Beowulf cluster:
ordinary PCs connected by network used for parallel computing

Beostar (16 X 2 CPUs/nodes):
cluster at Laboratory for Computational Stochastics

Exploring the Strauss point process model

Linda Stougaard Nielsen
Laboratory for Computational Stochastics
University of Aarhus

Peter Møller-Nielsen, Michael K. Sørensen, Eva Vedel Jensen

Programs: `www.imf.au.dk/ ~ lins/uni/sc`

Technical Report 2, Laboratory for Computational Stochastics

- Point processes
- Strauss point process
  - Monte Carlo techniques (OO programming)
  - Maximum likelihood estimation
- Project objective and parallel programming
  - Sequential program (and faster algorithm)
  - Embarrassing/perfect parallel
  - The program
- Results: Exploring the Strauss point process model
  - Simulation experiment for MLE

Notation

- $\mathcal{X} \subseteq \mathbb{R}^k$ full-dimensional bounded subset
- point $\eta \in \mathcal{X}$ realization of stochastic variable Y in $\mathcal{X}$
- point pattern $x = \{\eta_1, \ldots, \eta_n\}$ where $\eta_i \in \mathcal{X}$ realization of point process X in $\mathcal{X}$
- State space for X: $\Omega_{\mathcal{X}}$ set of all point patterns

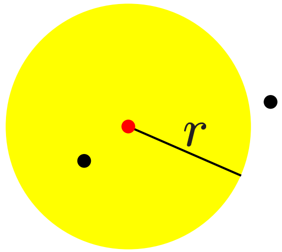
Homogeneous Strauss point process

Density:

$$f(x) = c(\beta, \gamma, r)^{-1} \beta^{n(x)} \gamma^{s_r(x)}, \quad x \in \Omega_{\mathcal{X}}$$

$n(x)$: number of points

$s_r(x)$: number of r-close neighbours



- γ close to 0: No neighbours (Hard-core)
- $\gamma = 1$: Complete spatial randomness (Poisson model)

Background: Working with point processes

Density

$$f_\psi(x) = \frac{h_\psi(x)}{c(\psi)}, \qquad c(\psi) = \int_{\Omega_{\mathcal{X}}} h_\psi(x)\Pi(dx)$$

cannot compute integral explicitly

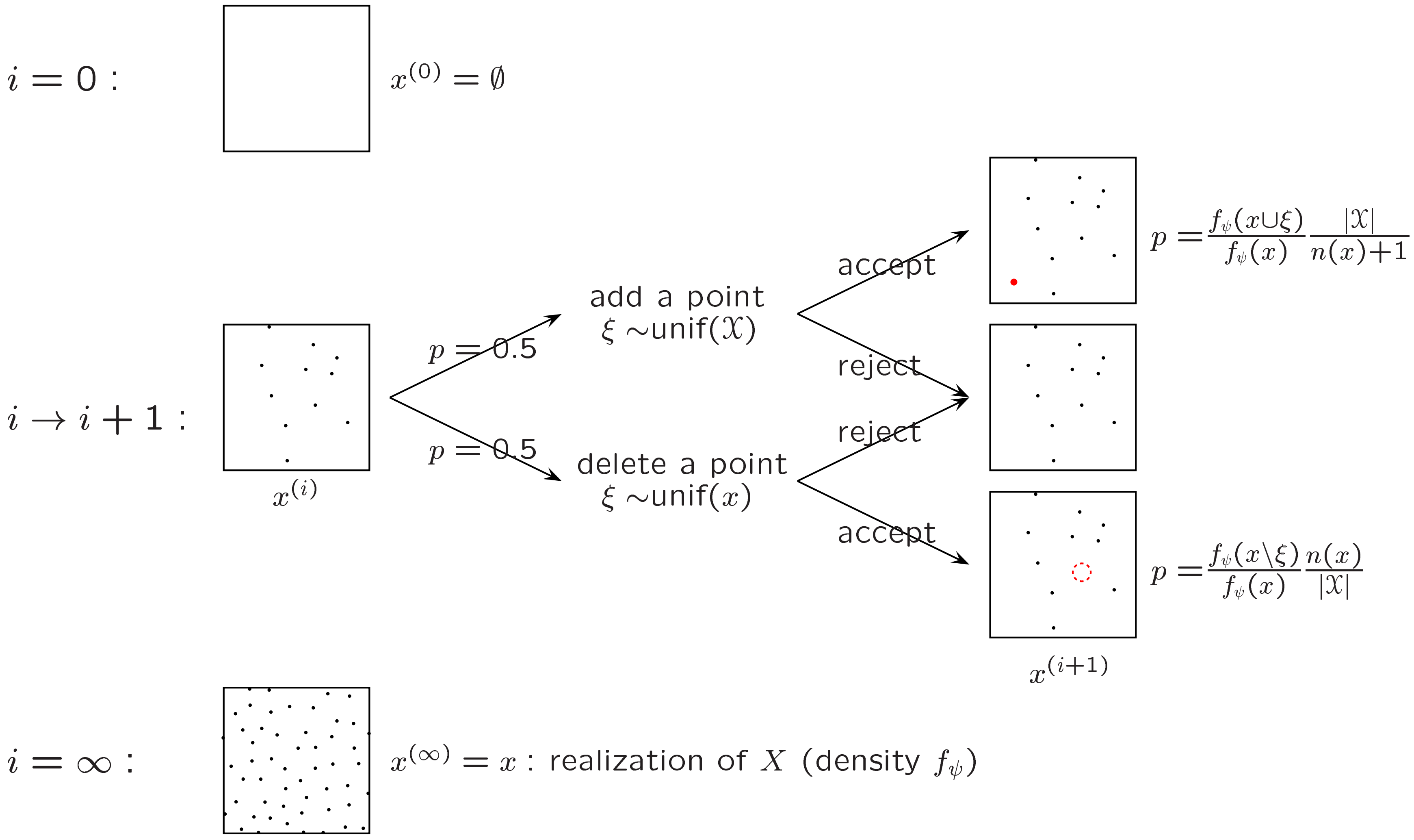
Solution: Monte Carlo techniques

Simulation based on density ratios

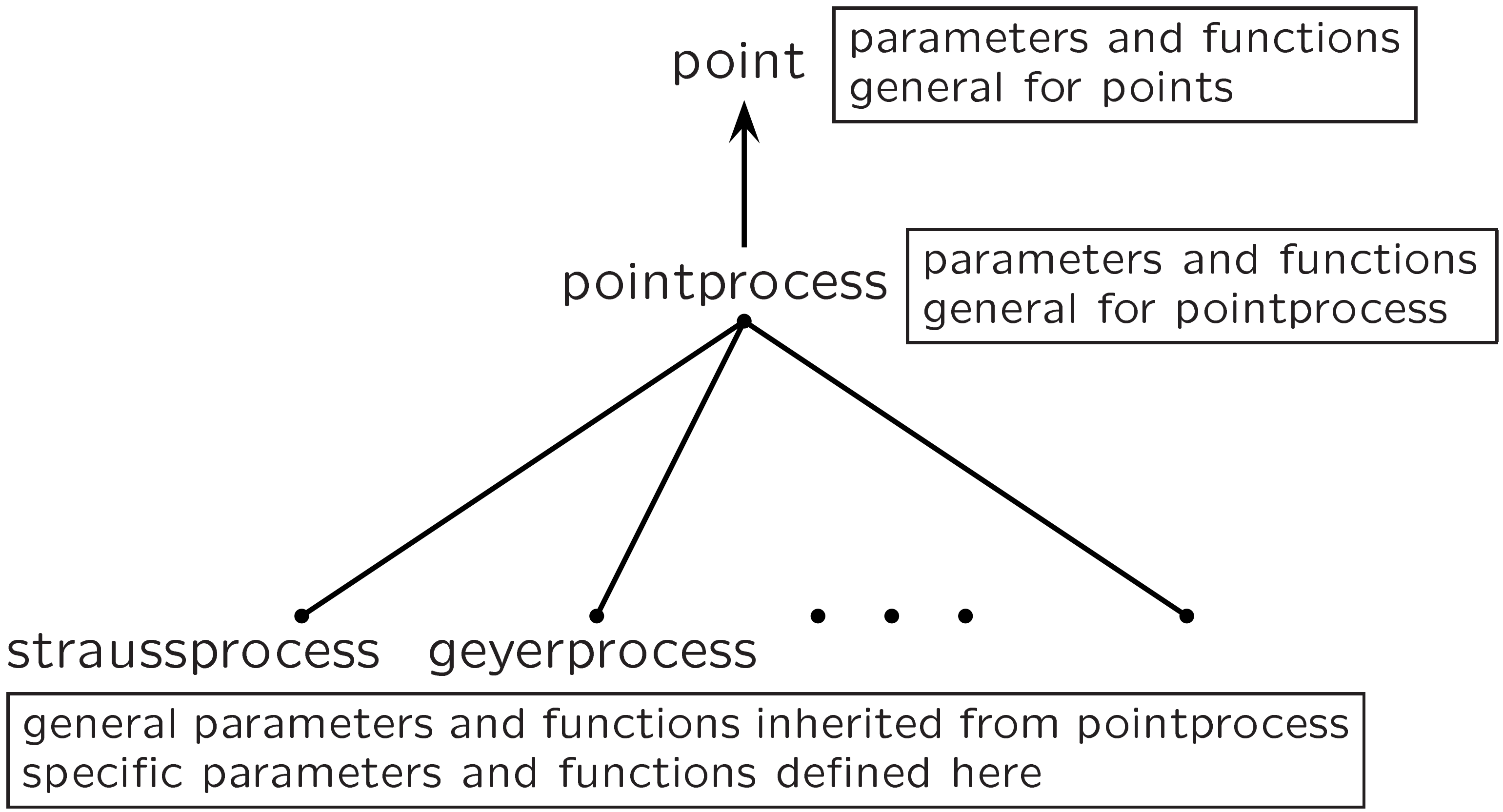
(1) $$\frac{f_\psi(x)}{f_\psi(y)} = \frac{h_\psi(x)}{h_\psi(y)}$$

(2) $$\frac{f_\psi(x)}{f_\theta(x)} = \frac{c(\theta)}{c(\psi)}\frac{h_\psi(x)}{h_\theta(x)} = \mathbb{E}_\psi\left(\frac{h_\theta(X)}{h_\psi(X)}\right)\frac{h_\psi(x)}{h_\theta(x)}$$

Draw x: Metropolis-Hasting birth-death algorithm



C++ library for point processes



point	pointprocess	straussprocess	geyerprocess
k	k, n, r, window, boolean n fixed	β, γ	β, γ, c
	read/save		
	add point		
	delete point		
replace	replace		
transform	transform		
	emptyprocess		
	binoprocess		
unifpoint	poissonprocess		
	makeMHBDstep	acceptprob	acceptprob
	makeprocess		
distance	countNeighbours		

Density ratios type (1)

- Simulating a realization/sample of X
- Approximation of means:

$$\mathbb{E}_\psi g(X) = \frac{1}{m}\sum_{i=1}^{m} g(x_i),$$

$x_1, \ldots, x_m$ samples from the density with parameter ψ

Density ratios type (2)

$$\mathbb{E}_\psi \left(\frac{h_\theta(X)}{h_\psi(X)}\right)$$

approximated by sample mean if ψ and θ are close

- Likelihood function computed up to a fixed constant

$$\overline{L}(\psi; x) = \frac{L(\psi; x)}{L(\psi_0; x)} = \frac{f_\psi(x)}{f_{\psi_0}(x)}.$$

Maximum likelihood estimation in Strauss process

r fixed $\Rightarrow$ density of exponential family form

- maximum likelihood estimate of (β, γ) based on an observed point pattern x is unique solution to the equations

$$\mathbb{E}_{\beta,\gamma,r} n(X) = n(x) \quad \text{and} \quad \mathbb{E}_{\beta,\gamma,r} s_r(X) = s_r(x)$$

Suppose we know

$$(\beta, \gamma, r) \to \left(\mathbb{E}_{\beta,\gamma,r} n(X), \mathbb{E}_{\beta,\gamma,r} s_r(X)\right)$$

for all values of β, γ, and r

Then given r we have MLE $(\widehat{\beta}(r), \widehat{\gamma}(r))$

Project objective

For X a Strauss process, tabulate functions

$$(\beta, \gamma, r) \to \mathbb{E}_{\beta,\gamma,r} n(X)$$
$$(\beta, \gamma, r) \to \mathbb{E}_{\beta,\gamma,r} s_r(X)$$

Method

- For each value of (β, γ, r) in a grid: Compute means
- Means approximated by sample means:
  Draw `nsamp` samples from model

Usage

- Explore connections between parameters and means
- Ease maximum likelihood estimation

Sequential program

```
for ( (β,γ,r) ∈ grid )
  {
    pp -> resetBeta(beta);
    pp -> resetGamma(gamma);
    pp -> initRelafst(r);

    for ( i = 0; i < nsamp; i++ )
      {
        pp -> emptyprocess();

        for ( j = 0; j < nburn; j++ )
          pp -> makeMHBDstep();

        ...save n(x_i) and s_r(x_i)...
      }
    ...E_{β,γ,r} n(X) = 1/nsamp Σ_{i=1}^{nsamp} n(x_i)...
    ...E_{β,γ,r} s_r(X) = 1/nsamp Σ_{i=1}^{nsamp} s_r(x_i)...
  }
```

Faster sequential program

for ($(\beta, \gamma, r) \in$ grid)

```
  {
    pp -> resetBeta(beta);
    pp -> resetGamma(gamma);
    pp -> initRelafst(r);
    pp -> emptyprocess();

    for ( i = 0; i < nburn; i++ )
      pp -> makeMHBDstep();

    for ( i = 0; i < nsamp; i++ )
      {
        for ( j = 0; j < nspac; j++ )
          pp -> makeMHBDstep();
```

…save $n(x_i) = n(x)$ and $s_r(x_i) = s_r(x)$…

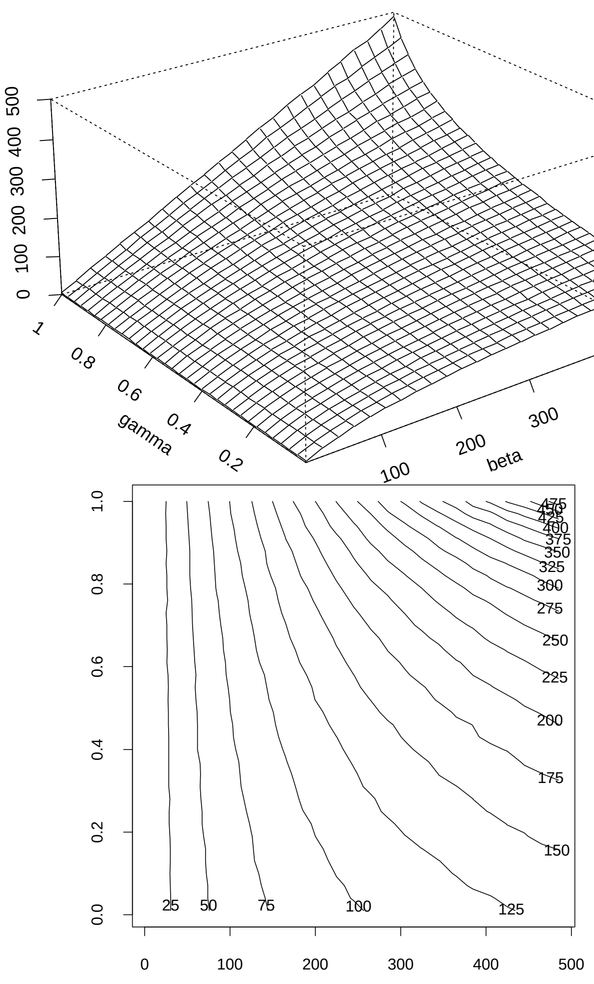
```
      }
```

…$\mathbb{E}_{\beta,\gamma,r} n(X) = \frac{1}{\texttt{nsamp}} \sum_{i=1}^{\texttt{nsamp}} n(x_i)$…

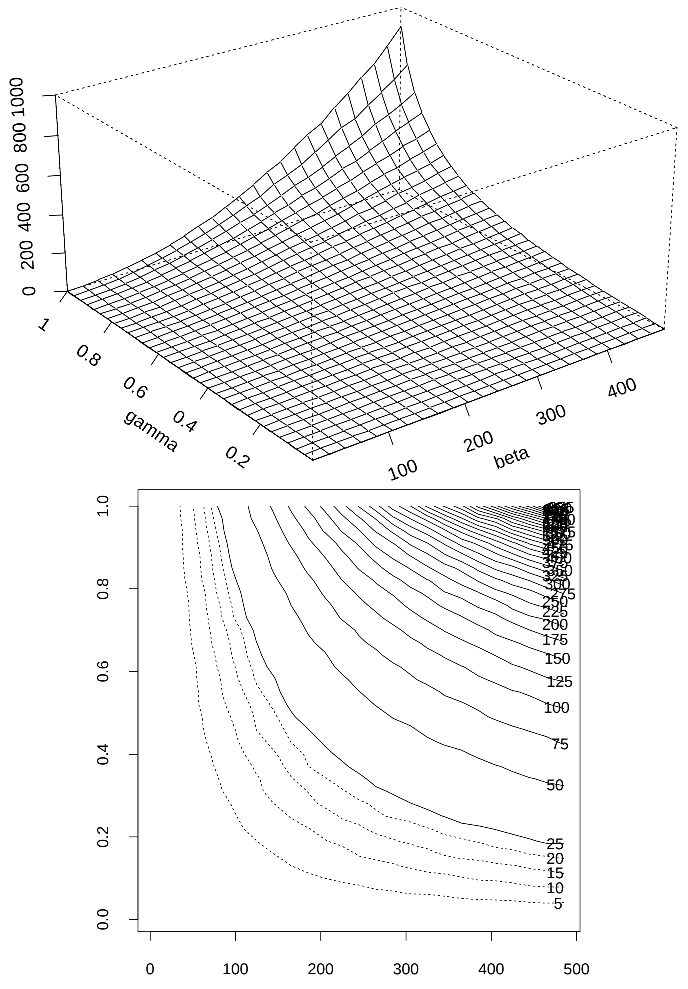
…$\mathbb{E}_{\beta,\gamma,r} s_r(X) = \frac{1}{\texttt{nsamp}} \sum_{i=1}^{\texttt{nsamp}} s_r(x_i)$…

```
  }
```

Point-wise plots of tabulated means ($r = 0.05$)

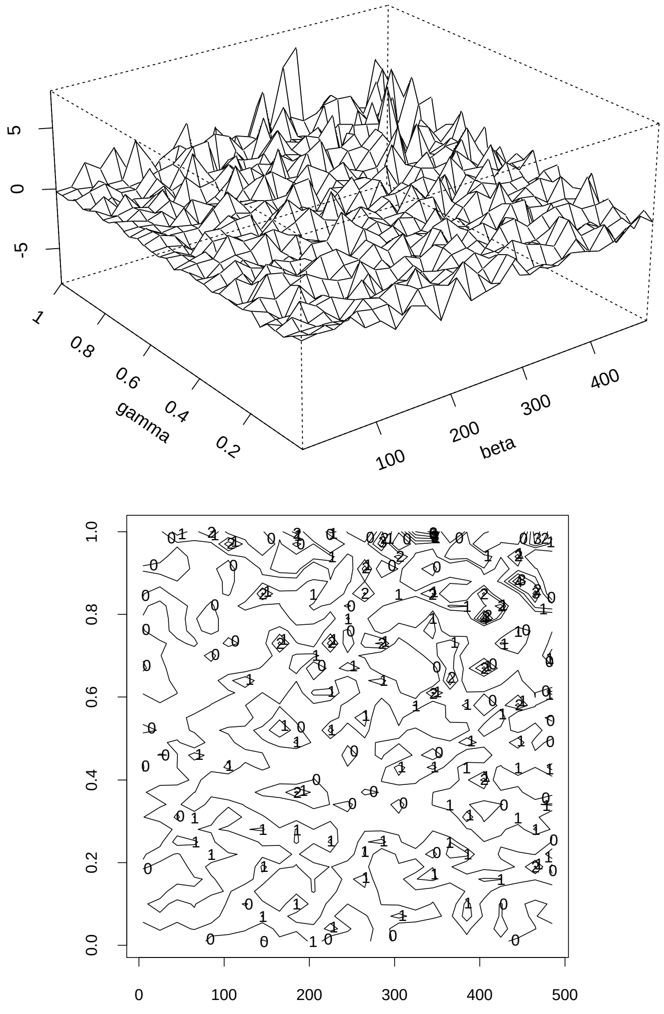


$$\mathbb{E}_{\beta,\gamma,r}\, n(X)$$

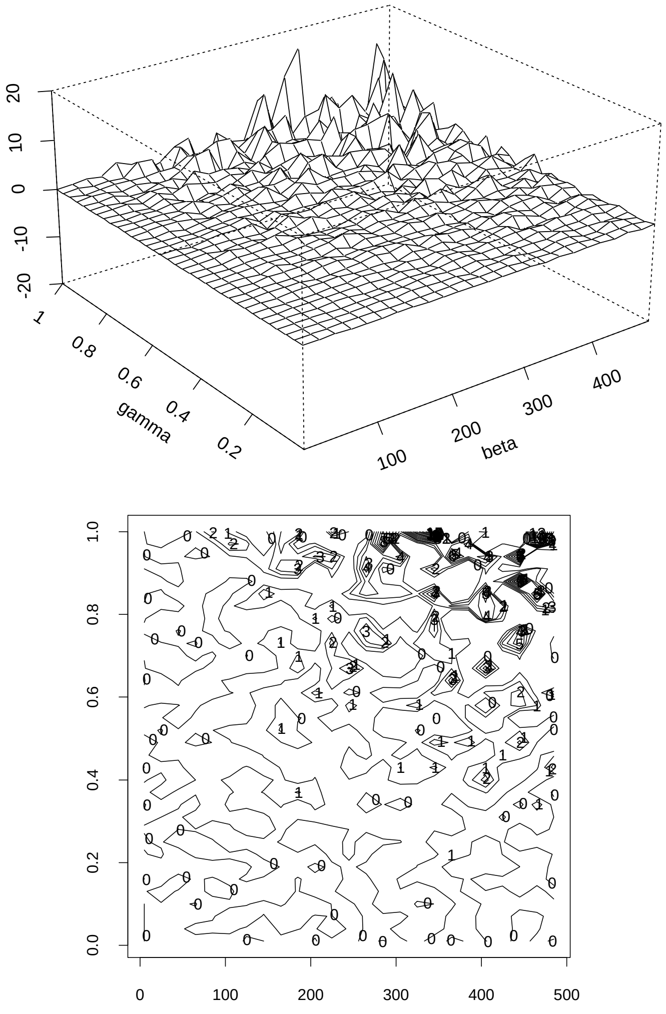


$$\mathbb{E}_{\beta,\gamma,r}\, s_r(X)$$

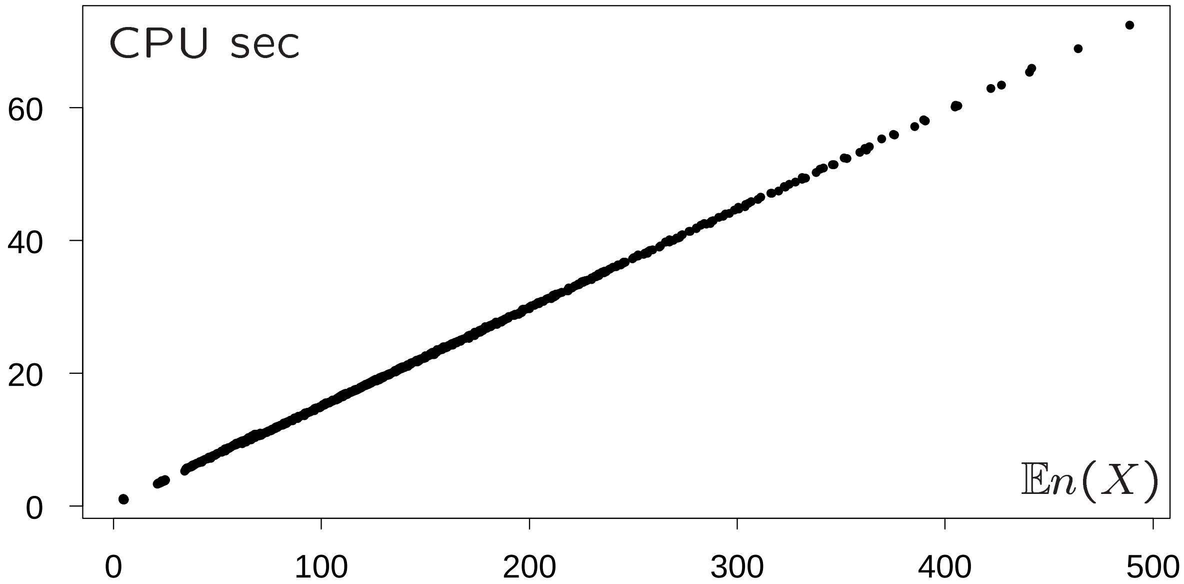
Differences data from two separate runs ($r = 0.05$)



$\mathbb{E}_{\beta,\gamma,r} n(X)$ residuals



$\mathbb{E}_{\beta,\gamma,r} s_r(X)$ residuals



CPU time plotted against the simulated expected number of points (sample average). $r = 0.05$ is fixed and (β, γ) varies.

Embarrassing/perfect parallel

Split program into small independent jobs

- *master* node (1): Allocates jobs and collects results
- *slave* node (rest): Perform work assigned by master

Note: Communication between master and slave only

Advantage: work performed $n-1$ times faster (no waiting)

Other situations: Communication between slaves

Parallelising for point processes

Monte Carlo methods:
Drawing independent samples from the distribution

Compute means:

- Saves time by reusing the same chain
  (reuse since parameter is unchanged)

Changing parameter:

- Not much time saved by reusing
- Isolated jobs: Compute means for 1 grid point value
  Well-suited for making embarrassing parallel

```
main( int argc, char **argv )
{
  int myrank;

  MPI_init( &argc, &argv );
  MPI_Size( MPI_COMM_WORLD, &number_of_processors );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );

  if ( myrank == 0 )
    master();
  else
    slave();

  MPI_Finalize();
}
```

Communication between master and slave

```
MPI_Send( BUF, COUNT, DATATYPE, DEST,   TAG, COMM, IEERROR )
MPI_Recv( BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS  )
```

`BUF` (pointer), `COUNT` (integer),
`DATATYPE` (`MPI` datatype, e.g. `MPI_DOUBLE`):
what is sent/received, where it be can found

`DEST` and `SOURCE` (integers):
destination/source of message (identity number of the node)

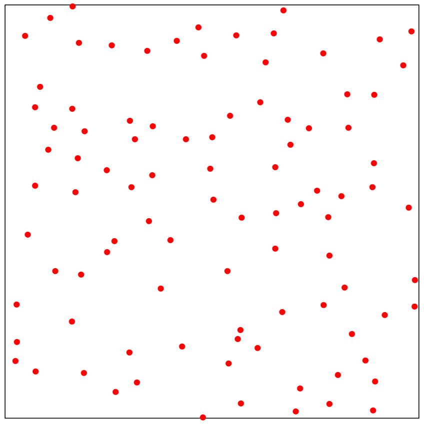
`TAG` (integer):
tag used in the communication (aliases)

`STATUS` (`MPI_Status`):
Functions: `MPI_SOURCE`: returns identity number of node
`MPI_TAG`: returns tag

Simulation experiment: MLE

Simulated Strauss point process

$$(\beta, \gamma, r) = (200, 0.1, 0.05)$$



$n(x) = 95$ and

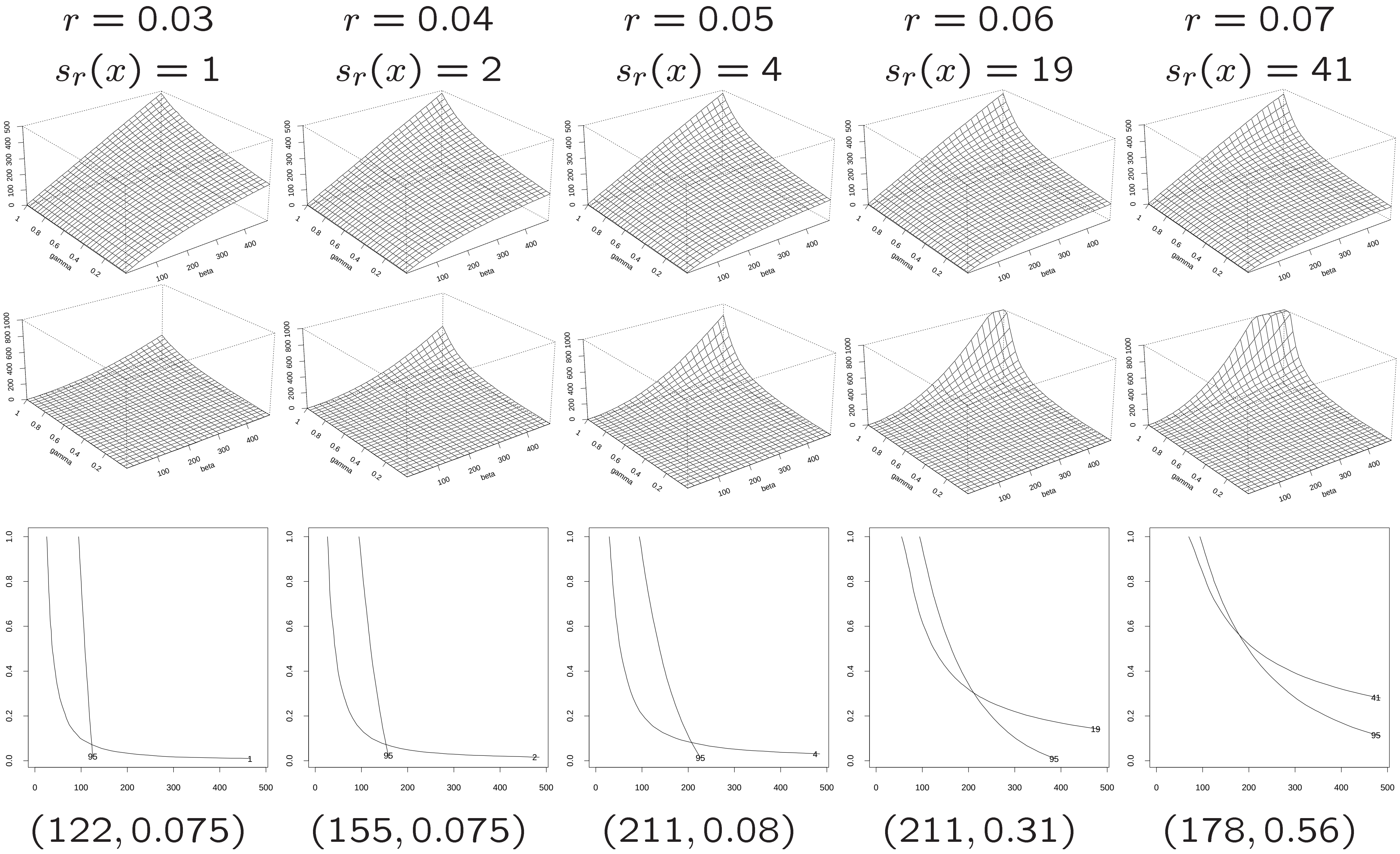
$s_r(x) = 1$ for $r = 0.03$

$s_r(x) = 2$ for $r = 0.04$

$s_r(x) = 4$ for $r = 0.05$

$s_r(x) = 19$ for $r = 0.06$

$s_r(x) = 41$ for $r = 0.07$



$r = 0.03$
$s_r(x) = 1$
$r = 0.04$
$s_r(x) = 2$
$r = 0.05$
$s_r(x) = 4$
$r = 0.06$
$s_r(x) = 19$
$r = 0.07$
$s_r(x) = 41$
gamma
beta
$(122, 0.075)$
$(155, 0.075)$
$(211, 0.08)$
$(211, 0.31)$
$(178, 0.56)$

Future work

- More studies like simulation experiment exploring MLE
- Reparametrizing – Ute Hahn: $(\beta r^2, \gamma)$
- Other interaction models and inhomogeneity
- Edge effects
- Perfect simulation for better sampling