    ,  J .LE. I .LE. N

      DIMENSION S(1),JFLAG(1)
      COMMON/LST/LS(1)
      DOUBLE PRECISION SA,SB
      DATA EPS1,EPS2/1.E-35,1.E-6/

      DO 10  I = 1,N
          LI = LS(I)
          IF(S(LI+I) .LT. EPS1) GO TO 14

          IF(I .EQ. 1) GO TO 11
          TEMP = S(LI+I)
          KK = I - 1

          DO 12  K = 1,KK
              LK = LS(K)
12            S(LI+I) = S(LI+I) - S(LK+I)**2
          IF(S(LI+I) .LT. EPS2*TEMP) GO TO 15

11        JFLAG(I) = 0
          SA = S(LI+I)
          SB = DSQRT(SA)
          S(LI+I) = SB

          IF(I .EQ. N) GO TO 10
          JJ = I + 1

          DO 13  J = JJ,N
              IF(I .EQ. 1) GO TO 13
              DO 50  K = 1,KK
                  LK = LS(K)
50                S(LI+J) = S(LI+J) - S(LK+I)*S(LK+J)
13            S(LI+J) = S(LI+J)/S(LI+I)

          GO TO 10
14        JFLAG(I) = 1
          GO TO 16
15        JFLAG(I) = 2
          IF(S(LI+I) .LT. -EPS2*TEMP) GO TO 100
16        DO 18  J = 1,I
              LJ = LS(J)
18            S(LJ+I) = 0.
          DO 19  J = I,N
19            S(LI+J) = 0.
          S(LI+I) = 1.

10    CONTINUE

      DO 20  I = 1,N
          LI = LS(I)
          S(LI+I) = 1./S(LI+I)
          IF(I .EQ. N) GO TO 20
          JJ = I + 1

          DO 21  J = JJ,N
```

```
                LJ = LS(J)
                S(LI+J) = S(LI+J)*S(LI+I)
                IF(J .EQ. JJ) GO TO 21
                KK = J - 1
                DO 52  K = JJ,KK
                    LK = LS(K)
52                  S(LI+J) = S(LI+J) + S(LI+K)*S(LK+J)
21              S(LI+J) = -S(LI+J)/S(LJ+J)
20      CONTINUE

        DO 30  I = 1,N
            LI = LS(I)
            DO 30  J = I,N
                LJ = LS(J)
                S(LI+J) = S(LI+J)*S(LJ+J)
                IF(J .EQ. N) GO TO 30
                KK = J + 1
                DO 54  K = KK,N
54                  S(LI+J) = S(LI+J) + S(LI+K)*S(LJ+K)

30      CONTINUE

        IFLAG = 0
        RETURN

100     IFLAG = I
        RETURN
        END
```

```
# tinit - initialize ROI symbol table

subroutine tinit

        ext_func integer mktabl
        DS_DECL(mem,TABLESIZE)
        include tablecom

        call dsinit(TABLESIZE)
        table = mktabl(T_SIZE)
        if (table == LAMBDA)
            call error("Can't create parameter table")

?       call fprint(STDERR, "[TINIT] Size = %d@n", TABLESIZE)
        return
end
```

```
# whopar - find out what kind of parameter the named variable is

subroutine whopar (var, kind, index)
character var(ARB)
integer kind, index

        define(NQPAR,7)

        byte name(2)
        integer iname
        integer in, up, ns
        integer mqnam(NQPAR)
        equivalence (iname, name)

        include namcom
        include parcom
        common /mqnam/ mqnam, in, up, ns

%       data mqnam/ 'tc', 'ec', 'zl', 'vl', 'co', 'vc', 'ep'/
%       data in, up, ns /'in', 'up', 'ns'/

        name(1) = var(1)
        name(2) = var(2)

?       call fprint(STDERR, "[WHOPAR] '%c' '%c'@n", name(1), name(2))

        do i = 1, ninpar [
            if (iname == innam(i)) [
                kind = INPUTPAR
                index = i
                return
            ]
        ]
        do i = 1, nuppar [
            if (iname == upnam(i)) [
                kind = UPTAKEPAR
                index = i
                return
            ]
        ]
        do i = 1, NQPAR [
            if (iname == mqnam(i)) [
                kind = MARQPAR
                index = i
                return
            ]
        ]

        if (iname == in)
            kind = INPUTFUNCTION
        else if (iname == up)
            kind = UPTAKEFUNCTION
        else if (iname == ns)
            kind = NSTEPS
        else
            kind = UNKNOWN

        return
end
```

Appendix B. Format of ROI and Blood Data Files

B.1. Region of Interest (ROI) File Format

Data reduced from PET images consist of several activity-per-volume-element vs. time sets. This section describes a file format to represent these data with adequate internal documentation, allowing for easy extension of the types of included data.

A ROI file consists of the following parts:

1. header comments (2 or more lines)
2. sample times
3. activity values (1 or more sets)

Comment lines begin with **#**, and may appear anywhere in the file. Blank lines may appear anywhere; they are ignored.

B.1.1. Header comments

Comment lines with **%** as the first nonblank character after **#** are parameter definitions, and have the following format:

```
# %PARAMETER_NAME% parameter_value
```

The parameter name consists of one or more printing characters embedded between **%**'s. The parameter value (string representation) starts with the first nonblank character after the closing **%** and continues to the end of the line. Thus

```
# %LABEL% My Dog Has Fleas
# %ITEM%
# %DIGIT% 5
# this is a comment
```

defines three parameters, LABEL="My Dog Has Fleas", ITEM="" (empty string), and DIGIT="5".

Parameters may be redefined anywhere in the file. The definition of a parameter may thus depend on how far one has read into the file. The only parameters which may be sensibly redefined are LABEL and NPIXELS. Definition of NREGIONS and NTIMES is mandatory. The basic set of parameters for the ROI files are:

Name	Description	Example
BED	bed position in mm	450
BOI	time of injection	14:43:12
COMPOUND	compound injected	Palmitate
DATE	date of study	21-Aug-82
HGAP	ring half-gap in cm	1.
HLIFE	half life of isotope in seconds	1230.
ISOTOPE	labeling isotope in XXnn format	C11
LABEL	description of region	Left ventricle
NPIXELS	number of pixels in region	234
NREGIONS	number of regions of interest	2
NTIMES	number of time points	45
ORGAN	organ counted/imaged	Heart
OVDATE	date overlay file was created	22-Aug-82
OVLABEL	overlay file label	Spot Heart Overlays
OVTIME	time overlay file was created	10:12:01
PWID	pixel width in proj. bins	.2
SPECIES	subject species	Dog
STUDY	experiment title	PA #2, +drug, Sn spheres
SUBJECT	subject's name	Spot
XCENT	image horizontal offset	2
YCENT	image vertical offset	0

B.1.2. Sample Times

There are NTIMES samples represented in a ROI file. The sample time list gives the start and stop times in seconds of each sample counted. The times are in floating point format, one start/stop pair per line, separated by white space.

B.1.3. Activity Data

NREGIONS sets of NTIMES data lines each follow the sample times. These lines contain two numbers each: activity in counts/volume/sec at the corresponding sample time, and the uncertainty in the measurement. Before each set of activities there will be at least one comment line describing the data. The parameters LABEL, NPIXELS and UNITS would be useful to set as well.

B.1.4. Sample ROI file

```
# ROI file with 2 overlays of 4 files each
# %DATE%                  21-Aug-82
# %BOI%                   14:43:12
# %HLIFE%                 76.
# %ISOTOPE%               Rb82
# %COMPOUND%              Rb-82
# %SPECIES%               Dog
# %SUBJECT%               Spot
# %ORGAN%                 Heart
# %STUDY%                 Rb #3
# %BED%                   450
# %HGAP%                  1.
# %OVLABEL%               Spot Heart Overlays
# %OVTIME%                10:12:01
# %OVDATE%                22-Aug-82
# %PWID%                  .2
# %XCENT%                 2
# %YCENT%                 0
# %NTIMES%                5
# %NREGIONS%              2
#
# Times:
# start  stop
    0.    5.
    5.   10.
   10.   15.
   15.   20.
#
# Overlay 1
# %NPIXELS%               1033
# %LABEL%                 Left Ventricle
# %UNITS%                 Cts/pix/sec
1.4032E-03   5.4543E-02
3.0123E-02   6.3432E-02
4.4343E-01   8.3432E-02
1.3432E-02   7.2353E-03
#
# Overlay 2
# %NPIXELS%               154
# %LABEL%                 Myocardium
1.5432E-03   2.3423E-04
6.1938E-01   3.5234E-02
4.1945E+00   1.2345E-02
3.5343E+00   1.3433E-02
```

B.2. Blood File Format

The blood data format has a long, sad history. The only relevant parts for this program are:

1. Eight lines of text at the top, to be ignored.
2. Variable number of data lines after header, with five fields: sample number, draw time (seconds after injection), weight (gm), counts/min, counts/min/gm. The second and fifth fields are the data we use in *fit*.

There are often spurious entries at the end of the file with odd times; therefore, we read data lines until we find a time earlier than the one last read, or end-of-file.

B.3. Archiving Convention

The ROI and blood files are stored in Software Tools **ar** archive files, in a hierarchical scheme. The outer file is given the subject's last name, with extension ".a". This file is an archive of study archive files, given names such as "rb1", "fdg2", and "water2", which denote the several studies for a given subject. The study archives contain data files with standardized names:

roi	ROI data from PET analysis
blood	Blood draw data (if any)
comments	any useful information about the particular study

For example, if patient Wilson had two Rubidium-82 studies and one FDG study with blood draws, the file structure would be:

```
wilson.a
        `rb1
            `roi
            `comments
        `rb2
            `roi
            `comments
        `fdg
            `roi
            `blood
            `comments
```

There are several programs and command files to manipulate these archives; see appendix D.

Appendix C. Software Tools Library

The table below lists library routines used by the fitting program. The routines are from the Software Tools Portable Library, except those marked * (local additions to Ratfor Library) and † (RSX-11M Fortran Library).

integer function agtlin
get next line from an archive module

filedes function aopen
open archive module for reading

subroutine cant
print "Can't open" message and terminate execution

subroutine close
close (detach) a file

subroutine concat
concatenate 2 strings together

integer function ctoi
convert string at in(i) to integer, increment i

real function ctor
convert string at in(i) to real, increment i

subroutine dsfree
free a block of dynamic storage

subroutine dsinit
initialize dynamic storage space

integer function enter
place symbol in symbol table

integer function equal
compare str1 to str2; return YES if equal

subroutine error
print single-line message and terminate execution

subroutine errset†
control printing of error messages

subroutine fmtdat
convert date information to character string

subroutine fold
convert string to lower case

subroutine fprint*
formatted output conversion to file

integer function getarg
get command line arguments

subroutine getarg
get command line arguments

subroutine getnow
determine current date and time

integer function index

find character c in string str

integer function isatty
determine if file is an interactive device

integer function length
compute length of string

integer function lookup
retrieve information from a symbol table

integer function mktabl
make a symbol table

filedes function open
open an existing file

subroutine penter
place symbol in symbol table

integer function prompt
get next line from file, prompting if a terminal

subroutine putch
write character to file

subroutine putlin
output a line onto a given file

subroutine query
print command usage information on request

subroutine remark
print single-line message

subroutine rtoe*
subroutine rtof*
convert real to character string

pointer function sdupl
duplicate a string in dynamic storage

subroutine skipbl
skip blanks and tabs at str(i)

subroutine sprint*
formatted output conversion to string

subroutine strcpy
copy string at "from" to "to".

integer function type
determine type of character

NAME

Fit - fit compartment models to ROI data

SYNOPSIS

fit [file] [-sfactor] [-i[n]] [file] [-sfactor] [-u[n]]

DESCRIPTION

Fit reads region-of-interest data from ROI-format files and can fit compartmental models to them. It generates two forms of output: a commentary on fitting progress and results, and a table of input data and model values. This latter can be used to plot the results of fitting, and for simulation purposes.

COMMAND LINE ARGUMENTS specify the source and treatment of input and residue function data.

file

Specifies a file from which the next region(s) are to be read. The file may be changed between regions. The file may be a subfile in an archive; the file`subfile... format of acat(1) is accepted. If there is no period in the filename (that is, no extension), a subfile name (`roi or `blood) is appended to the name when the region number is specified.

-sfactor

Specifies a scale factor by which the next region data and uncertainties are multiplied. 'Factor' is a number in floating point or exponential notation. A scale factor is applied only to the next region read with the -i or -u flag.

-i[n]
-u[n]

These direct fit to read input (-i) or uptake (-u) data from the last specified file. If the region number n is omitted or zero, data are assumed to be in the format of the .JOB file produced from well counter data by CTSDON. If the region number is a positive integer, the data are assumed to be in ROI format. If current filename does not contain a period (.) a subfile is appended to the specified filename: `roi if there is a region number or `blood if the region number is missing or 0. Examples:

```
fit dog`fdg1 -i1 -u2
fit [15,1]human.job -s6.26e-6 -i [100,6]human.roi -u3
```

COMMANDS are read from the standard input and direct fit to set parameters, fit to models, and report the results. A commentary is produced on the standard output, describing the input data, commands, and results. This documentation may be collected by redirecting fit's output to a file.

If the standard input is a terminal, fit prompts with a colon (:). The commands are:

debug [verbose|off]

Controls fit's comments on the progress of fitting. If the command 'debug' is given, chi-square and the current parameters are reported to the standard error output at the end of each iteration. If the 'debug verbose' command is given, further information is printed. This mode is generally useful only for debugging fit. The 'debug off' command suppresses debug output.

name=value

Sets the parameter named 'name' to 'value' expressed in floating point or exponential notation. The parameters select the input and uptake models, their rate constants, and control the behavior of the Marquardt fitting algorithm. Names may be abbreviated to two letters. The names are:

infun

selects input function model:

```
   1   a1   exp(-m1 T)    + a2   exp(-m2 T)
   2   a1 T exp(-m1 T)    + a2 T exp(-m2 T)
   3   a1 T exp(-m1 T**2) + a2 T exp(-m2 T**2)
   4   linear interpolation of input data
```

where T = (t - ti).

a1, a2, m1, m2, ti

input function model parameters. Ti does NOT affect the linear interpolation input model.

upfun

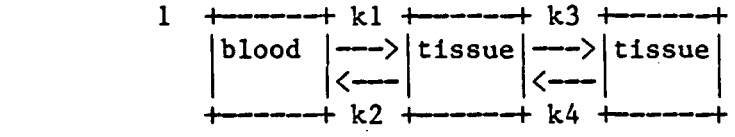
selects uptake function model. All are of form

```
   Up(t) = fv In(t') + (1-fv) In*Imp(t'),
      t' = t - t0
```

where Up = uptake model, In = input model, Imp = impulse response, and * denotes convolution.

The impulse responses are selected by upfun for the following models:

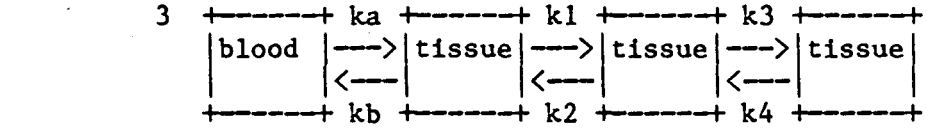
1



2 same model as 3 but parameters are those of its triexponential impulse response:

```
f1 exp(-k1 t) + f2 exp(-k2 t) + f3 exp(-k3 t)
```

3



tcon,econ

fit convergence parameters. When an iteration ends with

abs(step)/[abs(parameter) + tcon] <= econ for each parameter, the fitting algorithm terminates. Default values: 1.E-5, 1.E-4.

zlam, vlam, eps
Marquardt diagonal lambda control. Lambda is initially set to zlam. It is changed by multiplying or dividing by vlam but it not permitted to become smaller than eps. Default values: 0.1, 10, 1.E-6.

coz
limit of cosine of angle between gradient and Gauss-Newton vectors. When the angle exceeds arccos(coz), lambda is increased. Default: 0.8

vconst
factor by which stepsize is cut when gradient/Gauss angle is ok (cosine >= coz) but chi-square was not reduced. Default: .5

nsteps
maximum number of iterations allowed in fitting attempts. If the number of iterations exceeds nsteps, the fit is abandoned and a message is printed to the effect that a minimum was not found. This is not a fatal error.

fit list
specifies the names of parameters to fit, separated by commas. The list may contain all input or all uptake parameters. Input function parameters are varied to minimize the errors between the selected input function model and the input data read by the -i flag. Uptake function parameters are varied to minimize the errors between the selected uptake function model and the uptake data read by the -u flag, using the selected input function model and its current parameters.

During the fitting process, typing a <CONTROL-C> at the terminal keyboard will interrupt the fit at the current iteration. This works whether or not the standard input has been redirected. A note is printed on the output to the effect that fitting was interrupted before convergence.

The input and uptake function numbers and parameters are printed before and after fitting. The parameters have an estimated uncertainty next to them, and may include the comments:

(not fit)	not listed in the fit command
correlated	correlated to another parameter in the model
insensitive	has no effect on the model value.

The parameter uncertainty is computed with the assumption that the model is correct and that the uncertainties in the JOB or ROI file are off by a constant factor. We assume that chi-squared is equal to the number of degrees of freedom (number of data points minus

number of fitted parameters), and compute the uncertainties thereupon. The correlation matrix is printed after the parameters. For parameters bearing one of the comments above, the correlation is shown as 0.

write in|up [>|>> file]

Prints the input or uptake data and model values. The report goes to the standard output, or to a specified file. The >> version of file redirection means "append" rather than "write from scratch".

The first lines of the file describe the input data, the models selected and the input or uptake parameters. Subsequent lines are printed for each sample, listing the time, measurement, measurement uncertainty, and model value. The "write up" report also gives the value of the input function model at each time point.

In the examples below, the columns have been made a bit narrower to fit this page. The actual reports have the same layout.
Sample "write in" report:

```
-------------------------------------------------------
Input: dog.roi - BLOOD (reg. 1 *  1.000E+00)  Model 1

a1 = 1.82E+01  m1 = 1.07E+01  a2 = 2.26E+00  m2 = ...
ti = 7.00E+00

 time    input       uncert      in_model
    2.5  2.611E-01   3.046E-02   0.000E-01
    7.5  1.765E+01   2.335E-01   1.901E+01
    ...
-------------------------------------------------------
```

Sample "write up" report:

```
-------------------------------------------------------
Input:  man.epi - SAG SINUS (reg. 1 *  1.00)  Model 4
Uptake: man.epi - CORTEX    (reg. 2 *  1.00)  Model 1
k1 = 3.80E-03  k2 = 4.76E-02  k3 = 0.00E-01  k4 = ...
fv = 1.52E-01  t0 = 4.80E+00

 time    uptake      uncert      up_model    input
    2.5 -2.147E-03   2.147E-03   1.199E-03   7.877E-03
    7.5 -5.565E-03   5.565E-03   2.212E-05   0.000E-01
    ...
-------------------------------------------------------
```

IMPLEMENTATION

Fit is a Software Tools Ratfor program (with some Fortran-77 and Macro-11). The source is currently in [21,10]fit.tcs but may be moved to the ST binary directory ~bin. Fit.tcs maintains fit.w, which contains all necessary files:

Include files:
datcom, parcom, namcom, tablecom, fit.h

Sources (and routines):

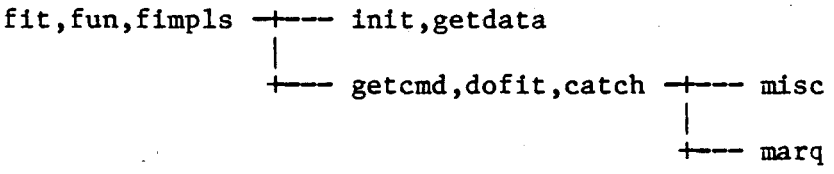
fit.r	main
fun.r	funin, funup, con, rt3
fimpls.r	fimpls, finit
init.r	init
getdata.r	getdat, getfun, getjob, getroi, datlin, tinit, penter, pget
getcmd.r	getcmd
dofit.r	dofit
misc.r	getnam, gettok, pflag, setmap, setvar, shocov, whopar, dowrit
marq.f	marq, mqchi, mqder, mqmap, spdinv, dot
catch.mac	catch

Build files:
makefit.cmd, fit.tkb, fit.odl

Documentation:
fit.fmt

The file fit.h contains a macro definition of a string 'VERSION' which should be updated to reflect the TCS revision level.

The program is overlayed as follows:



AUTHORS
Brian Knittel, Ron Huesman

NAME

makearch - make new patient archive

SYNOPSIS

makearch

DESCRIPTION

The data generated in Ring studies are stored in Software Tools archive files. The ST archive program combines many files into one, and provides the capability to insert, extract, list, and update consitituent files. Thus we can access the entire set of patient data with just one file name, but will retain the ability to play with the individual files.

Several study archives (e.g. fdg, rbl...) are combined in one patient archive. The study archives contain ROI, blood, and other study data. files. In particular, there is an optional 'comments' file which can contain text describing the experimental protocol and the regions of interest.

This program creates new patient archives - it is faster than modarch because it does not try to extract study archives before updating, and it does not update (or create) the patient archive until all the studies have been entered. The archive is given the patient name with extension '.a'.

The program asks for input in this order:

Patient name: enter a 1-9 letter name, or <return> to stop making archives.

Study name: enter a 1-9 letter name or <return> to stop entering studies into the patient archive. Studies should named something like xxxn where xxx is "rb" or "water" or "fdg" or some such, and n is the study number. For example, rb2 and water1.

For each study, you are asked for 6 files:

```
comments \
blood    |
roi      | enter name (including extension) or <return>
counts   |
weights  /
```

If one of the files you specify does not exist, you are returned to the Study Name question.

When all the files have been specified, you are given a chance reject the set of files and return to the Study Name without adding the study to the patient archive.

```
  \__back to Study name question
\__ back to Patient name question
```

FILES

patient.a	patient archive created
dr0:[100,1]makearch.cmd	command file

NAME

modarch - modify patient archives

DESCRIPTION

Modarch modifies patient data archives made with "makearch". The procedure is exactly the same, except that the patient archive must already exist. The questions asked and the procedure are the same.

Modarch attempts to extract named study archives from the patient archive. If they do not exist, they are created. Named study data files are inserted into the study archives, replacing any old files of the same type in the archive. Other files are left untouched.

For example, if a patient archive 'ROGER' was made with two studies composed as follows

```
roger
      `fdg1
           `roi              from ROGERFDG1.ROI
           `comments              ROGERFDG1.CMT
           `blood                 [15,1]ROGERFDG1.JOB
      `rb1
           `roi              from ROGERRB1.ROI
           `comments              ROGERRB1.CMT
```

and we told Modarch

```
                    ROGER
                    fdg1
      comments:     ROGERNEW.CMT
      counts:       [15,1]ROGERFDG1.CTS
```

then the new archive would be

```
roger
      `fdg1
          `roi              from ROGERFDG1.ROI
          `comments              ROGERNEW.CMT
          `blood                 [15,1]ROGERFDG1.JOB
          `counts                [15,1]ROGERFDG1.CTS
      `rb1
          `roi              from ROGERRB1.ROI
          `comments              ROGERRB1.CMT
```

FILES

patient.a	patient archive
dr0:[100,1]modarch.cmd	command file

NAME

plotfit - plot results of compartment model fits on line printer

SYNOPSIS

plotfit [file [inscale [timescale [spool [ymax [plotsize]]]]]]

DESCRIPTION

Plotfit reads a "write in" or "write up" file from FIT and plots the ROI data and the model values on the line printer. Plotfit is a command file in [100,1] and can be run in one of two ways:

from inside another command file, with

```
@dr0:[100,1]plotfit <args>
```

or from the terminal with

```
plotfit <args>
```

where <args> is an optional list of arguments separated by spaces:

file the name of the "write ..." file

inscale factor to scale input function by in uptake plots. This can be a number in floating point format or the letters 'Fv', which scales the input function by the fit vascular fraction. This shows how much of the tissue activity is due to blood. "-" suppresses plotting of the input function; this should be used when plotting input function fits.

timescale
'L' for log time x axis, 'T' for linear time axis, 'S' for sample number axis.

spool
"Y" - yes, spool plot immediately
"N" - no, make PLOTFIT.LST but dont spool yet.
"XXX" - use XXX instead of LPP
"XXX/YY" - use XXX and use /YY flag too. For example, answering "N" is the same as answering "LPP/-SP"

ymax maximum Y value for plot (Default - max data value)

plotsize
x, y size in inches for plot (Default "8,8.5")

If any of the first four arguments are not given, they are prompted for.

The fit rate constants are printed at the top of the plot.

FILES

plotfit.par	temporary parameter file for PLT.
plotfit.plt	temporary graphics file between PLT and LPP.
plotfit.lst	output listing (autodeleted if spool = "Y")

This report was done with support from the Department of Energy. Any conclusions or opinions expressed in this report represent solely those of the author(s) and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory or the Department of Energy.

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720