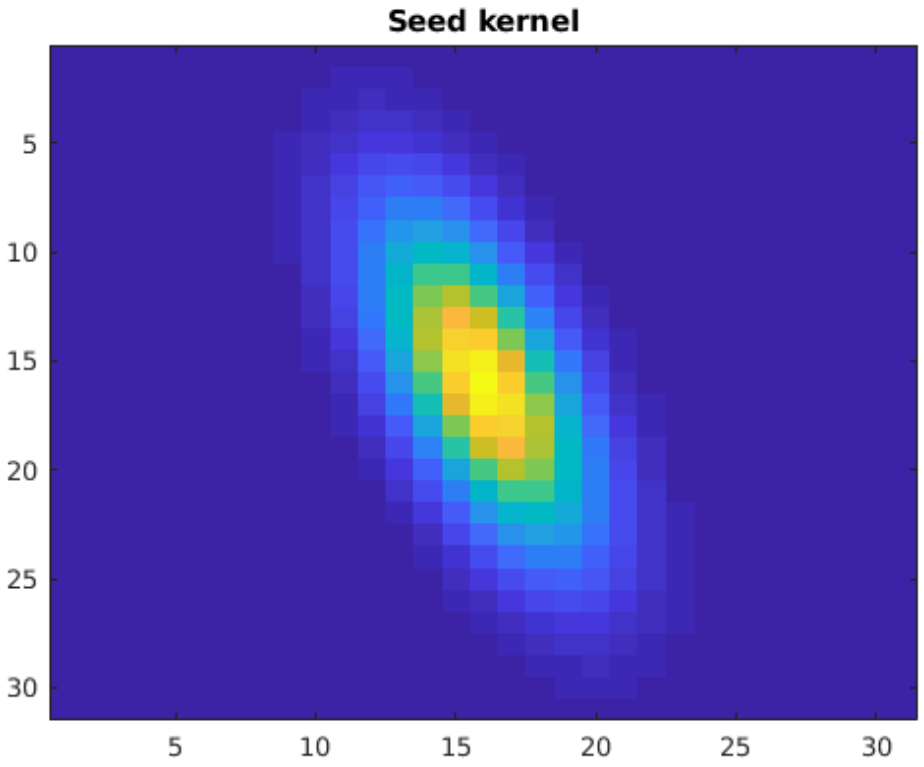
Computation speed test for seed dispersal fft based versus explit simulation.

Supplement to Lehsten et al. : Simulating migration in dynamic vegetation models efficiently with LPJ-GM 1.0 .

```
% 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')
```



S2.1 Seed kernel

```
%increase area size by looping
round=0;
for size_area=31:20:351 % 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

    % 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;
```

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);

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

Perform the explicit seed dispersal

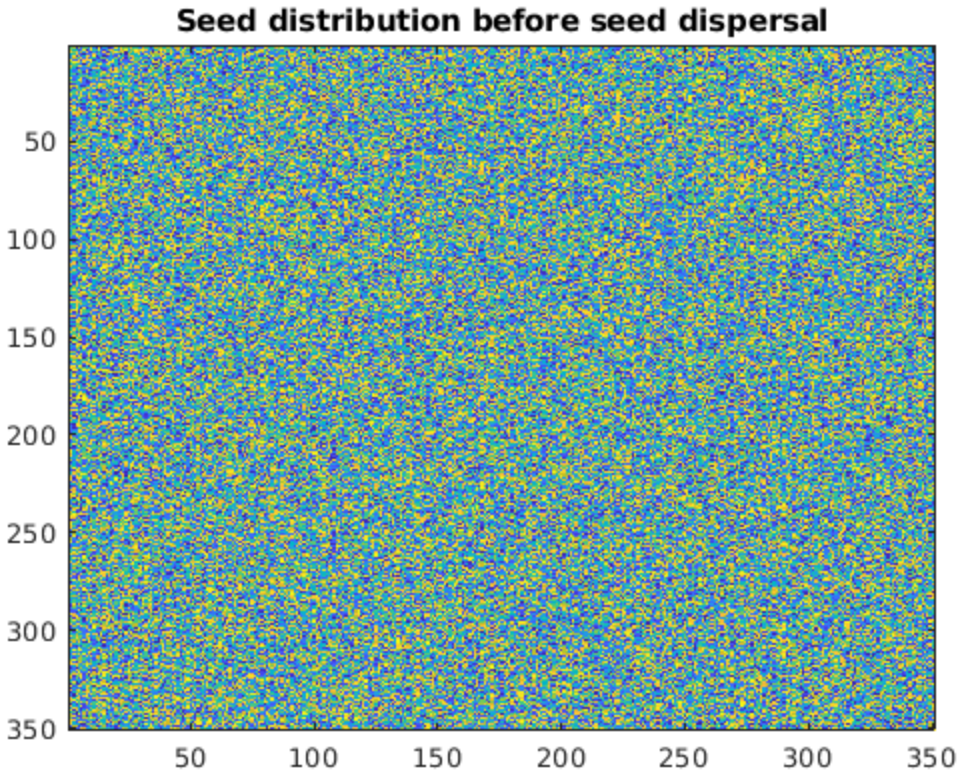
```
    % initialise the area
    newseeds_exp_e=zeros(size_area);

    % 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;
```

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;
```

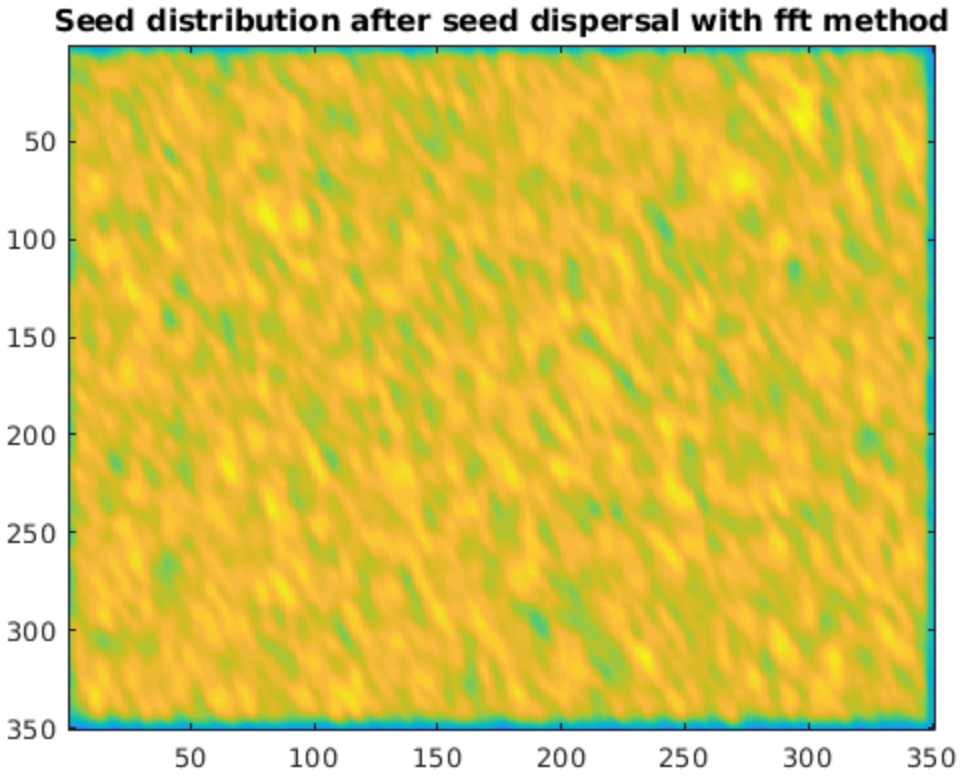
```
   x_max=size(temp_seed_end,1);
   y_max=size(temp_seed_end,2);

   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));

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

end

% plot the last rounds seed matrix, and the seeds after dispersal for
 both
% methods as a check
% The plots show the original seed distribution which is random and
 the
% dispersed seed distribution which shows that the kernel is elongated
 in
% NW to SE direction and that the borders have lower amounts of seeds
 since
% they have a lower area from which they receive seeds.



figure ;imagesc(seed_prod)
title ('Seed distribution before seed dispersal')
```



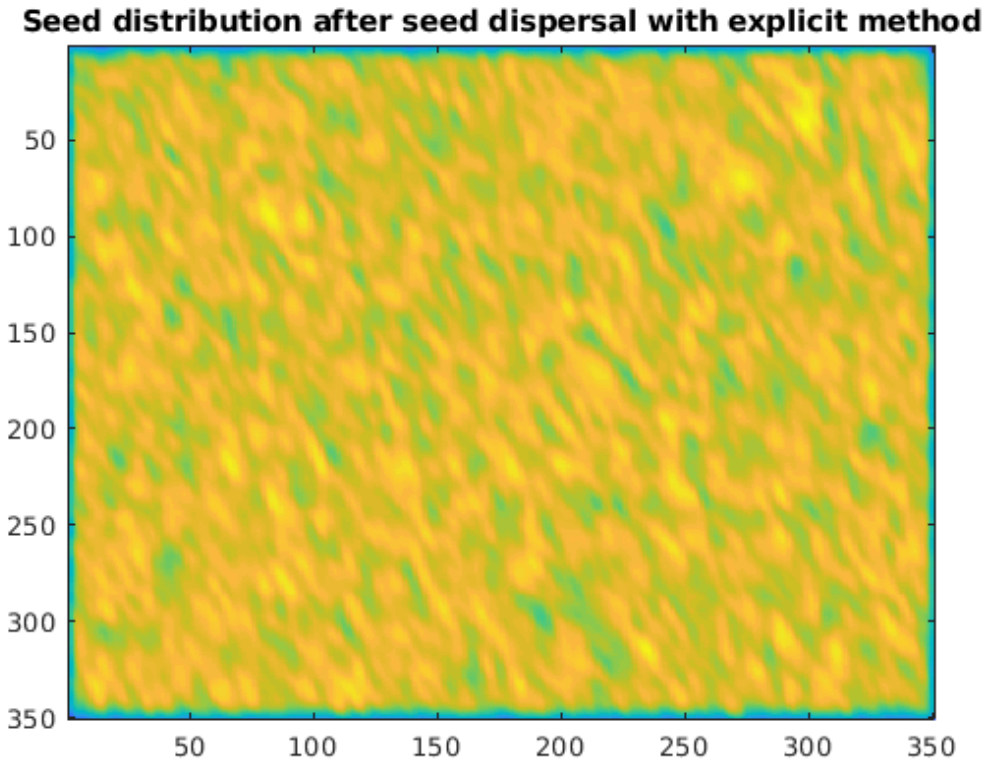
S 2.2 Seed distribution before seed dispersal

```
figure;imagesc(newseeds_fft)
title ('Seed distribution after seed dispersal with fft method')
```



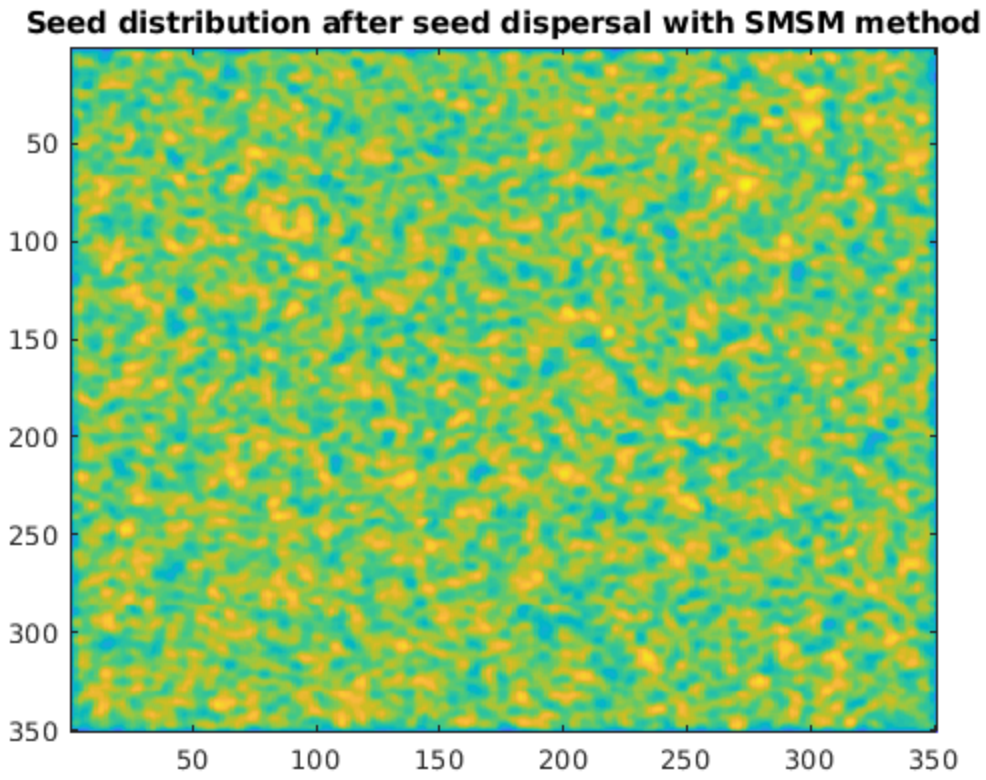
S 2.3 Seed distribution after seed dispersal with fft method

```
figure;imagesc(newseeds_exp_e)
title ('Seed distribution after seed dispersal with explicit method')
```



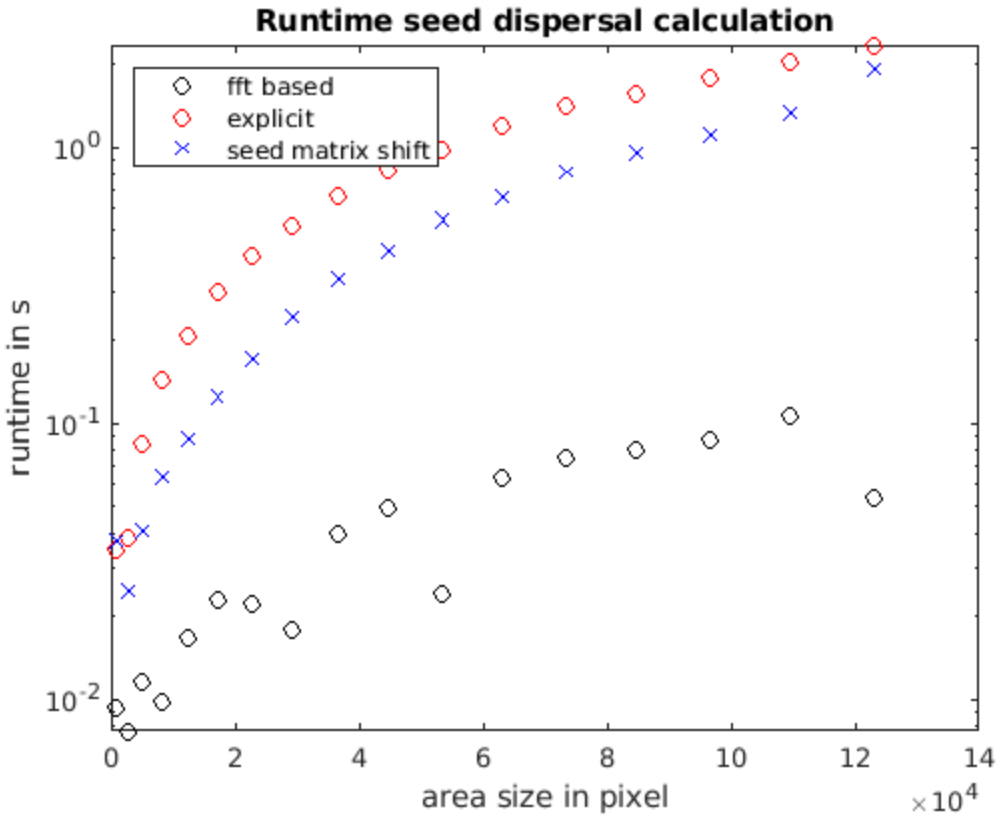
S 2.4 Seed distribution after seed dispersal with explicit method

```
figure;imagesc(newseeds_smsm)
title ('Seed distribution after seed dispersal with SMSM method')
```



S 2.5 Seed distribution after seed dispersal with SMSM method

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

% label
legend('fft based','explicit','seed matrix
 shift','location','northwest')
title('Runtime seed dispersal calculation')
ylabel('runtime in s')
xlabel('area size in pixel')
```



S 2.6 Run time of seed dispersal calculation. Note that though there are very pronounced differences between the different methods (FFTM, SMSM), as shown in table 1 these differences are not shown in the C++ implementation within LPJ-GUESS.

Published with MATLAB® R2018b