S 1. Supplementary material to Lehsten et al. Simulating efficiently migration in dynamic vegetation models

May 17, 2018

1 Derivation of the distribution of the SMSM kernel

The SMSM will shift the seed north, south, east or west with probability p, to north-west, north-east, south-west or south-east with probability $p/\sqrt{2}$. The discrete probability mass function is presented below

This shift can be seen as a two dimensional random variable (Y, X), where the 'north-south' Y component is independent of the 'east-west' X component. Furthermore, the distribution and hence all marginal moments are identical.

The expected value for X is zero, $E[X] = 0$, as the probability mass function is symmetric around 0. The same holds for the expected value for Y, $E[Y] = 0$.

The variance for X is given by $Var[X] = E[(X-E[X])^2]$. Straightforward calculations give that

$$\begin{aligned} var[X] &= (p + 2p/\sqrt{2})(\Delta x)^2 + 0 + (p + 2p/\sqrt{2})(-\Delta x)^2 \quad (1) \\ &= 2p(1+\sqrt{2})(\Delta x)^2 \quad (2) \end{aligned}$$

It will be convenient to define this quantity as

$$\sigma^2 = 2p(1+\sqrt{2})(\Delta x)^2. \quad (3)$$

Again, $var[X] = var[Y]$ by symmetries. Finally, the covariance between X and Y is zero as the random variables are independent.

The SMSM iterates this kernel several times, say K iterations. Let Z be a column vector

$$Z = \begin{pmatrix} X \\ Y \end{pmatrix} \tag{4}$$

The multivariate central limit theorem then states that

$$\lim_{k\to\infty} \frac{1}{\sqrt{K}} \sum_{k=1}^{K} Z_k - E[Z] \xrightarrow{d} MVN(0, \Sigma) \tag{5}$$

The interpretation of the multivariate central limit theorem is that a suitable scaled version of the sum of random variables converges in distribution, see Shiryaev [1996] to a multivariate Gaussian random variable.

Hence, we find that

$$\sum_{k=1}^{K} \begin{pmatrix} X \\ Y \end{pmatrix} \in MVN\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, K\begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix}\right). \tag{6}$$

Remark

This derivation holds on more general conditions as well. It is possible to derive the expected values, variances and covariances for any discrete probability mass function.

It is then possible to still derive central limit theorem results even if the random variable changes between the iterations. Formal conditions for the convergence of sums of independent but not identically distributed random variables are given for the Martingale central limit theorem, see Hall and Heyde [2014].

2 Derivation of the computational costs of the convolution

The computational complexity of the SMSM can be improved further if the kernel is separable, i.e. if the kernel can be written as a product of univariate terms. This is always possible for kernels that are joint densities for independent random variables, as by definition the joint density is the product of the marginal densities:

$$k_s(x, y) = k_s(x)k_s(y). \tag{7}$$

For separable kernels, it is possible to write the convolution:

$$S(x,y)**k(x,y) = S(x,y)**(k_s(x)**k_s(y)) = (S(x,y)**k(x))**k_s(y). \tag{8}$$

using the Helix transform where we used vector product of the marginal factors is identical to the convolution between these vectors. The last step was derived using the associative property of convolutions. The result is that the convolution can be computed in two steps, first the convolution between a matrix and a vector, leading to a computational cost of

O($\mathbf{N^2}$ R) followed by another convolution between a matrix and a vector, hence the overall cost will be

$$O(N^2K(R+R)). \tag{9}$$

References

Peter Hall and Christopher C Heyde. *Martingale limit theory and its application.* Academic press, 2014.

Albert N Shiryaev. *Probability, volume 95 of Graduate texts in mathematics.* Springer-Verlag, New York,, 1996.

Supplement 2

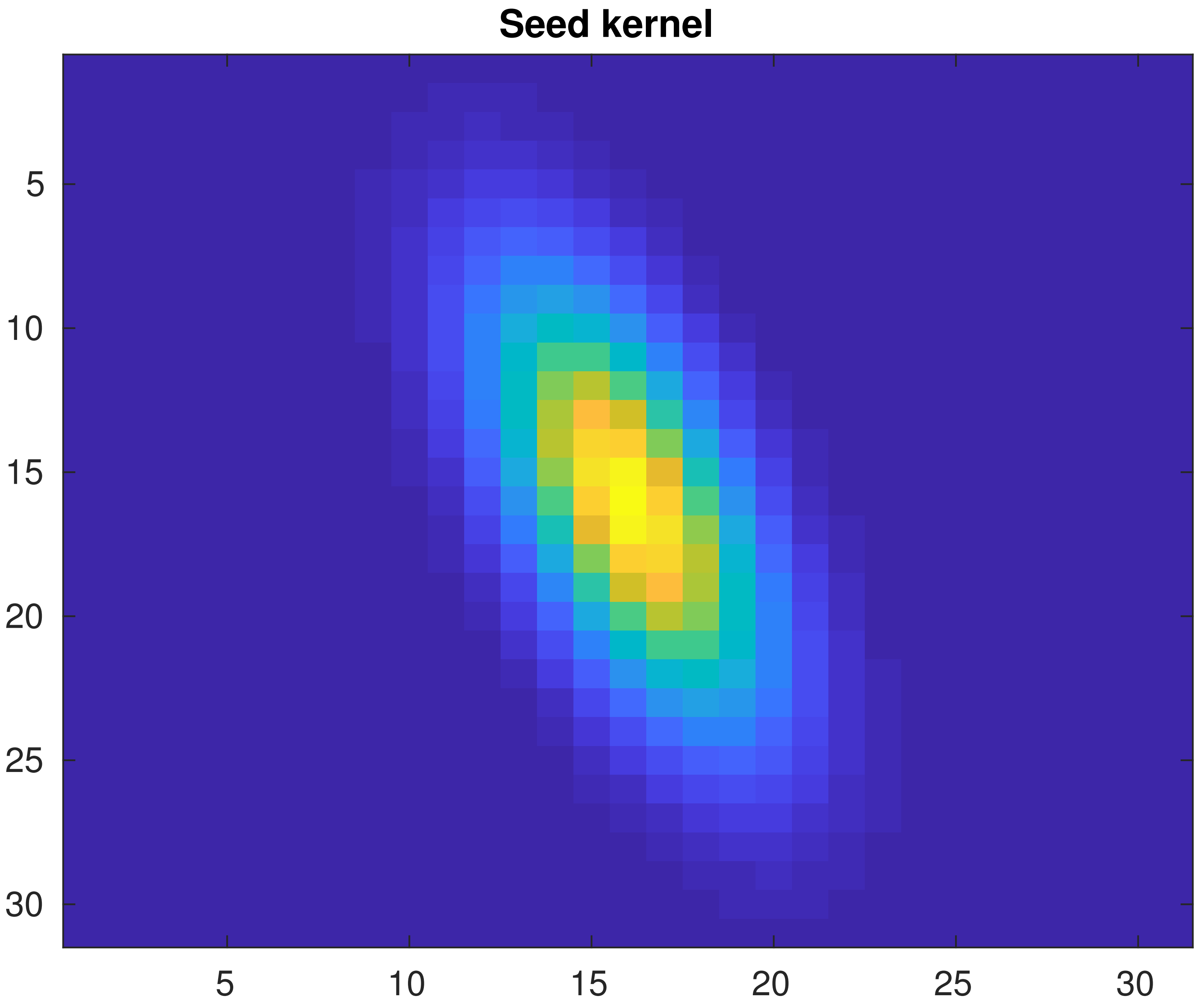
Computation speed test for seed dispersal using the FFTM, SMSM, (with and without terrain) versus explit simulation.

Supplement to Lehsten et al. : Simulating efficiently migration in dynamic vegetation models.

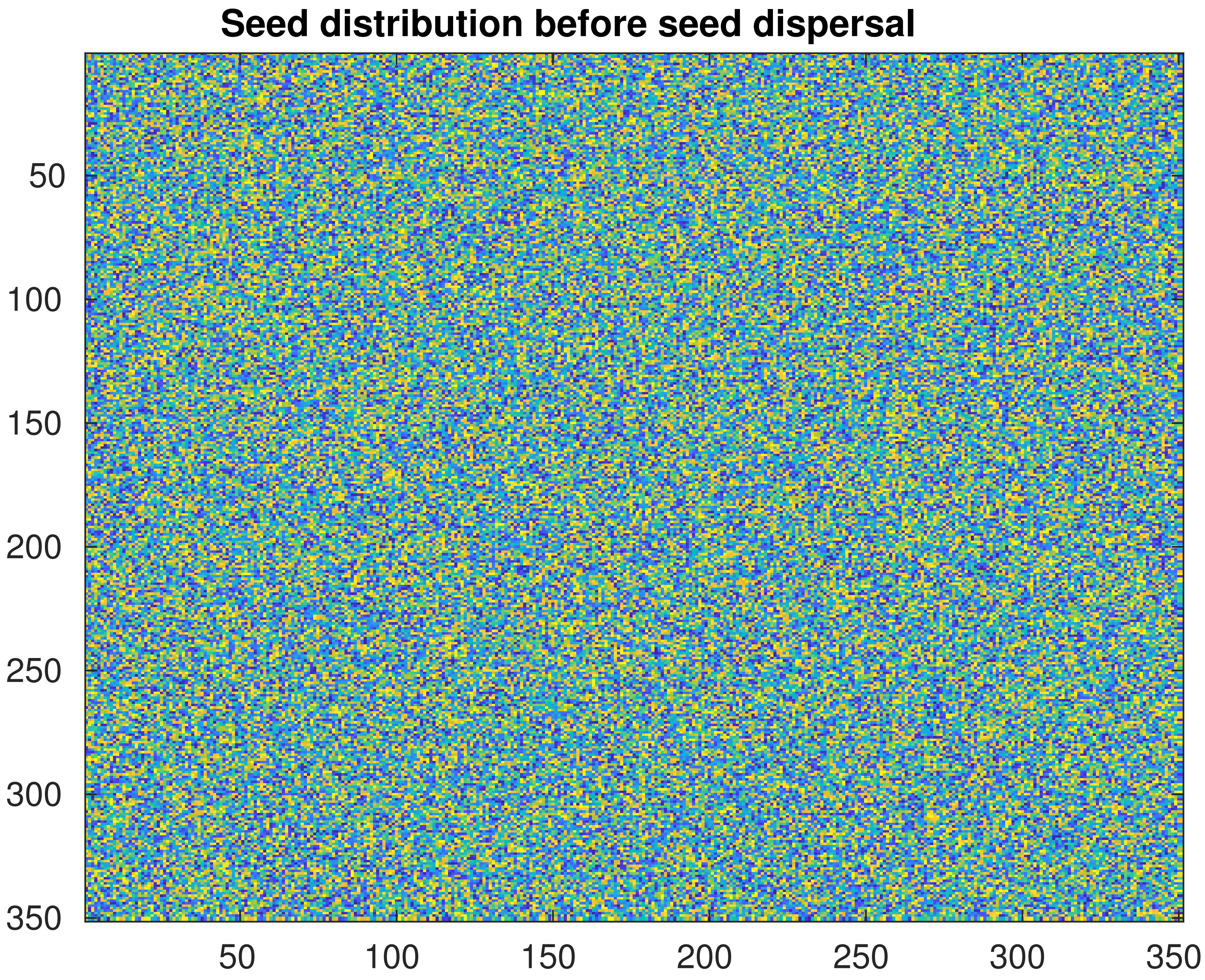
```
% clear the memory
clear
%
```

Generate a seed kernel

```
x1 = -3:.2:3; x2 = -3:.2:3;
[X1,X2] = meshgrid(x1,x2);
%here we generate a seed kernel which has a higher extent in NW to SE direction

F = mvnpdf([X1(:) X2(:)],[0 0],[.25 .3; .3 1]); % use a multivariate normal density distribution
kernelorg = reshape(F,length(x2),length(x1));   % this is now our seed kernel with the size of 31 by 31 cells
kernelorg = kernelorg/(sum(sum(kernelorg)));    % norm to unity sum
%plot the seed kernel
figure;imagesc(kernelorg)
title ('Seed kernel')
snapnow
```


Seed kernel
5
10
15
20
25
30
5
10
15
20
25
30

```
%increase area size by looping
round=0;
for size_area=31:20:351%31:20:331 % area should always be odd and quadratic for simplicity decrease the last number if you want to test smaller areas

    round=round+1;
    seed_prod=rand(size_area); % assign random values to the seed production

    if size_area==351 % if in the last loop print result
        figure
        imagesc(seed_prod)
        title ('Seed distribution before seed dispersal')
        snapnow
    end

    % add an empty border to avoid edge effects
    seed_prod_with_border=zeros(size_area*3);  %

    seed_prod_with_border(size_area+1:2*size_area,size_area+1:2*size_area)=seed_prod;

    %extend size and transform it by swaping columns and rows in a way that the
    %centercells are now in the corners of the kernel
    kernel_large=zeros(size_area*3);

    %place original kernel in the middle
    kernel_large((ceil(size_area*3/2)-ceil(size(kernelorg,1)/2)):(floor(size_area*3/2)+floor(size(kernelorg,1)/2)),...
        (ceil(size_area*3/2)-ceil(size(kernelorg,2)/2)):(floor(size_area*3/2)+floor(size(kernelorg,2)/2)))=...
        kernelorg;
```


Seed distribution before seed dispersal
50
100
150
200
250
300
350
50
100
150
200
250
300
350

Here comes the transformation of the seed kernel, see Methods section

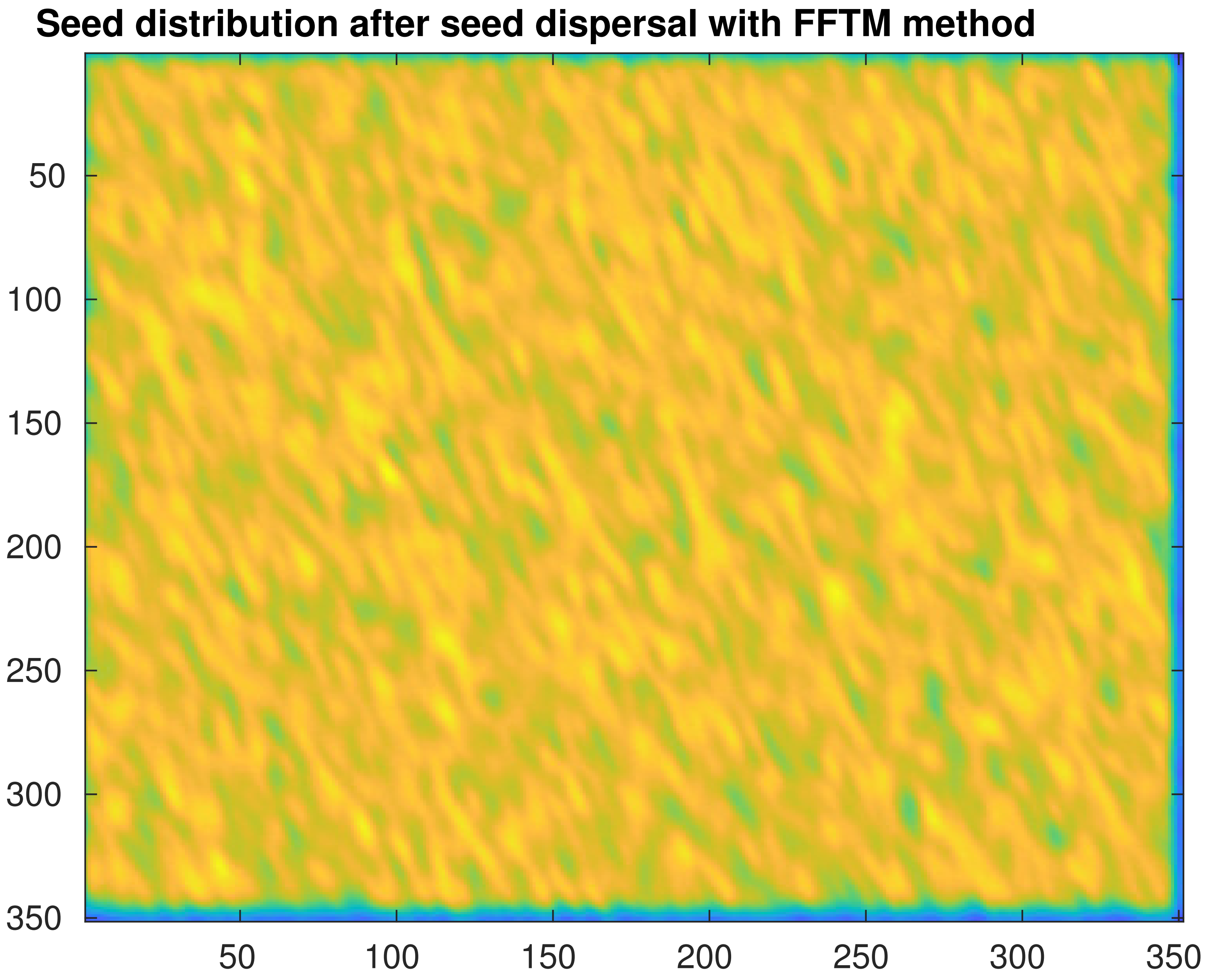
```
kernel_large_transformed = kernel_large(mod((1:size_area*3) - (size_area*3+3)/2, size_area*3) + 1, mod((1:size_area*3) - (size_area*3+3)/2, size_area*3) + 1) ;
```

Perform the fft based seed dispersal

```
% start taking time
tic
```

The fft based seed dispersal

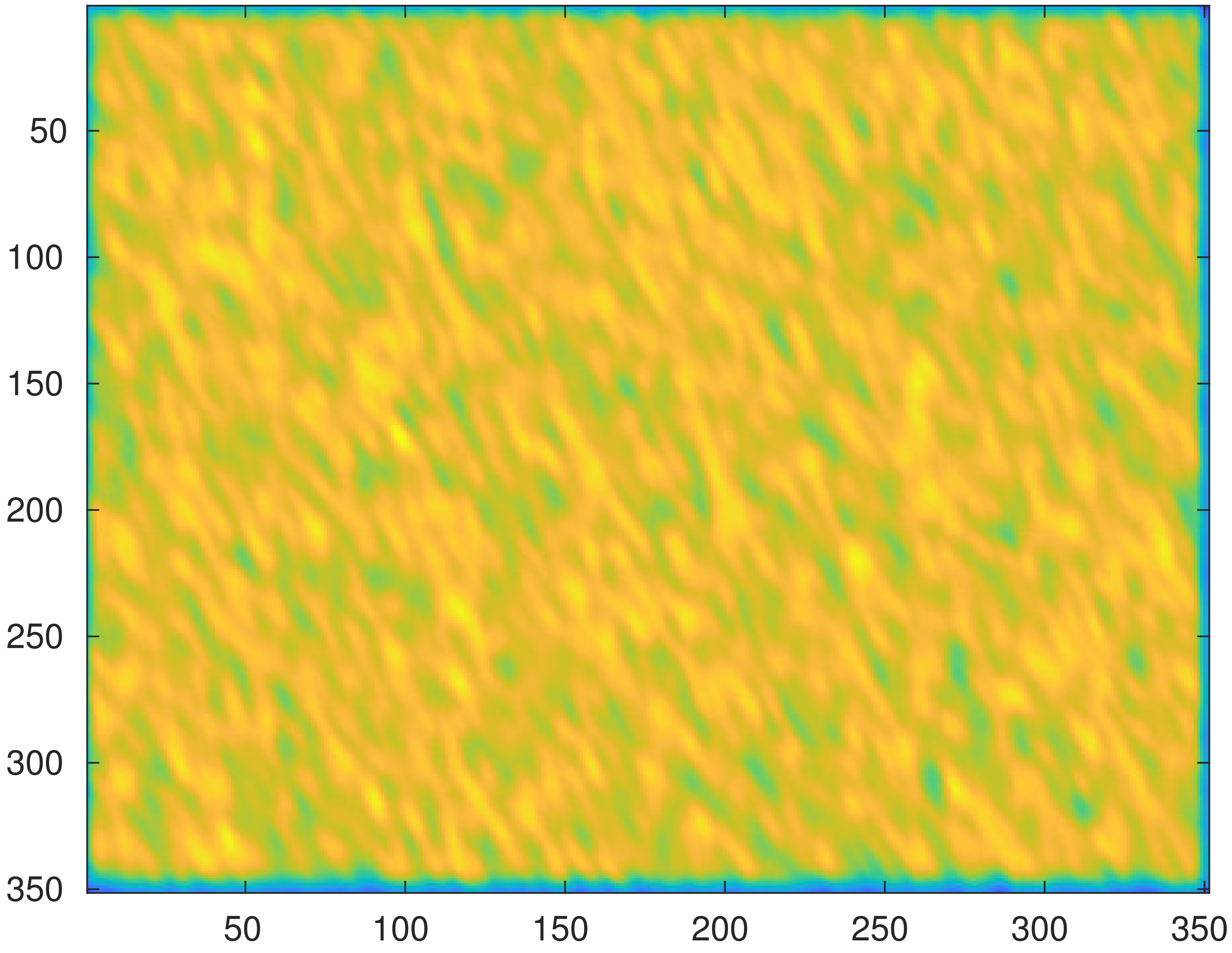
```
newseeds_fft_with_border=abs(ifft2(fft2(seed_prod_with_border).*fft2(kernel_large_transformed)));

% extract the area without the borders
newseeds_fft=newseeds_fft_with_border(size_area+1:2*size_area,size_area+1:2*size_area);
if size_area==351 % if in the last loop print result
    figure
    imagesc(newseeds_fft)
    title ('Seed distribution after seed dispersal with FFTM method')
    snapnow
end

% end taking time
fft_time(round)=toc;
```


Seed distribution after seed dispersal with FFTM method
50
100
150
200
250
300
350
50
100
150
200
250
300
350

Perform the explicit seed dispersal*

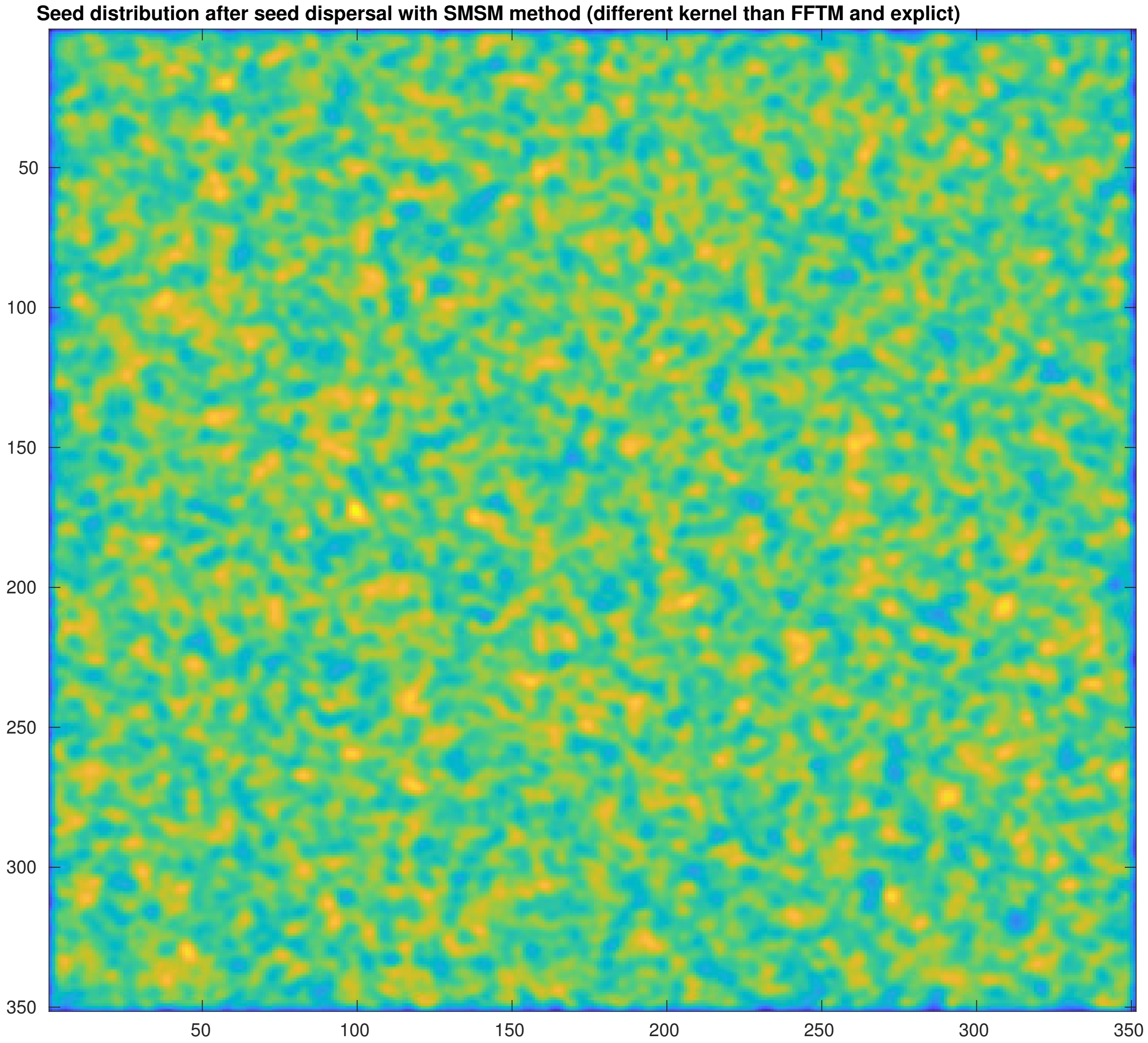
```
% initialise the area
newseeds_exp_e=zeros(size_area);

% start taking time
tic

% loop through the whole seed array to select a seed donor
for col1=1:(size_area)
    for row1=1:(size_area)

        % now only loop through those parts of the seed array which could
        % potentially be reached (depending on the size of the kernel) as a
        % seed receiver
        for pos_col=max(ceil(size(kernelorg,2)/2)-(col1)+1,1):min(ceil(size(kernelorg,2)/2)-(col1)+(size_area),size(kernelorg,2)) % this is in the kernel

            for pos_row=max(ceil(size(kernelorg,2)/2)-(row1)+1,1):min(ceil(size(kernelorg,2)/2)-(row1)+(size_area),size(kernelorg,2)) % this is in the kernel

                %disperse the seeds
                newseeds_exp_e(row1,col1)=newseeds_exp_e(row1,col1)+...
                    seed_prod(pos_row-ceil(size(kernelorg,2)/2)+row1,pos_col-ceil(size(kernelorg,2)/2)+col1)*kernelorg(pos_row,pos_col);

            end
        end
    end
end

% stop taking time for the explicit method
exp_time(round)=toc;


if size_area==351 % if in the last loop print result
    figure
    imagesc(newseeds_exp_e)
    title ('Seed distribution after seed dispersal with explicit method')
    snapnow
end
```

Seed distribution after seed dispersal with explicit method



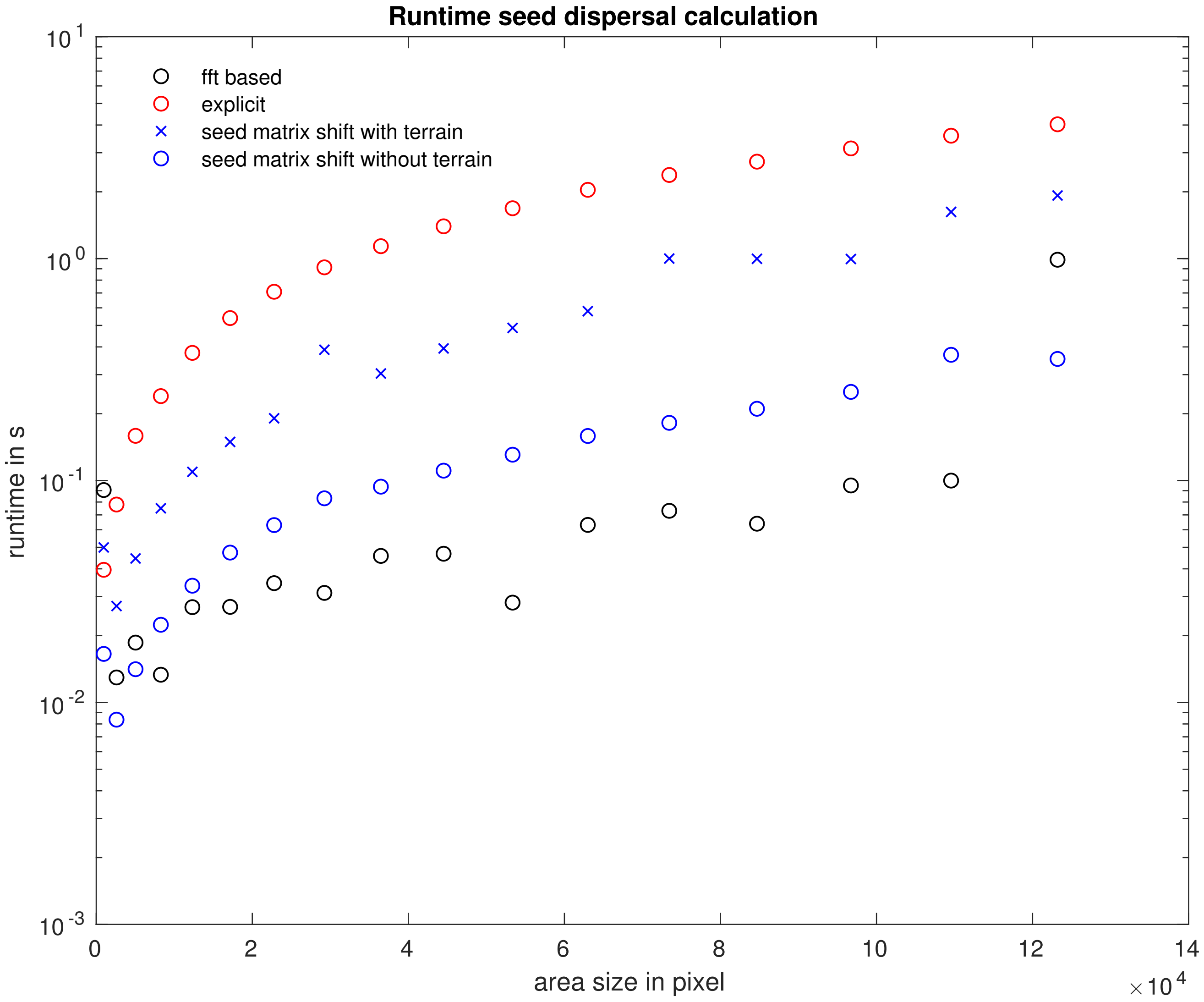
Perform the SMSM

```
%seeds=  seed matrix
% p=matrix of probabilities
% n=number of iterations

%now increase size to get rid of edge effects +edge in both directions

temp_seed_end=seed_prod_with_border;

terrain=ones(size(temp_seed_end)); % no specific terrain defined, but to keep this in for the time estimation

disprate=ones(size(temp_seed_end))*0.08;

x_max=size(temp_seed_end,1);
y_max=size(temp_seed_end,2);

seed_end=temp_seed_end;
tic
for step=1:10  % 10 steps for the SMSM

    temp_seed_end=zeros(size(seed_end));

    %neighbours
    temp_seed_end(3:x_max,2:y_max-1)=temp_seed_end(3:x_max,2:y_max-1)+disprate(3:x_max,2:y_max-1).*seed_end(2:x_max-1,2:y_max-1).*terrain(3:x_max,2:y_max-1);
    temp_seed_end(1:x_max-2,2:y_max-1)=temp_seed_end(1:x_max-2,2:y_max-1)+disprate(1:x_max-2,2:y_max-1).*seed_end(2:x_max-1,2:y_max-1).*terrain(1:x_max-2,2:y_max-1);
    temp_seed_end(2:x_max-1,3:y_max)=temp_seed_end(2:x_max-1,3:y_max)+disprate(2:x_max-1,3:y_max).*seed_end(2:x_max-1,2:y_max-1).*terrain(2:x_max-1,3:y_max);
    temp_seed_end(2:x_max-1,1:y_max-2)=temp_seed_end(2:x_max-1,1:y_max-2)+disprate(1:x_max-2,2:y_max-1).*seed_end(2:x_max-1,2:y_max-1).*terrain(2:x_max-1,1:y_max-2);


    % diagonalen
    temp_seed_end(3:x_max,3:y_max)=temp_seed_end(3:x_max,3:y_max)+disprate(3:x_max,3:y_max).*seed_end(2:x_max-1,2:y_max-1)/sqrt(2).*terrain(3:x_max,3:y_max);
    temp_seed_end(1:x_max-2,1:y_max-2)=temp_seed_end(1:x_max-2,1:y_max-2)+disprate(1:x_max-2,1:y_max-2).*seed_end(2:x_max-1,2:y_max-1)/sqrt(2).*terrain(1:x_max-2,1:y_max-2);
    temp_seed_end(1:x_max-2,3:y_max)=temp_seed_end(1:x_max-2,3:y_max)+disprate(1:x_max-2,3:y_max).*seed_end(2:x_max-1,2:y_max-1)/sqrt(2).*terrain(1:x_max-2,3:y_max);
    temp_seed_end(3:x_max,1:y_max-2)=temp_seed_end(3:x_max,1:y_max-2)+disprate(3:x_max,2:y_max-1).*seed_end(2:x_max-1,2:y_max-1)/sqrt(2).*terrain(3:x_max,1:y_max-2);

    %central
    temp_seed_end(2:x_max-1,2:y_max-1)=temp_seed_end(2:x_max-1,2:y_max-1)+seed_end(2:x_max-1,2:y_max-1)-...
        disprate(3:x_max,2:y_max-1).*seed_end(2:x_max-1,2:y_max-1)-...
        disprate(1:x_max-2,2:y_max-1).*seed_end(2:x_max-1,2:y_max-1)-...
        disprate(2:x_max-1,3:y_max).*seed_end(2:x_max-1,2:y_max-1)-....
        disprate(1:x_max-2,2:y_max-1).*seed_end(2:x_max-1,2:y_max-1)-...
        disprate(1:x_max-2,1:y_max-2).*seed_end(2:x_max-1,2:y_max-1)/sqrt(2)-...
        disprate(3:x_max,3:y_max).*seed_end(2:x_max-1,2:y_max-1)/sqrt(2)-...
        disprate(1:x_max-2,3:y_max).*seed_end(2:x_max-1,2:y_max-1)/sqrt(2)-...
        disprate(3:x_max,2:y_max-1).*seed_end(2:x_max-1,2:y_max-1)/sqrt(2)   ;

    seed_end=temp_seed_end;

end
sms_time(round)=toc;


%go back to original size

newseeds_smsm=seed_end((size_area+1):(size_area+size_area),(size_area+1):(size_area+size_area));
disprate=    0.08;
 tic
for step=1:10  % 10 steps for the SMSM

    temp_seed_end=zeros(size(seed_end));
    seed_shift=disprate*seed_end(2:x_max-1,2:y_max-1);
    %neighbours
    temp_seed_end(3:x_max,2:y_max-1)=temp_seed_end(3:x_max,2:y_max-1)+ seed_shift;
    temp_seed_end(1:x_max-2,2:y_max-1)=temp_seed_end(1:x_max-2,2:y_max-1)+seed_shift;
    temp_seed_end(2:x_max-1,3:y_max)=temp_seed_end(2:x_max-1,3:y_max)+   seed_shift;
    temp_seed_end(2:x_max-1,1:y_max-2)=temp_seed_end(2:x_max-1,1:y_max-2)+seed_shift;

        seed_shift=seed_shift/sqrt(2);
    % diagonals
    temp_seed_end(3:x_max,3:y_max)=temp_seed_end(3:x_max,3:y_max)+seed_shift;
    temp_seed_end(1:x_max-2,1:y_max-2)=temp_seed_end(1:x_max-2,1:y_max-2)+seed_shift;
    temp_seed_end(1:x_max-2,3:y_max)=temp_seed_end(1:x_max-2,3:y_max)+seed_shift;
    temp_seed_end(3:x_max,1:y_max-2)=temp_seed_end(3:x_max,1:y_max-2)+seed_shift;

    %central
    temp_seed_end(2:x_max-1,2:y_max-1)=temp_seed_end(2:x_max-1,2:y_max-1)+seed_end(2:x_max-1,2:y_max-1)+...
        -4*(seed_shift*(1+sqrt(2)));
    seed_end=temp_seed_end;

end
sms_time_s(round)=toc;

if size_area==351 % if in the last loop print result
    figure;
    imagesc(newseeds_smsm)
    title ('Seed distribution after seed dispersal with SMSM method (different kernel than FFTM and explict)')
    snapnow
end





%remember the area size for this round
area_size_disp(round)=size_area^2;
```



Seed distribution after seed dispersal with SMSM method (different kernel than FFTM and explict)

```
end




% plot the runtime
figure;plot(area_size_disp,fft_time,'ok',area_size_disp,exp_time,'or',area_size_disp,sms_time,'xb',area_size_disp,sms_time_s,'ob')
% set y axis to log for better visibility
set(gca, 'YScale', 'log')

% label
legend_handle= legend({'fft based','explicit','seed matrix shift with terrain','seed matrix shift without terrain'},'location', 'northwest')
            set(legend_handle, 'Box', 'off')
legend('fft based','explicit','seed matrix shift with terrain','seed matrix shift without terrain')
title('Runtime seed dispersal calculation')
ylabel('runtime in s')
xlabel('area size in pixel')

snapnow
```


Runtime seed dispersal calculation
fft based
explicit
seed matrix shift with terrain
seed matrix shift without terrain
runtime in s
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
0
2
4
6
8
10
12
14
area size in pixel
$\times 10^{4}$