MATH 612
Computational methods for equation solving and function minimization – Week # 1

Instructor: Francisco-Javier ‘Pancho’ Sayas

Spring 2014 – University of Delaware

Have a look at the syllabus

- Attendance is expected and controlled
- Evaluation: exams, problems (from the book – the difficult ones), and code
- Office hours: feel free to contact me at any time. Send an email in advance. Don't just show up.
- My mailbox is off-limits!

Refer to the website for any doubt on calendar, rules, etc. All documents will be duly posted there.

Coding

- For the next week, find you coding buddy/BFF. (You'll be working with them all semester long.) In a team, let the typing be done by the person who is less confident with matlab.
- Repeat, repeat, repeat. Hit the wall. You learn coding by making mistakes. You think you know how to do it. You might not! Do it!
- My code doesn't work is not an acceptable question.
- Your code has to be impeccable. (Dirty code will not be accepted.)

If you don't know how to code... start right now! Today, not tomorrow! Download any guide to Matlab and start typing up all the examples. Google *A beginner's guide to matlab*

The book

The text-book (Trefethen-Bau *Numerical Linear Algebra*) covers the first 2/3 of the semester. I am not going to explain from the book:

- You are expected to start reading the book right away. By next Monday, I'll have assumed you have read Lectures 1 and 2.
- Work out all the problems, especially the difficult ones.
- I'll give hints and general explanations of ideas that can be read in the book. I'll help with some of the problems.

I will prepare slides for some lectures. They will be posted a couple of days later.

WARMUP ALGORITHM

An algorithm: it doesn't matter what it does

Input:

- a matrix $A \in \mathbb{R}^{n \times n}$
- a column vector $x_0 \in \mathbb{R}^n$,
- two parameters: tol and itMax

Process: compute the sequence

$$x_{n+1} = \frac{1}{\|Ax_n\|} Ax_n \qquad \lambda_{n+1} = x_{n+1} \cdot Ax_{n+1}$$

Stopping criterion: $|\lambda_{n+1} - \lambda_n| \leq \text{tol} \times |\lambda_{n+1}|$.
Safety criterion: in case the stopping criterion is not reached, stop when $n >$ itMax

Just one mathematical idea

If

$$Ax = \lambda x$$

then

$$x \cdot Ax = \lambda \|x\|^2.$$

Therefore, if

$$Ax = \lambda x \qquad \text{and} \qquad \|x\| = 1,$$

then

$$x \cdot Ax = \lambda.$$

Optimizing the iterations

Instead of ...

$$x_{n+1} = \frac{1}{\|Ax_n\|} Ax_n \qquad \lambda_{n+1} = x_{n+1} \cdot Ax_{n+1}$$

do

$$y_n = Ax_n, \qquad x_{n+1} = \frac{1}{\|y_n\|} y_n \qquad \lambda_{n+1} = x_{n+1} \cdot Ax_{n+1}.$$

We have reduced the numer of matrix $\times$ vector multiplications from 3 to 2 (in each iteration).

Optimizing ... (one multiplication per iteration)

Instead of ...

$$y_n = Ax_n, \qquad x_{n+1} = \frac{1}{\|y_{n+1}\|} y_{n+1} \qquad \lambda_{n+1} = x_{n+1} \cdot Ax_{n+1}.$$

do ...

$$x_{n+1} = \frac{1}{\|y_n\|} y_n, \qquad y_{n+1} = Ax_{n+1}, \qquad \lambda_{n+1} = x_{n+1} \cdot y_{n+1}.$$

Note that now we need to precompute $y_0 = Ax_0$ before starting the iterations.

Pseudocode (still in algorithmic form)

Input: A, x_0, tol, itMax.

$y_0 = Ax_0$
for $n = 0$: itMax $- 1$
    $x_{n+1} = (1/\|y_n\|)\, y_n$
    $y_{n+1} = Ax_{n+1}$
    $\lambda_{n+1} = x_{n+1} \cdot y_{n+1}$
    if $|\lambda_{n+1} - \lambda_n| \le$ tol$|\lambda_{n+1}|$
        we are ready to leave the program
    end
end
if we got here we didn't converge

Output: λ_{n+1}, x_{n+1} (at the moment of convergence)
Warning. There's a problem in the first step. Can you see which?

Pseudocode (ready to code)

Input: A, $x0$, tol, itMax.

$y = Ax0$
$\lambda_{\text{old}} = \infty$ % take advantage of Matlab's infinity
for $n = 1$: itMax % we have shifted the counter
    $x = (1/\|y\|)\, y$
    $y = Ax$
    $\lambda_{\text{new}} = x \cdot y$
    if $|\lambda_{\text{new}} - \lambda_{\text{old}}| \leq \text{tol}|\lambda_{\text{new}}|$
        leave the program with λ_{new} and x
    end
    $\lambda_{\text{old}} = \lambda_{\text{new}}$ % update
end
error message

Output: λ_{new}, x (at the moment of convergence)

Simple rules for efficient coding

We will distinguish between:

- functions: there's input and there's output; input is not output; output is not input; (true most of the time) – functions do specific well designed tasks
- scripts: everything is input and output – scripts are written for tests and final runs

Rules I will be imposing (code not following these rules will not be accepted)

- Code has to be indented (all loops and conditionals)
- Use the same names for variables in the algorithms description and in the code
- Write a function prototype explaining input and output at the beginning of the function

The code: part I

```
function [lnew,x]=powermethod(A,x0,tol,itMax)

% [lnew,x]=powermethod(A,x0,tol,itMax)
%
% Input:
%       A     : n x n matrix
%       x0    : column vector with n components
%       tol   : relative tolerance for stopping criterion
%       itMax: maximum number of iterations
% Output:
%       lnew : approximate eigenvalue
%       x     : approximate eigenvector (column vector)
%
% Last modified: February 11, 2014
```

The code: part II

```
y=A*x0;
lold=Inf;
for n=1:itMax
    x=(1/norm(y))*y;
    y=A*x;
    lnew=dot(x,y);
    if abs(lnew-lold)<tol*abs(lnew)
        return
    end
    lold=lnew;
end
display('Maximum number of iterations reached...
                                without convergence');
lnew=[];
x=[];
return
```

EXPERIMENTS AND THEORY

How to build diagonalizable matrices

To build an $n \times n$ real diagonalizable matrix, do as follows:

1. Create an invertible matrix P. Its columns will be the eigenvectors.
2. Create a diagonal matrix D. Its values will be the eigenvalues.
3. Mix them $A = P\,D\,P^{-1}$

Note that when we compute $Ax = P\,D\,P^{-1}\,x$, we first compute $c = P^{-1}\,x$, that is, we decompose

$$x = c_1\,p_1 + \ldots + c_n\,p_n \qquad p_j \text{ are the columns of } P,$$

and then we multiply the coefficients by the eigenvalues

$$Ax = \lambda_1\,c_1\,p_1 + \ldots + \lambda_n\,c_n\,p_n.$$

Complex eigenvalues

To produce the eigenvalues $\alpha \pm \imath\beta$, use the 2×2 block

$$\begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}$$

and make D a block diagonal matrix. In this case, the associated columns of P, say p_1 and p_2, are not the eigenvectors (they cannot be since they are real!). The eigenvectors are $p = p_1 \pm \imath p_2$.

To create non-diagonalizable matrices, use the same strategy substituting D by a Jordan form.

Dominant eigenvalues

Let A be a real $n \times n$ matrix. The set of all eigenvalues (real and complex) of A is called the **spectrum of A** and denoted $\sigma(A)$. We say that λ is a **dominant eigenvalue** of A if

$$|\lambda| \geq |\mu| \qquad \forall \mu \in \sigma(A).$$

Note that there are many possibilities for dominant eigenvalues:

- a single eigenvalue (with any multiplicity) separated from the other eigenvalues in absolute value
- two real eigenvalues with opposite signs, separated frm the rest
- two complex eigenvalues
- etc

When does the power method work?

Assume A satisfies:

- it has a unique dominant eigenvalue (which is real) λ
- for this dominant eigenvalue, the agebraic and geometric multiplicities coincide[1]

Then, with probability one in the choice of initial vector x_0, the power method satisfies:

$$\lambda_n \to \lambda \qquad \text{and} \qquad \begin{cases} x_n \to x_\infty & \text{if } \lambda > 0, \\ (-1)^n x_n \to x_\infty & \text{if } \lambda < 0, \end{cases}$$

where

$$A x_\infty = \lambda x_\infty \qquad \|x_\infty\| = 1.$$

[1] there are as many linearly independent eigenvectors for λ as the multiplicity in the characteristic polynomial

Some questions

- Are the hypotheses sharp? Yes, quite (see experiments)
- What does probablity one mean? With exact arithmetic, the probablily of the method not working, or going to a non-dominant eigenvalue is zero. But you might get very unlucky.
- Any idea on the speed? (see experiments)
- Can we verify the hypotheses in practical cases? Not really. We might have additional information on the matrix. Otherwise, it's mission impossible and we have to find other methods.
- How about other eigenvalues? (See inverse and shifted-inverse power methods)

A proof

Let us prove convergence of the power method for the particular case of diagonalizable matrices. Assume then that

$$\lambda_1 = \ldots = \lambda_k = \lambda \qquad |\lambda| > |\lambda_j| \quad j \geq k+1$$

are the eigenvalues of A with associated eigenvectors, real or complex, p_j. We decompose

$$x_0 = \underbrace{c_1 p_1 + \ldots c_k p_k}_{u_\infty} + c_{k+1} p_{k+1} + \ldots + c_n p_n.$$

With probability one, $u_\infty \neq 0$. Then

$$\begin{aligned} A^m x_0 &= \lambda^m u_\infty + \sum_{j=k+1}^{n} \lambda_j^m c_j p_j \\ &= \lambda^m \left(u_\infty + \sum_{j=k+1}^{n} \left(\frac{\lambda_j}{\lambda}\right)^m c_j p_j \right) = \lambda^m u_m. \end{aligned}$$

The proof continues

Following the algorithm of the power method, it is simple to see that

$$x_m = \frac{1}{\|A^m x_0\|} A^m x_0 = \frac{1}{\|u_m\|} u_m.$$

However, $u_m \to u_\infty$, which is an eigenvector associated to λ. Therefore

$$x_m \to x_\infty = (1/\|u_\infty\|) u_\infty,$$

where $Ax_\infty = \lambda x_\infty$, and

$$\lambda_m = x_m \cdot Ax_m \to x_\infty \cdot Ax_\infty = \lambda.$$

This proof shows how the eigenvector depends on the choice of x_0. It also shows that even if A is invertible, with probability one $x_m \neq 0$ for all m and the algorithm does not break down due to a division by zero.

Two (or three) observations, and one idea

1. If A is invertible, then
$$Ax = \lambda x \qquad \Longleftrightarrow \qquad A^{-1}x = \tfrac{1}{\lambda}\, x.$$
2. For any real μ,
$$Ax = \lambda x \qquad \Longleftrightarrow \qquad (A - \mu\, I)x = (\lambda - \mu)\, x.$$
3. Therefore, for real μ, if $A - \mu I$ is invertible, then
$$Ax = \lambda x \qquad \Longleftrightarrow \qquad (A - \mu\, I)^{-1}x = \frac{1}{\lambda - \mu}\, x.$$

The idea

To use the power method for $(A - \mu I)^{-1}$, you do not need to compute this matrix. In each iteration, you solve a linear system.

To do

- Test the power method for different types of matrices, relaxing each of the hypotheses of the convergence theorem.
- For diagonalizable matrices with a single dominant eigenvalue λ, the quantity

$$r := \frac{\max\{|\mu| \,:\, \mu \in \sigma(A), \quad \mu \neq \lambda\}}{|\lambda|}$$

  can be proved to be the rate of convergence of the method. Test it. To do that you will need to count iterations leading to convergence: modify the output of `powermethod`.
- Program the inverse-shifted power method. You just need two modifications in the algorithm:

$$y_{n+1} = (A - \nu I)^{-1} x_{n+1}, \qquad \lambda_{n+1} = \frac{1}{x_{n+1} \cdot y_{n+1}} + \mu.$$

A script for testing

```
P=[2 1 -3 0;...
   1 3 1 1;...
   2 2 1 -1;
   0 1 0 -2];  % eigenvectors by columns

D=diag([1 2 3 4]); % eigenvalues
A=P*D*inv(P);    % use inv only in cases like this
x0=rand(4,1);    % uniform random in [0,1];
                 % randn would be Gaussian random

[lamb,x]=powermethod(A,x0,1e-7,100);
```

Try these examples

$$D = \begin{bmatrix} 2 & & & \\ & -3 & & \\ & & 1/2 & \\ & & & 1/4 \end{bmatrix} \qquad D = \begin{bmatrix} 2 & & & \\ & 2 & & \\ & & 1/2 & \\ & & & 1/4 \end{bmatrix}$$

$$D = \begin{bmatrix} 2 & & & \\ & -2 & & \\ & & 1/2 & \\ & & & 1/4 \end{bmatrix} \qquad D = \begin{bmatrix} 2 & -1 & & \\ 1 & 2 & & \\ & & 1 & \\ & & & c \end{bmatrix} \quad c = 1, 3$$

$$D = \begin{bmatrix} 2 & 1 & & \\ & 2 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 2 & & & \\ & 2 & & \\ & & 1 & \\ & & 1 & 1 \end{bmatrix}$$

The shifted inverse power method

Input: A, $x0$, μ (shifting parameter), tol, itMax.

$B = A - \mu I$
$y = B^{-1}x0$ % solve the system $By = x_0$
$\lambda_{\text{old}} = \infty$
for $n = 1$: itMax
    $x = (1/\|y\|)\, y$
    $y = B^{-1}x$
    $\lambda_{\text{new}} = 1/(x \cdot y) + \mu$
    if $|\lambda_{\text{new}} - \lambda_{\text{old}}| \leq \text{tol}|\lambda_{\text{new}}|$
        leave the program with λ_{new} and x
    end
    $\lambda_{\text{old}} = \lambda_{\text{new}}$ % update
end
error message

Output: λ_{new}, x (at the moment of convergence)

Code

```
function [lnew,x]=inversepowermethod(A,x0,mu,tol,itMax)

% Prototype/help lines not shown

B=A-mu*eye(size(A,1));    % size(A,1)=# rows of A
y=B\x0;
lold=Inf;
for n=1:itMax
    x=(1/norm(y))*y;
    y=B\x;
    lnew=1/dot(x,y)+mu;
    if abs(lnew-lold)<tol*abs(lnew)
        return
    end
    lold=lnew;
end
display('Maximum number of its w/o convergence');
lnew=[]; x=[];
return
```

A TRANSLATION EXERCISE

The steepest descent method

Input: an $n \times n$ matrix A, two vectors b, x_0, tol, itMax
Process:

$$d_n = b - Ax_n; \qquad \delta_n = \frac{\|d_n\|^2}{d_n \cdot Ad_n} \qquad x_{n+1} = x_n + \delta_n x_n$$

Stopping criterion: $\|x_{n+1} - x_n\| \leq \mathrm{tol}\, \|x_{n+1}\|$
Safety check: stop after itMax iterations

Code this!

Some requirements

The format should be

```
[x,it]=SteepestDescent(A,b,x0,tol,itMax)
```

where `it` is the number of iteration at which the process is stopped. If we reach the maximum number of iterations, return the vector with a warning message. DO NOT FORGET THE HEADER (help lines/prototype)

What does this do? This is a quite bad method to solve iteratively a system

$$Ax = b$$

where A has to be symmetric and positive definite (this means symmetric with all eigenvalues positive)

How to test it

- Create an invertible matrix P
- Define $A = P^\top P$ (this matrix is symmetric and PD)
- Choose the solution x and compute the rhs $b = Ax$
- Start with $x_0 = 0$ or with $x_0 = b$ (they produce the same iteration, why?)

Another translation exercise

This is the Conjugate Gradient algorithm. We start with

$$r_0 = b - Ax_0 \qquad p_0 = r_0$$

and then iterate

$$\alpha_n = \frac{r_n \cdot r_n}{p_n \cdot Ap_n}, \qquad x_{n+1} = x_n + \alpha_n p_n, \qquad r_{n+1} = r_n - \alpha_n A p_n,$$

$$\beta_n = \frac{r_{n+1} \cdot r_{n+1}}{r_n \cdot r_n}, \qquad p_{n+1} = r_{n+1} + \beta_n p_n.$$

Stop when

$$|x_{n+1} - x_n| = |\alpha_n p_n| \leq \text{tol}|x_{n+1}|$$

or we have done more than itMax iterations.

Instruction for coding this

- Store Ap_n to avoid computing this matrix-vector product twice.
- Careful when updating r_n. The dot product of $r_n \times r_n$ is needed after r_{n+1} has been computed, so it has to be stored.
- For testing, use the same examples as in the Steepest Descent method. (The Conjugate Gradient method is much faster.)

END OF WEEK # 1