MATLAB: soluzione di equazioni differenziali e fitting non lineare

Focus principale sull'utilizzo delle funzioni ***ode45*** e ***lsqcurvefit***

Michele Scipioni
Ph.D. Student
Dip. Ingegneria dell'informazione
Università di Pisa

Corso di Immagini Biomediche, 02 Dicembre 2016

UNIVERSITÀ DI PISA

Outline

- *Introduzione*
- *Concetti matematici e farmacocinetici di base*
- *Modelli compartimentali*
- *Stima parametrica (fitting)*
- *Mappe parametriche*
- *Metodi di stima parametrica da sinogrammi (direct approach)*

PARTE 1

Soluzione di un sistema di equazioni differenziali in MATLAB

Obiettivo

$$\begin{cases} \dfrac{dC_1}{dt} = K_1 C_a(t) - k_2 C_1(t) - k_3 C_1(t) + k_4 C_2(t) \\ \dfrac{dC_2}{dt} = k_3 C_1(t) - k_4 C_2(t) \end{cases}$$

MODELLO BI-COMPARTIMENTALE DESCRITTO DA UN SISTEMA DI 2 EQUAZIONI DIFFERENZIALI

1. Prendere confidenza con le funzioni integrate in MATLAB dedicate alla soluzione delle ODE.
2. Imparare ad impostare le opzioni e I parametri richiesti dal motore di soluzione ODE
3. Imparare a passare il nostro modello ed eventuali parametri aggiuntivi a risolutore.

Risoluzione numerica di equazioni differenziali

MATLAB fornisce un gran numero di strumenti per risolvere numericamente delle equazioni differenziali. Noi ci concentreremo sui due metodi principali già integrati in tutte le version, ossia le funzioni ***ode23*** e ***ode45***, entrambe basate sul metodo di risoluzione **Runge-Kutta**.

Entrambe queste funzioni si presentano con la medesima interfaccia e si aspettano gli stessi parametri in ingresso.

[t, y] = ode45(odefun, tspan, y0)

- ***y***: vettore di soluzione, in cui ogni colonna è associata ad una variabile (più di una se lavoriamo con un sistema di ODE), ed ogni riga rappresenta un istante temporale riferito al vettore t.
- ***odefun***: funzione esterna (m-file) che restituisce un vettore colonna con l'uscita del sistema ODE
- ***tspan***: specifica il vettore dei tempi di integrazione per risolvere il Sistema
  - se ***tspan*** è un vettore di due elementi, questi sono trattati come tempi di inizio e fine, e gli step di integrazione intermedi sono scelti arbitrariamente dal risolutore
  - se ***tspan*** ha più di 2 valori, i risultati sono calcolati soltanti per gli specifici istanti temporali richiesti
- ***y0***: vettore delle condizioni iniziali per (t=0)

Esempio: equazione differenziale del primo ordine

$$\frac{dy}{dx} = xy^2 + y; \qquad y(0) = 1 \qquad x \in [0; 0.5]$$

1) Definire la funzione odefun: $f(x, y) = y'$

```
function yprime = firstode(x,y)
% FIRSTODE: Computes yprime = x*y^2+y

yprime = x*y^2 + y;
```

Esempio: equazione differenziale del primo ordine

$$\frac{dy}{dx} = xy^2 + y; \qquad y(0) = 1 \qquad x \in [0; 0.5]$$

2) Passare la funzione appena creata in input al risolutore *ode45()*

```
% Solution of First-Order Differential Equations
% Numerically approximate the solution of the first order differential equation
% dy/dx = xy2 + y;
% y(0) = 1,
% x ∈ [0, .5].

xspan = [0,.5];
y0 = 1;
[x,y]=ode45(@firstode,xspan,y0);

plot(x,y)
```

Esempio: equazione differenziale del primo ordine

$$\frac{dy}{dx} = xy^2 + y; \qquad y(0) = 1 \qquad x \in [0; 0.5]$$

3) Specificare un *tspan* personalizzato

```
tspan = 0:.1:.5;
y0 = 1;
[x,y]=ode45(@firstode,tspan,y0);

figure
plot(x,y)
title('tspan = 0:.1:.5')
```

Quello che è interessante puntualizzare riguardo il tspan è che MATLAB in background continua ad usare più o meno lo stesso timestep predefinito per risolvere l'equazione differenziale, variando soltanto il modo in cui restituisce in output il risultato finale. Lavorare con un tspan personalizzato quindi non altera in modo eccessivo l'accuratezza della soluzione numerica.

Esempio: equazione differenziale del primo ordine

$$\frac{dy}{dx} = xy^2 + y; \qquad y(0) = 1 \qquad x \in [0; 0.5]$$

4) Opzioni di personalizzazione

Ad ogni iterazione del risolutore viene calcolato un errore. Se chiamiamo y_k l'approssimazione di $y(x_k)$ allo step *k,* ed e_k l'erroe corrispondente, MATLAB adatta il suo passo di integrazione in modo che risulti:

$$\boldsymbol{e_k \leq max(RelTol * |y_k|, AbsTol)},$$

Dove I valori di default sono $RelTol = 10^{-3} = .001$ and $AbsTol = 10^{-6} = .000001$.

N.B.

Con questa convenzione, se la soluzione $|y_k|$ diventa molto grande, di conseguenza lo è anche l'errore, e per evitare che il risolutore si arresti prima di convergere è necessario ridurre *RelTol.* Viceversa, se la soluzione tende ad avere valori molto piccoli ($< 10^{-6}$), allora è *AbsTol* a dover essere ridotto.

Esempio: equazione differenziale del primo ordine

$$\frac{dy}{dx} = xy^2 + y; \qquad y(0) = 1 \qquad x \in [0; 0.5]$$

4) Opzioni di personalizzazione

$$\frac{dy}{dx} = xy^2 + y; \qquad y(x) = \frac{1}{1 - x}$$

```
format long
tspan=linspace(0,1-1e-6,5);
[x,y] = ode45(@firstode,tspan,1)
```

```
options = odeset('RelTol',1e-6)
format long
tspan=linspace(0,1-1e-6,5);
[x,y] = ode45(@firstode,tspan,1,options)
```

Esempio: sistemi di ODE

Modello Predator-Prey

$$\begin{cases} \dfrac{dy_1}{dt} = \alpha y_1 - \beta y_1 y_2 \\ \dfrac{dy_2}{dt} = \sigma y_2 + \rho y_1 y_2 \end{cases}$$

$\alpha = 1{,}2$ $\quad y_1(0) = 2$

$\beta = -0{,}6$ $\quad y_2(0) = 1$

$\sigma = -0{,}8$

$\rho = 0{,}3$

```
function yp = predprey(t, y)
% Modello predator-prey

alpha =  1.2;
beta  = -0.6;
sigma = -0.8;
rho   =  0.3;

dy1dt =  alpha*y(1) + beta*y(1)*y(2);
dy2dt =  sigma*y(2) + rho*y(1)*y(2);

yp = [dy1dt ; dy2dt];
```

Vettore colonna

```
tspan = [0 20];
y0 = [2, 1];

[t, y] = ode45(@predprey, tspan, y0)

figure, plot(t,y);

figure, plot(y(:,1),y(:,2));
```

Esempio: passaggio di parametri aggiuntivi

Immaginiamo di voler continuare a lavorare all'esempio del modello predatore-preda, ma **variando il valore delle costanti** del modello α, β, σ e ρ, ma ovviamente **senza modificare manualmente il file MATLAB**.

```
function yp = predprey1(t, y, p)
% Modello predator-prey

alpha =  p(1);
beta  =  p(2);
sigma =  p(3);
rho   =  p(4);

dy1dt =  alpha*y(1) + beta*y(1)*y(2);
dy2dt =  sigma*y(2) + rho*y(1)*y(2);

yp = [dy1dt ; dy2dt];
```

```
p = [1.2, -0.6, -0.8, 0.3];

[t, y] = ode45(@predprey1, tspan, y0, [], p)

figure, plot(t,y);

figure, plot(y(:,1),y(:,2));
```

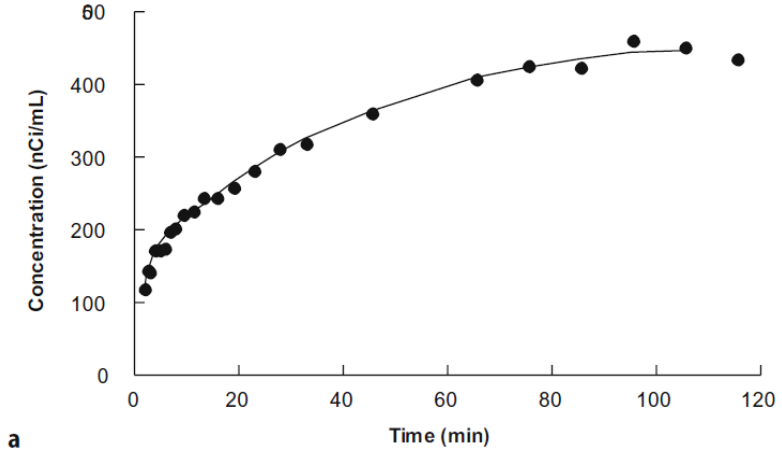
PARTE 2

Fitting di modelli non lineari in MATLAB

Obiettivo

STIMA AI MINIMI QUADRATI

$$\sum_{i=1}^{N}\left(C_i - C\left(T_i\right)\right)^2$$



a

1. Prendere confidenza con le funzioni integrate in MATLAB dedicate al fitting non lineare.
2. Imparare ad impostare le opzioni e i parametri richiesti dalla funzione *lsqcurvefit.*
3. Imparare a passare il nostro modello ed eventuali parametri aggiuntivi alla funzione di fitting.

Least Squares Estimator

L'obiettivo di uno stimatore ai minimi quadrati è trovare un vettore *x* che sia un minimizzatore locale della funzione data dalla somma degli scarti quadratic tra curva misurata e uscita del modello teorico, possibilmente sottoposta ad alcuni vincoli:

$$min_x \|F(x)\|_2^2 = min_x \sum_i F_i^2(x)$$

tale che $A \cdot x \leq b, Aeq \cdot x = beq, lb \leq x \leq ub.$

Solver	*F*(*x*)	Constraints
`mldivide`	*C*·*x* – *d*	None
`lsqnonneg`	*C*·*x* – *d*	*x* ≥ 0
`lsqlin`	*C*·*x* – *d*	Bound, linear
`lsqnonlin`	General *F*(*x*)	Bound
`lsqcurvefit`	*F*(*x*, *xdata*) – *ydata*	Bound

lsqcurvefit

[x,resnorm,residual,exitflag,output] = **lsqcurvefit**(fun,x0,xdata,ydata,lb,ub,options)

INPUT

- $\boldsymbol{x0}$: stima inziale dei coefficienti del modello da fittare
- $\boldsymbol{fun}(\boldsymbol{x}, \boldsymbol{xdata})$: funzione non lineare che fornisce in output la stima della curva da fittare
  - $\boldsymbol{x}$: vettore dei parametri incogniti
  - $\boldsymbol{xdata}$: variabile indipendete (generalmente il vettore temporale a cui è campionato ydata)
- $\boldsymbol{ydata}$: curva misurata (deve essere della stessa misura dell'output di fun)
- $\boldsymbol{lb}, \boldsymbol{ub}$: lower e upper bounds tali che $lb \leq x \leq ub$ (devono avere stessa dimensione di x0, o essere matrici vuote se non si è interessati a vincolare la stima parametrica)
- $\boldsymbol{options}$: opzioni di ottimizzazione

OUTPUT

- $\boldsymbol{x}$: parametri del modello stimati (stessa dimensione di x0 e del vettore di parametri preso in input da fun(x,xdata))
- $\boldsymbol{resnorm}$: somma degli scarti quadratic (residui) corrispondenti all'output del fitting
- $\boldsymbol{residual}$: vettore dei residui [fun(x,xdata)-ydata] alla soluzione x
- $\boldsymbol{exitflag}$: valore che descrive la condizione di uscita
- $\boldsymbol{output}$: struttura contenente informazioni riassuntive sul processo di ottimizzazione

Esempio: fitting di una funzione esponenziale

$$y = a_1 \exp(-b_1 t) + a_2 \exp(-b_2 t) \qquad t \in [0; 2]$$

1) Definire la funzione odefun: $y = fun(x, xdata)$

```
function y = non_linear_model(x,xdata)

a1 = x(1);
b1 = x(2);
a2 = x(3);
b2 = x(4);

y = a1*exp(-b1*xdata) + a2*exp(-b2*xdata);
```

Esempio: fitting di una funzione esponenziale

$$y = a_1 \exp(-b_1 t) + a_2 \exp(-b_2 t) \qquad t \in [0; 2]$$

2) Usiamo il modello implementato per simulare una misura rumorosa *ydata*

```
t = linspace(0,2,50);
p = [ 3   10.5    8    1.4];
y_true = non_linear_model(p,t);
ydata = y_true + 0.3*randn(size(y_true));

figure,
plot(t,y_true,'r-',t,ydata,'o')
legend('curva simulata','misura')
```

Esempio: fitting di una funzione esponenziale

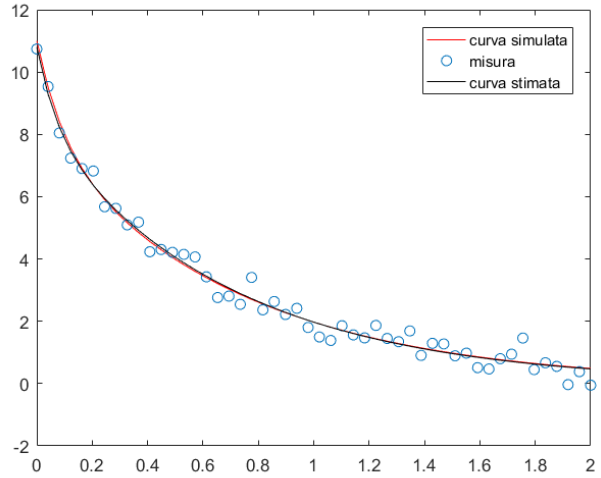
$$y = a_1 \exp(-b_1 t) + a_2 \exp(-b_2 t) \qquad t \in [0; 2]$$

3) Passare la funzione modello e il vettore misurato con input ad *lsqcurvefit*

```
x0 = [1 1 1 0];

% We run the solver and plot the resulting fit.
[p_est,resnorm,~,exitflag,output] = lsqcurvefit(@non_linear_model,x0,t,ydata)
y_fit = non_linear_model(p_est,t);

hold on
plot(t,y_fit,'-k')
legend('curva simulata','misura','curva stimata')
hold off
```



Esempio: fitting di una funzione esponenziale

$$y = a_1 \exp(-b_1 t) + a_2 \exp(-b_2 t) \qquad t \in [0; 2]$$

4) Passare parametri aggiuntivi alla funzione modello

```
function y = non_linear_model2(x,xdata,k)

a1 = x(1);
b1 = x(2);
a2 = x(3);
b2 = x(4);

y = a1*exp(-b1*xdata) + a2*exp(-b2*xdata) + k;
```

```
t = linspace(0,2,50); %Data(:,1);
p = [ 3   10.5    8    1.4];
k = 10;
y_true = non_linear_model2(p,t,k);
ydata = y_true + 0.3*randn(size(y_true));

figure,
plot(t,y_true,'r-',t,ydata,'o')
legend('curva simulata','misura')


x0 = [1 1 1 0];
fun = @(x,xdata)non_linear_model2(x,xdata,k);
[p_est,resnorm,~,exitflag,output] = lsqcurvefit(fun,x0,t,ydata)
y_fit = non_linear_model2(p_est,t,k);
```

Esempio: fitting di una funzione esponenziale

$$y = a_1 \exp(-b_1 t) + a_2 \exp(-b_2 t) \qquad t \in [0; 2]$$

5) Principali opzioni per personalizzare il risultato del fitting

```
x0                  = [1 1 1 0]
options             = optimset('Display', 'none');
options.TolFun      = 1e-6;
options.TolX        = 1e-6;
options.MaxFunEvals = 1e6;
options.MaxIter     = 1e6;
lb                  = [0.  5.  5.   0.];
ub                  = [5.  15.  10.   3.];
```

```
fun = @(x,xdata)non_linear_model2(x,xdata,k);
[p_est,resnorm,~,exitflag,output] = ...
                        lsqcurvefit(fun,x0,t,ydata,lb,ub,options)
y_fit = non_linear_model2(p_est,t,k);
```

PARTE 3

Esercitazione: implementazione e fitting di modelli bi-compartimentali applicati a dati PET

Punti principali

1. *Implementare modello bicompartimentale mediante ode45*
2. *Implementare modello bicompartimentale mediante la soluzione analitica bi-esponenziale*
3. *Verificare che i due modelli si comportano in modo analogo simulando delle curve usando i medesimi parametri cinetici*
4. *Estrarre una input function ed una curva tissutale dal dataset clinic fornito, e stimare i parametri cinetici usando entrambe le implementazioni del modello compartimentale*
5. *Usando l'implementazione più veloce delle due, applicare il fitting a tutti i voxel dell'imagine, generando delle mappe parametriche*

***FACOLTATIVO**: sfruttando l'implementazione dell'algoritmo ml-em sviluppato in una precedente esercitazione, cimentarsi nell'implementazione del metodo diretto di stima da sinogrammi, descritto nell'ultima parte della precedente lezione.*

1. Implementare modello bicompartimentale mediante ode45

$$\frac{dC_1}{dt} = K_1 C_a(t) - k_2 C_1(t) - k_3 C_1(t) + k_4 C_2(t)$$

$$\frac{dC_2}{dt} = k_3 C_1(t) - k_4 C_2(t)$$

$$C_1(t=0) = K_1$$
$$C_2(t=0) = 0$$

Consiglio: *implementare due funzioni distinte nello stesso m-file*

function Ct = modello_ode(k, t) ← funzione principale che calcola l'uscita della ode45 e da in output Ct=C(:,1)+C(:,2);

function Cp = sistema_ode(t,C,param) ← funzione interna in cui vengono esplicitate le due equazioni differenziali del sistema

2. Implementare modello bicompartimentale mediante la soluzione analitica bi-esponenziale

$$C_T = \alpha_1 e^{-\beta_1 t} + \alpha_2 e^{-\beta_2 t}$$

function Ct = modello_exp(k, time)

Questa funzione deve prendere gli **stessi ingressi della precedente** (le 4 costanti del modello compartimentale e il vettore dei tempi) e svolgere al suo interno la **conversione nel set di variabili ausiliarie** prima di calcolare C_T(t).

$$\alpha_1 = K_1 \frac{k_3 + k_4 - \beta_1}{\beta_2 - \beta_1}$$

$$\alpha_2 = K_1 \frac{\beta_2 - k_3 - k_4}{\beta_2 - \beta_1}$$

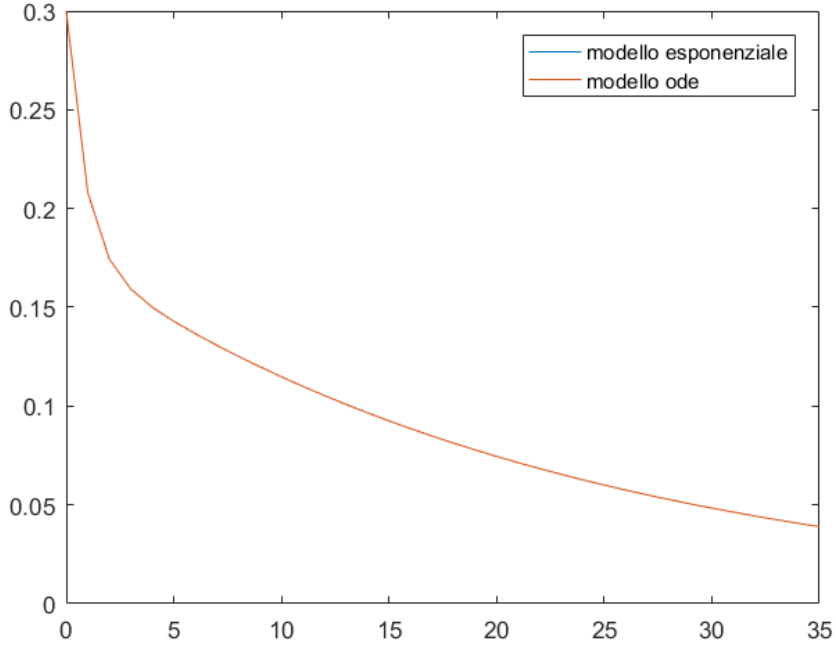
$$\beta_1 = \frac{(k_2 + k_3 + k_4) - \sqrt{(k_2 + k_3 + k_4)^2 - 4k_2k_4}}{2}$$

$$\beta_2 = \frac{(k_2 + k_3 + k_4) + \sqrt{(k_2 + k_3 + k_4)^2 - 4k_2k_4}}{2}$$

3. Verificare che i due modelli si comportano in modo analogo

Simulare le **risposte impulsive (IRF)** dei due tessuti con i seguenti valori di ingresso:

$$t = 0{:}35; \% \; in \; minuti$$
$$k = [0.3 \quad 0.5 \quad 0.6 \quad 0.1];$$



4. Stimare i parametri cinetici dai dati sperimentali (1/2)

1. **Importare il dataset allegato all'esercitazione**

2. **Estrarre 2 curve tempo-attività dalla matrice 4D "Volume"**
   - ROI Input function: slice 7, time frame 4
   - ROI tessuto: slice 23, time frame 24

3. **Importare dalla struttura "PETInfo" il vettore dei tempi:**
   - time = mean(PETInfo.time,2)./60;

4. **Implementare le formule riportate nella diapositiva 33 della lezione precedente utilizzando la *funzione riportata nella prossima slide.* Usare quest'ultima come ingresso all'algoritmo di ottimizzazione *lsqcurvefit***

5. **Impostare le seguenti opzioni di ottimizzazione:**

   - lb = [0. 0. 0. 0. 0.]; % K1 k2 k3 k4 vB
   - ub= [1. 1. 1. 1. 1.]; % K1 k2 k3 k4 vB
   - k0= [0.1 0.1 0.01 0.01 0.1]; % K1 k2 k3 k4 vB

4. Stimare i parametri cinetici dai dati sperimentali (2/2)

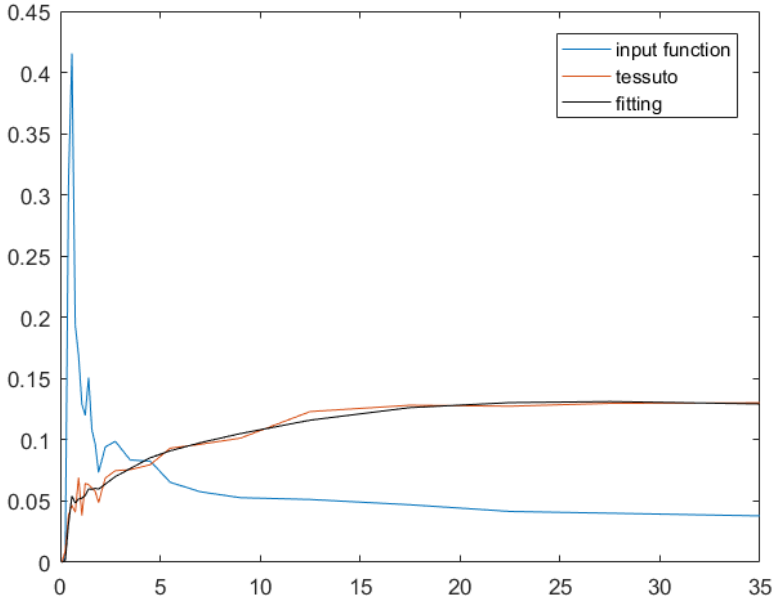
```
function Ct = modello_conv_IF(k, time, Cp)

vB = k(5);

dt = 0.1;
t  = 0:dt:time(end);
IF = max(0,interp1(time,Cp,t,'linear',0) );

sol = conv(modello_exp(k, t), IF)*dt;
tsol = (0:dt:2*max(t));
% Taking into account natural radioactive decay
dk = log(2)/109.8;  % radioactive decay constant for F-18
sol = sol .* exp(-dk * tsol);

% Taking into account blood fraction in tissue
sol = max(0,interp1(tsol,sol,time,'linear',0) );
Ct = (1-vB)*sol + vB* Cp;
Ct(Ct<=0) = eps;
```

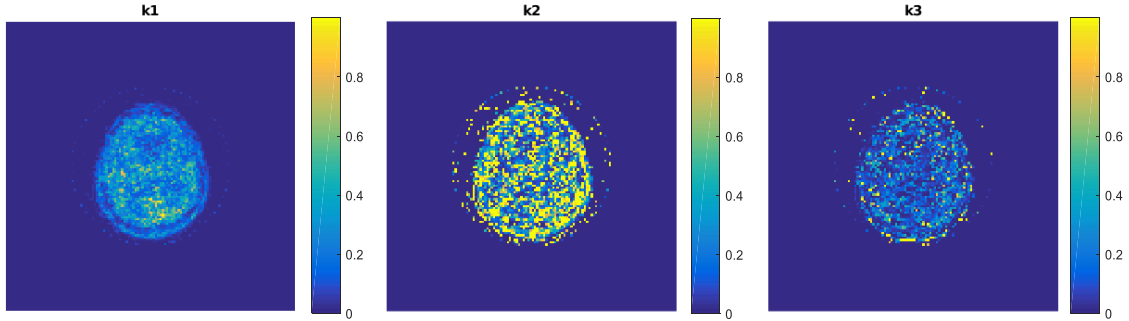


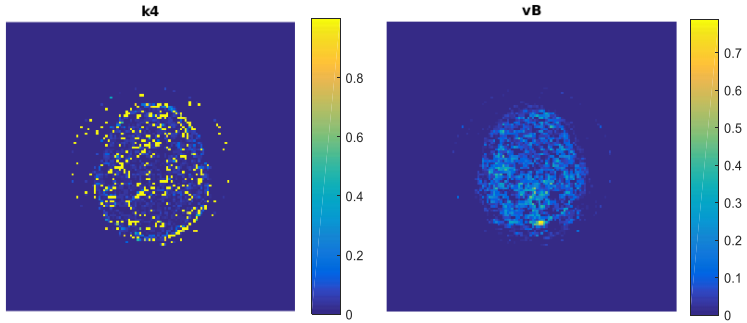
K1	k2	k3	k4	vB
0.2820	0.5087	0.6218	0.0804	0.0656

5. Generazione di mappe parametriche

1. **Importare il dataset allegato all'esercitazione**
2. **Estrarre 1 curva tempo-attività dalla matrice 4D "Volume" relativa alla input function** (ROI da slice 7, time frame 4)
3. **Importare dalla struttura "PETInfo" il vettore dei tempi:**
   - time = mean(PETInfo.time,2)./60;
4. **Scegliere una fetta del volume PET importato (ad esempio la 24 da cui si è estratta la ROI precedente o qualsiasi altra a piacere) e fittare voxel per voxel il modello, come fatto al punto precedente.**
   - img = squeeze(Volume(:,:,24,:));

5. Generazione di mappe parametriche





Consiglio

Impostare un "filtro" che riduca il numero di pixel da fittare, per accelerare l'esecuzione del codice ed anche per ottenere delle mappe più pulite (evitando di fittare anche pixel di sfondo). Le mappe qui riportate sono ottenute:

- Moltiplicando il volume importato x1000 (***img = img.*1000***)
- Fittando solo le curve estratte da img tali che (***max(curva)>=15***)

6. Stima diretta dei parametri da sinogramma (FACOLTATIVO)

1. **Importare il dataset allegato all'esercitazione**
2. **Estrarre 1 curve tempo-attività dalla matrice 4D "Volume" relativa all input function** (slice 7, time frame 4)
3. **Importare dalla struttura "PETInfo" il vettore dei tempi:**
   - time = mean(PETInfo.time,2)./60;
4. **Scegliere una fetta del volume PET importato (ad esempio la 24 da cui si è estratta la ROI precedente o qualsiasi altra a piacere) e fittare voxel per voxel il modello, come fatto al punto precedente.**
5. **Richiamare l'esercitazione sull'algoritmo di ricostruzione MLEM (28 Ottobre 2016) e:**
   - utilizzare la matrice di sistema (A) per convertire il volume PET in un sinogramma dinamico (come fatto con il fantoccio l'altra volta)
   - "copiare" il blocco di codice che esegue 1 iterazione di MLEM, facendo l'ipotesi che il sinogramma non abbia artefatti da correggere
6. **Implementare il metodo diretto di stima parametrica, alternando una iterazione MLEM ad un fitting dell'intero volume ricostruito, come svolto al punto 5.**

6. Stima diretta dei parametri da sinogramma (FACOLTATIVO)