Norwegian University of Science and Technology
Department of Mathematical Sciences

MA2501 Numeriske Metoder
Olivier Verdier

Project

2011-02-23

Important Facts

- This project is mandatory, and accounts for 30% of the final grade.
- You may work in groups of at most two people
- You have to produce a report, of at most five pages, (preferably using LaTeX).
- The report should be sent electronically to olivier.verdier@math.ntnu.no at the latest on **Wednesday 30 March at 14:00**.
- At any moment before the end of the course (i.e., before 2011-04-13) you may ask me to *validate* your project (this is mandatory). We will do that orally during the exercise sessions (or outside if there is not enough time).

The project will be graded using the following criteria

- cleanliness and robustness of the code
- the code is expected to be tested (with short explanations)
- structure and presentation of the numerical experiments
- style of the report (should be short and to the point)

1. Orthogonalisation

The goal of this problem is to understand and implement an algorithm used in real life to compute the QR decomposition of matrices. The QR algorithm is the method of choice to solve a least-square algorithm. It is also the building block of a *fascinating* algorithm to compute eigenvectors and eigenvalues.

The orthogonalisation problem can be stated as follows:

> Given n vectors $\mathsf{c}_1, \ldots, \mathsf{c}_n$, form a matrix A which has these vectors as columns. Find an orthogonal matrix Q and an upper triangular matrix R such that $\mathsf{A} = \mathsf{QR}$. (The columns $\mathsf{q}_1, \ldots, \mathsf{q}_n$ of the matrix Q are the orthogonalized vectors).

The suggested algorithm is based on Householder transformations as elementary steps. A detailed description is given in Appendix B.

Problem 1

a) You will need to use the *reflection* operation – illustrated on Figure 2 – several times. Implement a function which computes the following operation, for two vectors a and v:

$$\mathsf{a} - 2\frac{(\mathsf{v}, \mathsf{a})}{\|\mathsf{v}\|^2}\mathsf{v} \tag{$\diamondsuit$}$$

where

$$\|\mathsf{v}\|$$

is the standard Euclidean norm stemming from the scalar product. The corresponding Python function is `norm`. Recall that the scalar product is computed using the function `dot`.

b) Modify the last function so that the vector a may be larger than v. The operation should work such that if a is larger (in number of components), compute ($\diamondsuit$) *as if* zeroes were added in the *beginning* of v in order to match the length of a.

You may use slicing to achieve this.

c) Create a function `householder` which takes a *matrix* A as argument, and returns a list of vectors v, to be interpreted as reflections, as well as the matrix on the right of ($\clubsuit$).

d) Create a function that computes the orthogonal matrix Q, from a list of reflection vectors.

e) Note that you now have a complete QR decomposition. Explain how you may now solve a least square problem by using the routines above, but *without* computing the matrix Q.

2. Poisson Problem

The distribution of temperature of a heated plate at equilibrium satisfies the following Poisson equation

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 1, \quad \text{in the interior}$$
$$u(x, y) = 0, \quad \text{on the boundary}$$

We will see later in the course how to transform this problem in discrete problems that are solvable on a computer. For the moment it is enough to know that such a problem may be discretised into a linear problem in the following manner:

$$\mathsf{A}x = b$$

where x represents values of u at discretized points, and f is the heating source. See Appendix A to see how to obtain those matrices in Python.

For any choice of shape or size, the matrix A enjoys two very important properties:

- A is *symmetric positive definite*, i.e., for any two vectors x, and y, we have

  $$(x, \mathsf{A}y) = (y, \mathsf{A}x)$$

  and

  $$(\mathsf{A}x, x) = 0 \implies x = 0.$$

- The matrix A is *sparse*: it is mostly filled with zeroes. More precisely, a row of the matrix contains at most five non-zero elements.

Problem 2

a) Describe the SOR algorithm. Find a theorem in the book that guarantees that the iteration will converge independently of the initial guess.

b) Give further arguments in favour of iteration methods: estimate in particular what is the cost of a matrix-vector multiplication (take into account the sparsity).

c) Implement the SOR algorithm, and test it on the system at hand. Try different shapes and sizes. You may also investigate how the intermediate solutions look like, compared to the exact solution.

d) Write a program that, for a given problem size, estimates numerically an optimal value for the parameter ω.

e) Change the problem size and describe how the optimal value of the parameter changes.

f) Implement the Jacobi method. Choose a problem size and shape. Assume that the Jacobi method was in fact implemented in parallel in an optimal way, i.e., that it is N times faster. How does that compete with the optimal SOR?

A. Creating the Matrix A

A.1. Producing a Grid

You may download a file in the course homepage (http://www.math.ntnu.no/emner/MA2501/2011v/proj.py) that contains routines to produce the matrices A corresponding to the problem above. The syntax is as follows:

```
G = numgrid(n,'S') # n: size of the problem; 'S': shape
```

The possible shapes are

S - the entire square.

L - the L-shaped domain made from 3/4 of the entire square.

C - like the L shape, but with a quarter circle in the 4-th square.

D - the unit disc.

A - an annulus.

You may visualize the domain with

```
spy(G>-1)
```

A.2. Producing the matrix A

The matrix A is then produced from the numerical grid as:

```
A = delsq(G)
```

You may visualize the non zero elements of the matrix with

```
spy(A)
```

A.3. Plotting the Results

Once you have a result u to the problem

$$\mathsf{A}u = b$$

you may plot it using

```
U = zeros_like(G)
U[G>-1] = u
contourf(U) # try also contour, or imshow
axis('equal') # for the square to look like a square
```

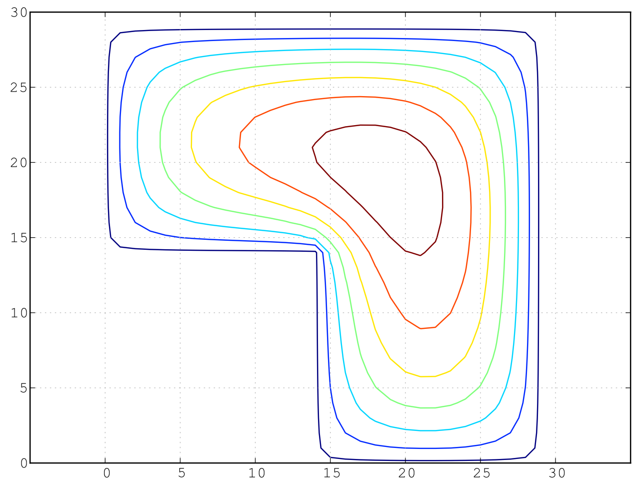

Figure 1: The result of `contour` on the L-shape.

B. Householder base algorithm and QR-factorization

The principal idea of of Householder reflections can be described geometrically:

Given a vector v, a reflection across $\mathrm{span}(\mathsf{v})^{\perp}$ is given by the (orthogonal) matrix

$$\mathsf{H} = \mathbb{I} - 2\frac{\mathsf{v}\mathsf{v}^{\mathsf{T}}}{\|\mathsf{v}\|^2}$$

You should check that $\mathsf{H}\mathsf{x} = \mathsf{x}$ for all $\mathsf{x} \in \mathrm{span}(\mathsf{v})^{\perp}$.

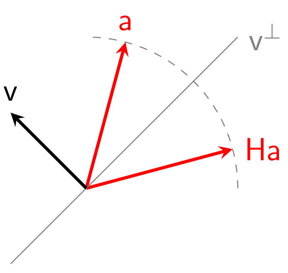

Figure 2: Reflection across $\mathsf{v}^\perp$

We select a vector v in such a way that it reflects a given vector a in such a way that:

$$\mathsf{Ha} = \mathsf{H}\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} \tilde{a}_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

This can be achieved by setting

$$\mathsf{v} = \mathsf{a} \pm \sigma \mathsf{e}_1 \qquad \sigma := \|\mathsf{a}\|.$$

It can easily checked that with this choice that $\mathsf{Ha} = \sigma\mathsf{e}_1$.

This special choice of v is illustrated in Figure 3.

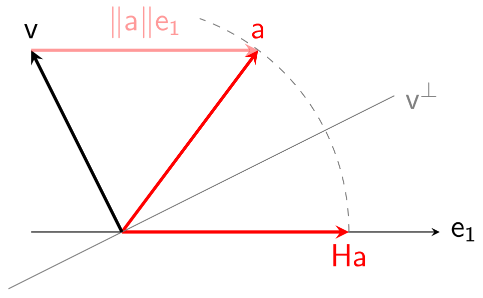

Figure 3: Householder transformation for annihilating entries in a vector: in this example, the resulting vector Ha has a non-zero entry only on the first axis spanned by e_1; v was chosen as $\mathsf{v} := \mathsf{a} - \|\mathsf{a}\|\mathsf{e}_1$

It is important to note that multiplications with Householder matrices can be done with $n+1$ multiplications and additions as

$$\mathsf{Ha} = \left(\mathbb{I} - \frac{\mathsf{v}\mathsf{v}^\mathsf{T}}{\gamma}\right)\mathsf{a} = \mathsf{a} - \frac{1}{\gamma}(\mathsf{v}, \mathsf{a})\mathsf{v},$$

with $\gamma := \frac{1}{2}\|\mathsf{v}\|^2$. (The standard matrix vector multiplications requires n^2 multiplications and additions.)

With this elementary Householder transformations a matrix can be transformed into a triangular matrix. To this end we apply $n-1$ Householder transformations $\mathsf{H}_1, ..., \mathsf{H}_{n-1}$ to A, where the i^{th} transformation introduces zeros in the i^{th} column, while leaving the columns $1, \ldots, i-1$ unaffected.

To demonstrate the process we assume that the first two columns are already transformed, i.e.,

$$\mathsf{H}_2\mathsf{H}_1\mathsf{A} = \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & \tilde{a}_{33} & x & x \\ 0 & 0 & \star & x & x \\ 0 & 0 & \star & x & x \end{bmatrix}. \tag{$\clubsuit$}$$

The goal for the third transformation is then to construct a Householder matrix $\tilde{\mathsf{H}}_3$ with

$$\tilde{\mathsf{H}}_3 \begin{bmatrix} \tilde{a}_{33} \\ \star \\ \star \end{bmatrix} = \begin{bmatrix} \bar{a}_{33} \\ 0 \\ 0 \end{bmatrix}.$$

Then we set

$$\mathsf{H}_3 = \begin{bmatrix} \mathbb{I} & 0 \\ 0 & \tilde{\mathsf{H}}_3 \end{bmatrix}$$

where we augment $\tilde{\mathsf{H}}_3$ by the identity matrix to keep the earlier columns of A unaffected.

If we set then $\mathsf{Q}^\mathsf{T} = \mathsf{H}_{n-1} \cdot \ldots \cdot \mathsf{H}_1$ we obtain $\mathsf{Q}^\mathsf{T}\mathsf{A} = \mathsf{R}$, where R is the desired triangular matrix and we thus got the decomposition $\mathsf{A} = \mathsf{QR}$.