Prof. Dr. Angela Kunoth
Sandra Boschert
Jose Licon

Numerische Mathematik
SoSe 2018

Übungsblatt 2

Ausgabe: Donnerstag, 19.4.2018
Abgabe: (Ausdruck Programmcodes) Mittwoch, 02.05.2018 bis 16 Uhr im Briefkasten vor dem Eingang Gyrhofstr. 8a.

(i) In this exercise sheet, $\|\cdot\|$ will represent the Euclidean norm, $\|\cdot\|_2$.

(ii) For all the functions you implement, take care to establish a break condition if convergence has not been achieved after 100 iterations.

(iii) Use a tolerance value of $\varepsilon =$ `1e-10`.

(iv) Take care to label all axes when plotting, and use figure titles. Scale your plots appropriately.

(v) The algorithms given here should all be implemented as Matlab functions which have as arguments the algorithm input variables, and return the algorithm outputs.

(vi) All functions should print out the following information at runtime:

Iteration no. k	Eigenvalue approximation λ_k (3 significant figures)	Error term err (10 significant figures)
1	1.23	3.293515666 e-01
2	1.24	3.293515667 e-01
⋮	⋮	⋮

Aufgabe 5: (4 Punkte)

Algorithm 1: Vector iteration (cf. Algorithmus 5.5.8 in lecture notes)

input : A matrix
$y^{(0)}$ starting vector with $\|y^{(0)}\| = 1$
ε tolerance
output: $(\lambda_k, y^{(k)})$

$k = 0$
do
  $k = k + 1$
  $\tilde{y}^{(k)} = A y^{(k-1)}$
  $\lambda_k = (\tilde{y}^{(k)})^T y^{(k-1)}$
  $y^{(k)} = \tilde{y}^{(k)} / \|\tilde{y}^{(k)}\|$
  $\text{err} = |A y^{(k)} - \lambda_k y^{(k)}|$
while err $> \varepsilon$

Implement the method as a Matlab function and save it in the file `vector_iteration.m`. Use your implementation to compute eigenvector-eigenvalue pairs for the following matrices:

$$A_1 = \begin{bmatrix} 4 & 5 \\ 6 & 5 \end{bmatrix}, \qquad A_2 = \begin{bmatrix} -4 & 10 \\ 7 & 5 \end{bmatrix}.$$

Use the vector $y^{(0)} := (1,3)^T / \|(1,3)^T\|$ as starting vector in both cases. Generate a plot of the two absolute error curves against iteration number labeling them $A1$ and $A2$ respectively, and save the figure as `vector_iteration.png`. Explain the difference in behaviour.

Aufgabe 6: (12 Punkte, 3 + 2 + 3 + 2 + 2)

Algorithm 2: Inverse vector iteration (cf. Algorithmus 5.6.2 in lecture notes)

input : A matrix
$y^{(0)}$ starting vector (normalized)
μ spectral shift
ε tolerance
output: $(\lambda_k, y^{(k)})$

Compute the QR factorization of $A-\mu I$: $QR = A-\mu I$
$k = 0$
do
  $k = k+1$
  $\tilde{y}^{(k)} = R^{-1}Q^{-1}y^{(k-1)}$
  $\omega_k = \tilde{y}^{(k)^T} y^{(k-1)}$
  $\lambda_k = \frac{1}{\omega_k} + \mu$
  $y^{(k)} = \tilde{y}^{(k)}/\|\tilde{y}^{(k)}\|$
  err $= |Ay^{(k)} - \lambda_k y^{(k)}|$
while err $> \varepsilon$

(a) Implement the algorithm as a Matlab function, saving it as `inverse_iteration.m`.

(b) Use your function implementation to compute eigenvalue-eigenvector pairs for the matrix

$$M_n = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{nxn}.$$

The initial vector is $v_n/\|v_n\|$, where $v_n := (1,-1,-1,\ldots,-1,-1)^T \in \mathbb{R}^n$, with the value $n = 1000$. Call the function twice, the first time using $\mu_1 = 1$ as spectral shift, $\mu_2 = 5$ the second time. Plot both absolute error curves (in logarithmic scale) against iteration number, and save it as `inverse_iteration.png`. The curves should be labeled μ_1 and μ_2.

(c) Recall the Sturm-Liouville problem (cf. lecture notes, Beispiel 5.5.9), which is discretized as

$$Au = \lambda Ru,$$

where the matrix A satisfies $A = M_n/h^2$, with $h = (n+1)^{-1}$. By setting $r(x) = 1$ constant, we obtain $R = I$, which produces an eigenvector equation for A with the eigenvalues given by

$$\lambda_{n+1-j} = \frac{4}{h^2}\sin^2\left(\frac{j\pi h}{2}\right), \qquad j = 1,\ldots n.$$

We assume that $\lambda_1 > \lambda_2 > \ldots > \lambda_n$. We can use the fact that the eigenvalues are known in this case to monitor the performance of our approximation algorithms: modify your `vector_iteration` and `inverse_iteration` functions to admit an optional argument, `known_ev`, which if present is the known eigenvalue λ^* we are approximating, and is used to print out an additional column of information containing the quantity $|\lambda^* - \lambda_k|$. A way to implement this is for example:

```
vector_iteration(A, y0, tol, known_ev):
    % If argument is not present, set it to false
    if ~exist('known_ev', 'var')
        known_ev = false
    end

    % Other function code

    if ~known_ev
        % Old print statement
    else
        % New print statement
    end
```

[continues on following page ...]

Aufgabe 6 (cont.)

(d) Apply the vector iteration method to the Sturm-Liouville problem using a step size of $h = 1/30$, starting vector $x^{(0)}/\|x^{(0)}\|$ with $x^{(0)} = (1, 2, \ldots, 29)^T$, and the dominant eigenvalue λ_1 as the known eigenvalue. Do the same using now a step size of $h = 10^{-4}$.

(e) Use the inverse vector iteration to approximate the smallest eigenvalue of the Sturm-Liouville problem, namely λ_n, by choosing a spectral shift of $\mu = 0$, for the same step sizes of $h = 1/30, 10^{-4}$. Compare with the results from (d): how are they different, and what could explain this difference?

Aufgabe 7: (12 Punkte, 4 + 4 + 4)
If the matrix we are dealing with is symmetric or Hermitian, we can use a different strategy. We have the following result:

Lemma. *For a nonzero vector q , the number ρ that minimizes $\|Aq - \rho q\|$ is the* Rayleigh quotient*:*

$$\rho = \frac{q^T A q}{q^T q}.$$

If we use this quotient instead of the spectral shift we obtain the following algorithm:

Algorithm 3: Rayleigh quotient iteration

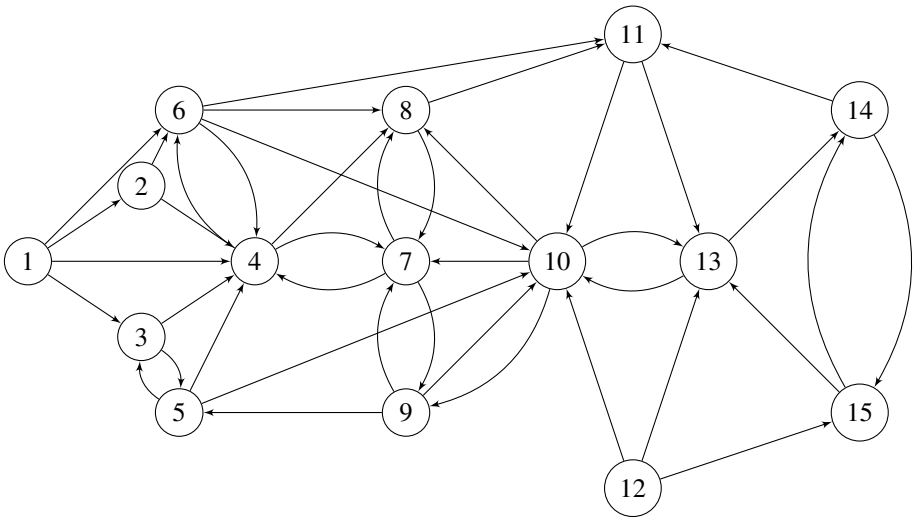
input : A matrix
$y^{(0)}$ starting vector (normalized)
ε tolerance
output: $(\rho_k, y^{(k)})$

k = 0
do
  $k = k + 1$
  $\rho_k = y^{(k-1)^T} A y^{(k-1)}$ // The eigenvalue approximation at the current step is given by ρ_k
  Solve $(A - \rho_k I)\tilde{y}^{(k)} = y^{(k-1)}$ for $\tilde{y}^{(k)}$ // Use the Matlab backslash operator to solve this system.
  $y^{(k)} = \tilde{y}^{(k)}/\|\tilde{y}^{(k)}\|$
  err $= |A y^{(k)} - \rho_k y^{(k)}|$
while err $> \varepsilon$

(a) Prove the lemma.

(b) Implement this as a Matlab function and save it in the file `rayleigh_iteration.m`. Use this function to find an eigenvector-eigenvalue pair for the matrix M_n from Aufgabe 6, with the same initial vector v_n for $n = 1000$. Plot the absolute error curve against iteration number, and save it in `rayleigh_iteration.png`. What can you say about the convergence rate of this method?

(c) We will now compare the performance of both the inverse iteration and the Rayleigh quotient iteration methods. For each value of $n = 10, 50, 100, 500, 1000, 5000, 10000$: generate the corresponding matrix M_n and vector v_n, and measure the runtime for each of the function calls `inverse_iteration` for $A = M_n$, $y^{(0)} = v_n$, $\mu = 5$, $\varepsilon =$ `1e-10`, and `rayleigh_iteration` for $A = M_n$, $y^{(0)} = v_n$, $\varepsilon =$ `1e-10`. Plot both sets of runtimes as a function of matrix size, labeling the curves 'inverse' and 'rayleigh', and save the file as `runtime_scaling.png`. What causes the difference in runtime? Explain your reasoning.



Aufgabe 8: (12 Punkte, 5 + 3 + 4)
As an application of the power iteration method we will now consider the idea behind Google's *PageRank* algorithm [1]. A network of interconnected websites can be represented by a directed graph such as the one shown above, where the set of nodes V represents the websites, and the set of edges E represents the hyperlinks between them. We seek to measure the importance of each website based on how many websites point to it. Concretely, suppose that each node i has a total of ℓ_i outgoing edges. Let $B_i = \{j \in V \mid j \to i\}$ denote the set of nodes j that link to i. Then we can express the importance (or ranking) of site i as

$$r(i) = \sum_{j \in B_i} \frac{r(j)}{\ell_j}.$$

The key idea is that by defining a hyperlink matrix H as

$$H_{i,j} = \begin{cases} 1/\ell_j & \text{if } j \in B_i \\ 0 & \text{otherwise} \end{cases}$$

then the vector r having as entries the ranking for each node i, $r := (r(1), r(2), \ldots, r(n))^T$, must satisfy the equation $r = Hr$. That is, r is an eigenvector of H with eigenvalue 1. We can therefore use the vector iteration algorithm to calculate it.

(a) Create the hyperlink matrix for the above network, $H \in \mathbb{R}^{15x15}$. Keep in mind the Matlab function `sparse` for defining matrices of this kind.

(b) Use your `vector_iteration` function from Aufgabe 5 to approximate the eigenvector r, with $y^{(0)} := (1, 0, 0, \ldots, 0)^T$ as starting vector.

(c) Report your resulting approximation to r. Which nodes receive the highest ranking, and which the lowest? Interpret the results.

Important: When writing computer code, it is very important to also write comments to explain what each step does. Therefore, submitting program files with no comments in them might result in points being deducted. Collect all your program files, plots, and a text or pdf file with your question answers in one directory; compress it into the file `yourname_uebung2.zip` and send this (and only this) to your corresponding tutor by email. You can find their email addresses in our website [2]. Hand in a printout of your code, function outputs, plots, and question answers as well.

Geben Sie einen Ausdruck des Programmdurchlaufs und des vorbildlich kommentieren, perfekt dargestellten m-files ab. Schicken Sie Ihren Programmcode zusätzlich per Email an Ihren jeweiligen Übungsgruppenleiter und zwar mit Subject/Betreff à la:

Subject: Uebung2, Muster, Hans

Bitte erstellen Sie hierzu einen ZIP komprimierten Ordner mit den m-files. Die Email-Adressen der jeweiligen Übungsgruppenleiter finden Sie auf der Veranstaltungshomepage.

[1] You can find a good overview of the method here: `https://www.ams.org/publicoutreach/feature-column/fcarc-pagerank`.
[2] `http://www.numana.uni-koeln.de/16890.html`