Project 1
Scientific Computing III

R.G.A. Deckers

Contents

Creating the matrices K, M, and A

First, before we can do anything else, we of course have to create our matrices. the function we use for doing so is given in listing 2.

Power method

Implementation

The implementation of the power-method is given in listing 3.

Convergence

Let $1 = |\lambda_1| \geq \ldots \geq |\lambda_n|$ be the eigenvalues of a matrix A and $\vec{e}_1, \ldots, \vec{e}_n$ the corresponding eigenvectors. Then $A^k\vec{x} = \sum_{i=1}^{i=n} a_i \lambda_i^k \vec{e}_i$. where a_i are the eigenspace coefficients of $\vec{x}$. All eigenvectors with eigenvalues less than one converges to zero for $k \to \infty$, therefore if all λ_i for $i \neq 1$ are smaller than λ_1 the rate of convergence in the first order will be $|\lambda_1/\lambda_2|$. If the dominant eigenvalue is not unique (down to a sign) than the eigenvalue will still convert, at a rate defined by the next eigenvalue less than λ_1 in the absolute, but the eigenvector will not.

Results

Eigenvector

The eigenvector is plotted in figure 2.

Iterations vs. Matrix Size

The iteration count is plotted in figure 1.

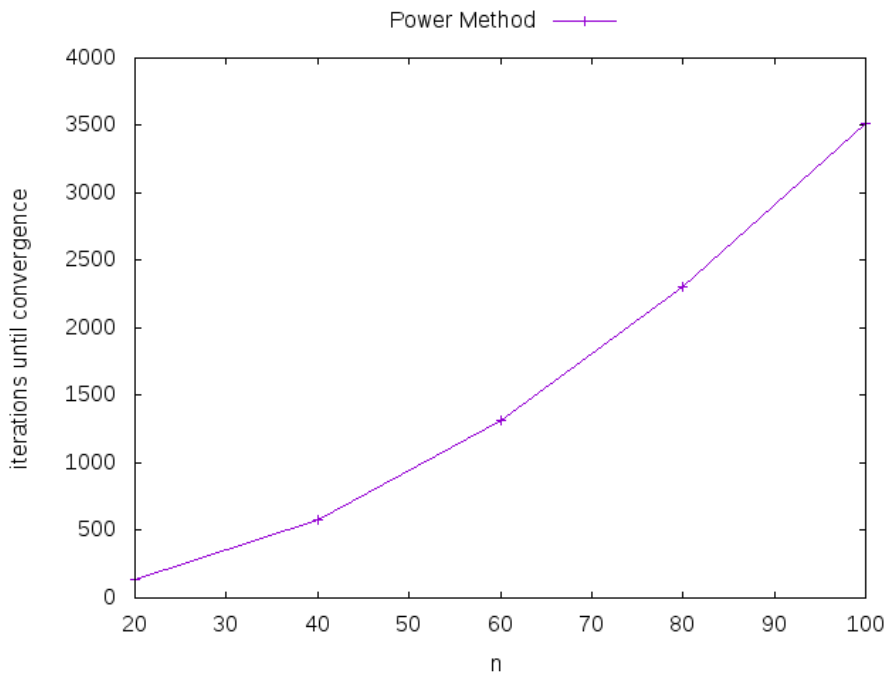

Figure 1: Iterations required to find the largest and smallest eigenvalue of A

Error

The relative error is plotted in figure 4.

Inverse iteration

Implementation

In order to implement inverse iteration, we find the smallest eigenvector by using power iteration over $A - \lambda_1 I$ and adding λ_1 to the result. We then use power iteration over $(A - \lambda_n I)^{-1}$ to compute the corresponding eigenvector. We do however not compute the inverse directly but use LU decomposition for faster iteration. The technical implementation is shown in listing **??**.

Convergence

The inverse iteration converges linearly just like the power method. This is because, as described above, it can be implemented using just power iteration.

results

Eigenvectors

The eigenvector is plotted in figure 2.

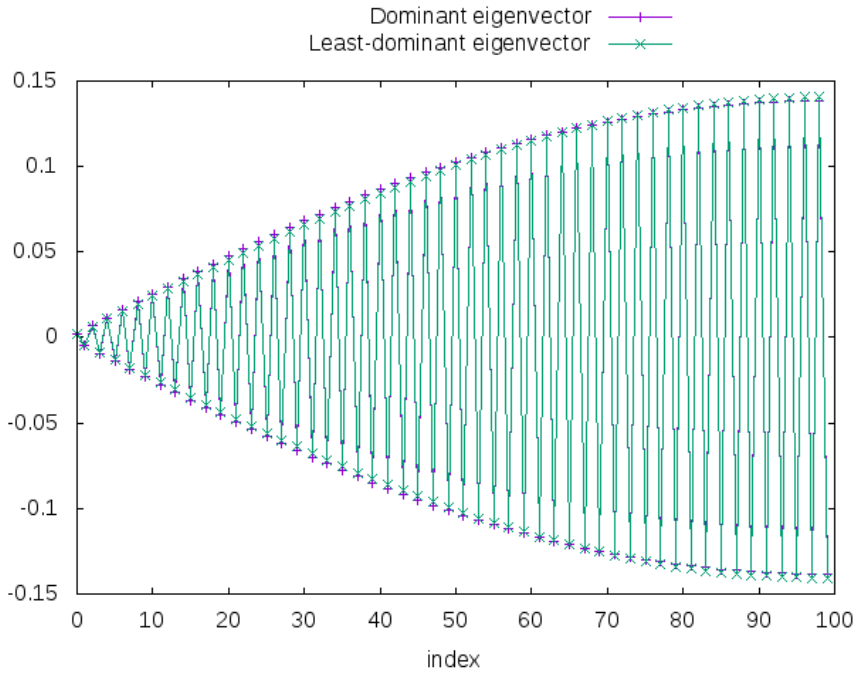

Figure 2: eigenvectors corresponding to the largest and smalles eigenvalue.

iterations vs. matrix size

The iteration count is plotted in figure 3.

error vs. matrix size

The relative error is plotted in figure 4. The condition number a function of the matrix size is plotted in figure 5, this difference explains the difference in relative errors.

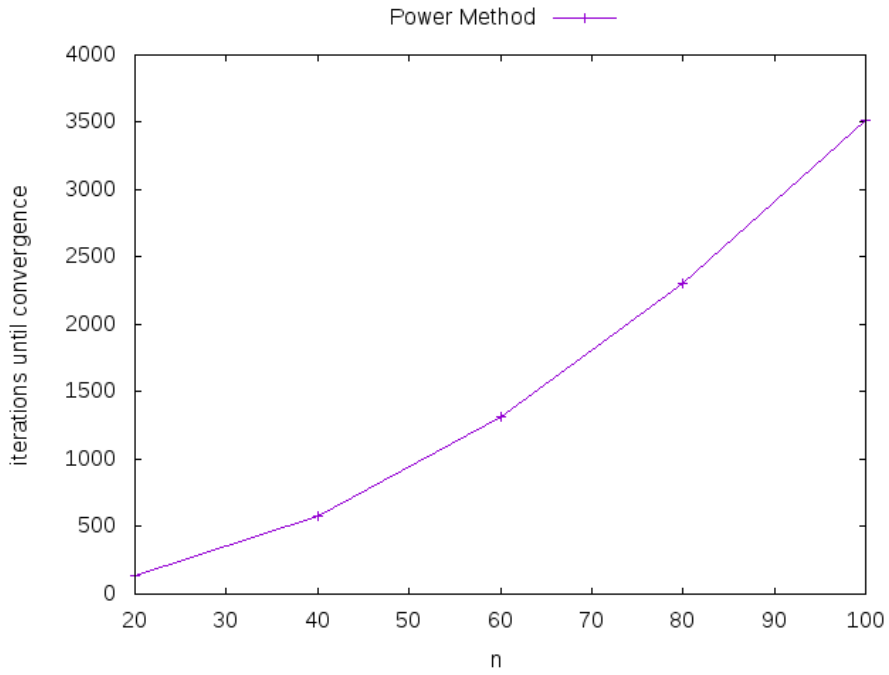

Figure 3: Iterations required to find the largest eigenvector of A using inverse iteration

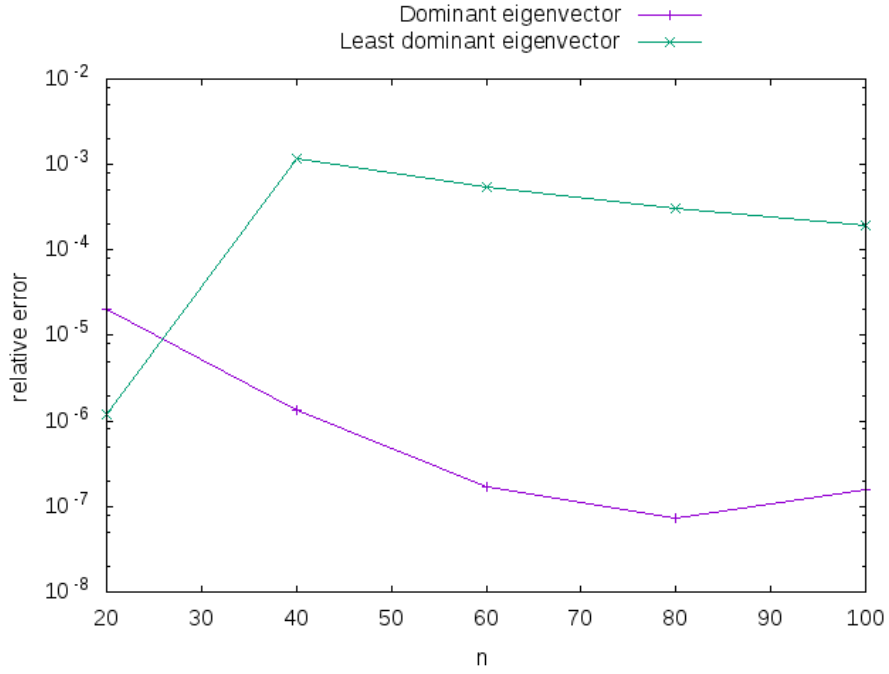

Figure 4: Relative error of the power method corresponding to the largest and smalles eigenvalue as a function of matrix size.

Source Code

Listing 1: Matlab function for creating our problem matrices.

```
1 function [ A, K, M ] = create_matrices( N )
2 %CREATE_MATRICES Summary of this function goes here
3 %   Detailed explanation goes here
```

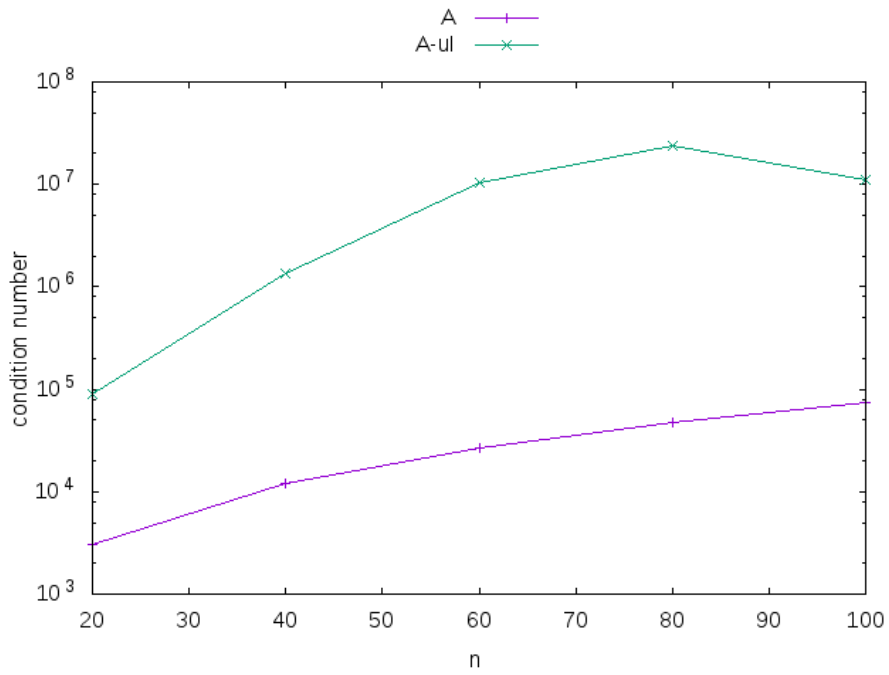

Figure 5: Condition number of the A and B matrices as a function of matrix size.

```
h = 1.0/N;

K = sparse(-diag(ones(N-1,1),-1)-diag(ones(N-1,1),+1)+diag(2*ones(N,1),0));
K(N,N) = 1;
K = K*N;

M = sparse(diag(ones(N-1,1),-1)+diag(ones(N-1,1),+1)+diag(4*ones(N,1),0));
M(N,N) = 2;
M = h/6*M;

A = M\K;

end
```

Listing 2: Matlab function for creating our problem matrices.

```
function [ eigenvalues, eigenvector ] = inverse_iteration( matrix, tolerance )
%INVERSE_ITERATION Summary of this function goes here
%first some basic checks on the input
assert(ismatrix(matrix));
assert(size(matrix,1) == size(matrix,2));

%if the tolerance isn't specified, set it to 1e-4
 if ~exist('tolerance','var')
      tolerance = 1e-4;
 end
 %LU-deocmpose
```

```
 [L, U] = lu(matrix);
 %preset the delta to always get 1 loop.
 eigen_delta = 2*tolerance;

 %first guess
 z = ones(size(matrix,1),1);
 y = z/norm(z);
 z = matrix*y;
 eigenvalues(1) = dot(y, z);

 i = 2;
 while eigen_delta > tolerance
    y = z/norm(z);
    z = U\(L\y);
    eigenvalues(i) = dot(y, z);
    eigen_delta = abs(eigenvalues(i-1)-eigenvalues(i));
    i = i +1;
 end
 eigenvector = z/norm(z);
end
```

Listing 3: Matlab function for computing the dominant eigenvalue of a matrix using the power method.

```
function [ eigenvalues, eigenvector ] = eig_power( matrix, tolerance  )
%POWER_ITERATION Computed the largest eigenvalue by power iteration
%returns an array of the computed eigenvalues

%first some basic checks on the input
assert(ismatrix(matrix));
assert(size(matrix,1) == size(matrix,2));

%if the tolerance isn't specified, set it to 1e-4
 if ~exist('tolerance','var')
      tolerance = 1e-4;
 end

 %preset the delta to always get 1 loop.
 eigen_delta = 2*tolerance;

 %first guess
 z = ones(size(matrix,1),1);
 y = z/norm(z);
 z = matrix*y;
 eigenvalues(1) = dot(y, z);

 i = 2;
 while eigen_delta > tolerance
```

```
    y = z/norm(z);
    z = matrix*y;
    eigenvalues(i) = dot(y, z);
    eigen_delta = abs(eigenvalues(i-1)-eigenvalues(i));
    i = i +1;
 end

 eigenvector = z/norm(z);

end
```

Listing 4: Matlab script used to generate the data.

```
fileID = fopen('../data/eigenvector.dat','w');
n=100
A = create_matrices(n);
[eigenvalues, eigenvector] = eig_power(A);
%shift so that the power method find sthe smallest
B = A-eigenvalues(end)*speye(n);
eigenvalues_min = eig_power(B, 1e-8)+eigenvalues(end);
%use inverse iteration to find the corresponding eigenvector.
[value_min, vector_min] = eig_power(inv(B), 1e-8);
fprintf(fileID, '#N= %u, lambda=%f\n', n, eigenvalues(end));
for i=1:n
    fprintf(fileID, '%f %f\n', eigenvector(i),vector_min(i));
end

fileID = fopen('../data/power_iterations.dat','w');
fprintf(fileID, '#N iterations error\n');
n= 20;
iteration = 1;
steps = [];
N = [];
while n <= 100
    A = create_matrices(n);
    cond_A = cond(A);
    real_ev = max(eig(full(A)));
    real_ev_min = min(eig(full(A)));
    eigenvalues = eig_power(A);
    B = A-eigenvalues(end)*speye(n);
    cond_B = cond(B);
    %be more accurate here
    eigenvalues_min = eig_power(B, 1e-8)+eigenvalues(end);
    %use inverse iteration to find the corresponding eigenvector.
    [iters, vector_min] = inverse_iteration(B, 1e-4);
    fprintf(fileID, '%u ', n);
    fprintf(fileID, '%u %f %f %f ', [size(eigenvalues,2) eigenvalues(end) real_ev cond_A
```

```
        ]);
    fprintf(fileID, '%u %f %f %f\n', [size(iters,2) eigenvalues_min(end) real_ev_min
        cond_B]);
    iteration = iteration + 1;
    n = n + 20;
end
fileID = fopen('../data/convergence.dat','w');
errors = (eigenvalues-real_ev)/real_ev;
errors_min = (eigenvalues_min-real_ev_min)/real_ev_min;
for x=1:100
    fprintf(fileID, '%f %f\n', [errors(x) errors_min(x)]);
end
```