Fitting spatial ICAPP models

It is recommended that you go over the tutorial for temporal models first, since the mechanics of model fitting are the same. We will only highlight the differences here.

The data

As with the temporal programs, the data should be in an $n \times 1$ cell array x, where *n* is the number of replications. But now each x{i}, instead of a vector, must be an $m_i \times 2$ matrix where the rows are the spatial coordinates of the observed events for the *i*th replication.

Fitting the model

The parameter estimators are computed, basically, by calling

```
>> [param,u,y,logf,OF,F] = …
                    icapp2d(x,rng,Rmax,knts,p,sm,itmax,start,...)
```

The main difference with the temporal `icapp` program is in the variables `rng` and `Rmax`, that specify the range, and `knts`, that specifies the basis knots.

Suppose the region *B* of interest is the triangle with vertices (0,0), (0,1) and (2,0). Then one would write the following indicator function:

```
function y = rng_tri(x)
      y = x(:,1)>=0 & x(:,1)<=2 & x(:,2)>=0 & x(:,2)<=(1-0.5*x(:,1));
end
```

This function takes as input an $m \times 2$ matrix and gives as output a logical $m \times 1$ vector indicating whether each row of the matrix is inside the triangle. If we save this function in the file `rng_tri.m`, say, then we call `icapp2d` as follows:

```
>> [param,u,y,logf,OF,F] = …
                    icapp2d(x,@rng_tri,Rmax,knts,p,sm,itmax,start,...)
```

The variable `Rmax` is a struct that specifies a rectangle that contains *B*, which is necessary for numerical integrations within the program. Naturally, although not strictly necessary, one would specify the smallest possible rectangle. In the case of the triangle with vertices (0,0), (0,1) and (2,0), this would be the rectangle $[0,2] \times [0,1]$:

```
>> Rmax = struct('I1',[0 2],'I2',[0 1])
```

The variable `knts` is a $q \times 2$ matrix giving the knots of the basis functions β_k of which the independent components φ_j are linear combinations. The β_ks are normalized Gaussian densities centered at the knots (the normalization ensures that $\sum_{k=1}^{q} \beta_k(t) = 1$ for all t). It is not strictly necessary that all knots be inside *B*, but for numerical stability it is better that they are. The easiest way to do this is to create

equally spaced knots in a rectangle and filter out the unwanted knots with the function `rng`. For example,

```
>> t1 = linspace(0,2,10)';
>> t2 = linspace(0,1,10)';
>> knts0 = [kron(t1,ones(10,1)) kron(ones(10,1),t2)];
>> i = rng_tri(knts0);
>> knts = knts0(i,:);
```

Plotting

The functions `icapp2d` and `NGRbasis` were written so as to make their output amenable for computations, but not for plotting. If, say, we want to evaluate the 2nd model component φ_2 at points $t_1, \ldots, t_m$, we collect these points in an $m \times 2$ matrix t and simply do

```
>> B = NGRbasis(t,knts,0);
>> phi2 = B*param.C(:,2);
```

as we would for the temporal components. Here `phi2` is an $m \times 1$ vector and its jth row is $\varphi_2(t_j)$. But in order to plot φ_2 using the standard Matlab functions `surf`, `mesh` or `contour`, we need `phi2` in matrix form.

This can be done as follows:

```
>> t1 = linspace(0,2,20)';
>> t2 = linspace(0,1,20)';
>> t = [kron(t1,ones(20,1)) kron(ones(20,1),t2)];
>> B = NGRbasis(t,knts,0);
>> phi2 = B*param.C(:,2);
>> Phi2mat = reshape(phi2,20,20);
```

Note that, given the way we constructed `t`, the rows of `Phi2mat` correspond to the grid `t1` and the columns to `t2`. So we plot φ_2 with, for instance, the command

```
>> mesh(t1,t2,Phi2mat)
```

But this plots φ_2 over a rectangular region, not the triangular region we are interested in. So we mask out the extra points by, for example, replacing them with NaN's before plotting:

```
>> i = rng_tri(t);
>> phi2(~i,:) = NaN;
>> Phi2mat = reshape(phi2,20,20);
>> mesh(t1,t2,Phi2mat)
```

The functions `surf` and `contour` are used in a similar way.