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 4.

Power method

Implementation

The implementation of the power-method is given in listing 5.

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

Iterations vs. Matrix Size

Inverse iteration

Convergence

Note that the inverse iteration algorithm is equivalent to the power method over $A - \mu I$ instead of A, where μ is the eigenvalue of iterest. It is therefore trivial to deduce that the convergence rate is given by $\left|\bar{\lambda}_1/\bar{\lambda}_2\right|$ where $\bar{\lambda}_i$ are the original eigenvalues shifted by μ.

Implementation

The implementation of the the inverse iteration algorithm is given in listing 3.

results

iterations vs. matrix size

error vs. matrix size

Source Code

Listing 1: Matlab script used to generate the data for task 1.

```matlab
figure(1)
n= 16;
iteration = 1;
steps = [];
N = [];
while n<=512
    A = create_matrices(n);
    real_eigenvalue = max(eig(full(A)));
    eigenvalues = eig_power(A);
    steps(iteration) = size(eigenvalues,2)
    N(iteration) = n
    iteration = iteration + 1
    errors = abs(eigenvalues-real_eigenvalue);
    loglog(errors(1:end));
    hold on;
    n = n*2;
end
figure(2)
plot(N,steps)
```

Listing 2: Matlab script used to generate the data for task 2.

```matlab
figure(1)
n= 16;
iteration = 1;
steps = [];
N = [];
while n<=512
    A = create_matrices(n);
    real_max_eigenvalue = max(eig(full(A)));
    real_min_eigenvalue = min(eig(full(A)));
    eigenvalues_max = eig_power(A);
    B = A-eigenvalues_max(end)*eye(n);
    eigenvalues_min = eig_power(B)+eigenvalues_max(end);
    steps(iteration) = size(eigenvalues_max,2)
    N(iteration) = n;
    iteration = iteration + 1;
    errors_max = abs(eigenvalues_max-real_max_eigenvalue);
    errors_min = abs(eigenvalues_min-real_min_eigenvalue);
    loglog(errors_max(1:end));
    hold on;
    loglog(errors_min(1:end));
    n = n*2;
end
```

```
figure(2)
plot(N,steps)
```

Listing 3: Matlab function for computing the eigenvector corresponding to a specific eigenvalue using inverse iteration.

```
function [ eigenvalue ] = inverse_iteration( A, mu )
%INVERSE_ITERATION Summary of this function goes here
%   Detailed explanation goes here
    B = A-sparse(mu*diag(ones(size(A,1),1),0));
    eigenvalue = power_iteration(B)+mu;
end
```

Listing 4: Matlab function for creating our problem matrices.

```
function [ A, K, M ] = create_matrices( N )
%CREATE_MATRICES Summary of this function goes here
%   Detailed explanation goes here
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 5: 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;
```

```matlab

%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
```