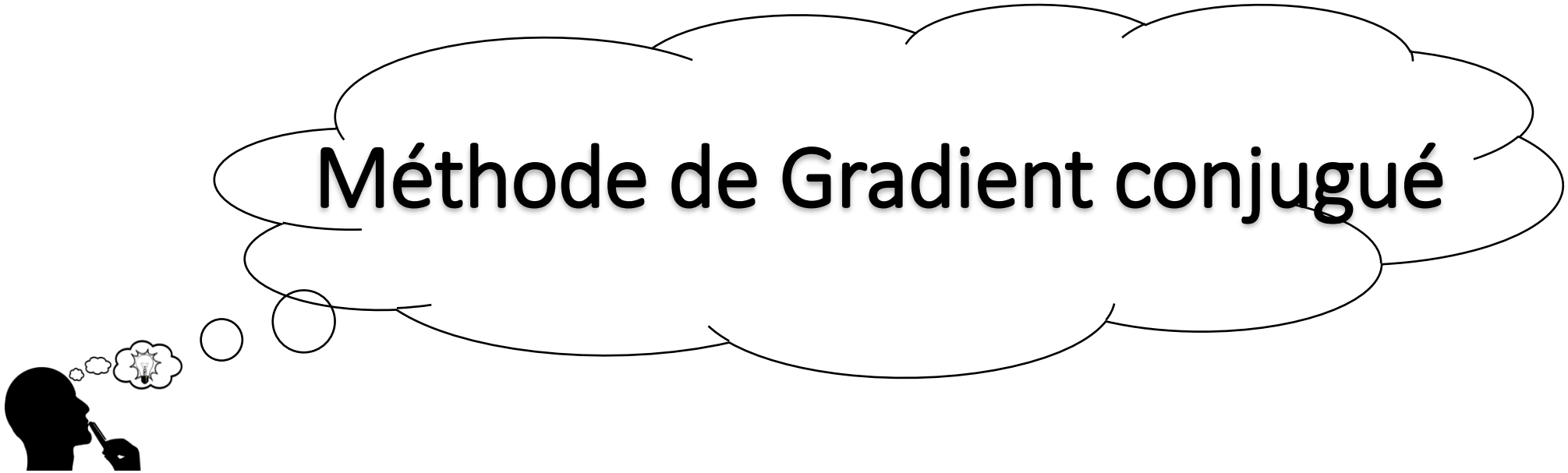
Calcul Scientifique

Ecole Hassania des Travaux Publics

Salem Nafiri

Méthode de Gradient conjugué



Notation

Let us begin with a few definitions and notes on notation.

With a few exceptions, I shall use capital letters to denote matrices, lower case letters to denote vectors, and Greek letters to denote scalars. A is an $n \times n$ matrix, and x and b are vectors — that is, $n \times 1$ matrices. Written out fully, Equation 1 is

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

The *inner product* of two vectors is written $x^T y$, and represents the scalar sum $\sum_{i=1}^{n} x_i y_i$. Note that $x^T y = y^T x$. If x and y are orthogonal, then $x^T y = 0$. In general, expressions that reduce to 1×1 matrices, such as $x^T y$ and $x^T A x$, are treated as scalar values.

A matrix A is *positive-definite* if, for every nonzero vector x,

$$x^T Ax > 0. \tag{2}$$

This may mean little to you, but don't feel bad; it's not a very intuitive idea, and it's hard to imagine how a matrix that is positive-definite might look differently from one that isn't. We will get a feeling for what positive-definiteness is about when we see how it affects the shape of quadratic forms.

Finally, don't forget the important basic identities $(AB)^T = B^T A^T$ and $(AB)^{-1} = B^{-1} A^{-1}$.

Quadratic Form

A *quadratic form* is simply a scalar, quadratic function of a vector with the form

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c \tag{3}$$

where A is a matrix, x and b are vectors, and c is a scalar constant. I shall show shortly that if A is symmetric and positive-definite, $f(x)$ is minimized by the solution to $Ax = b$.

Throughout this paper, I will demonstrate ideas with the simple sample problem

$$A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}, \qquad b = \begin{bmatrix} 2 \\ -8 \end{bmatrix}, \qquad c = 0. \tag{4}$$

The system $Ax = b$ is illustrated in Figure 1. In general, the solution x lies at the intersection point of n hyperplanes, each having dimension $n - 1$. For this problem, the solution is $x = [2, -2]^T$. The corresponding quadratic form $f(x)$ appears in Figure 2. A contour plot of $f(x)$ is illustrated in Figure 3.

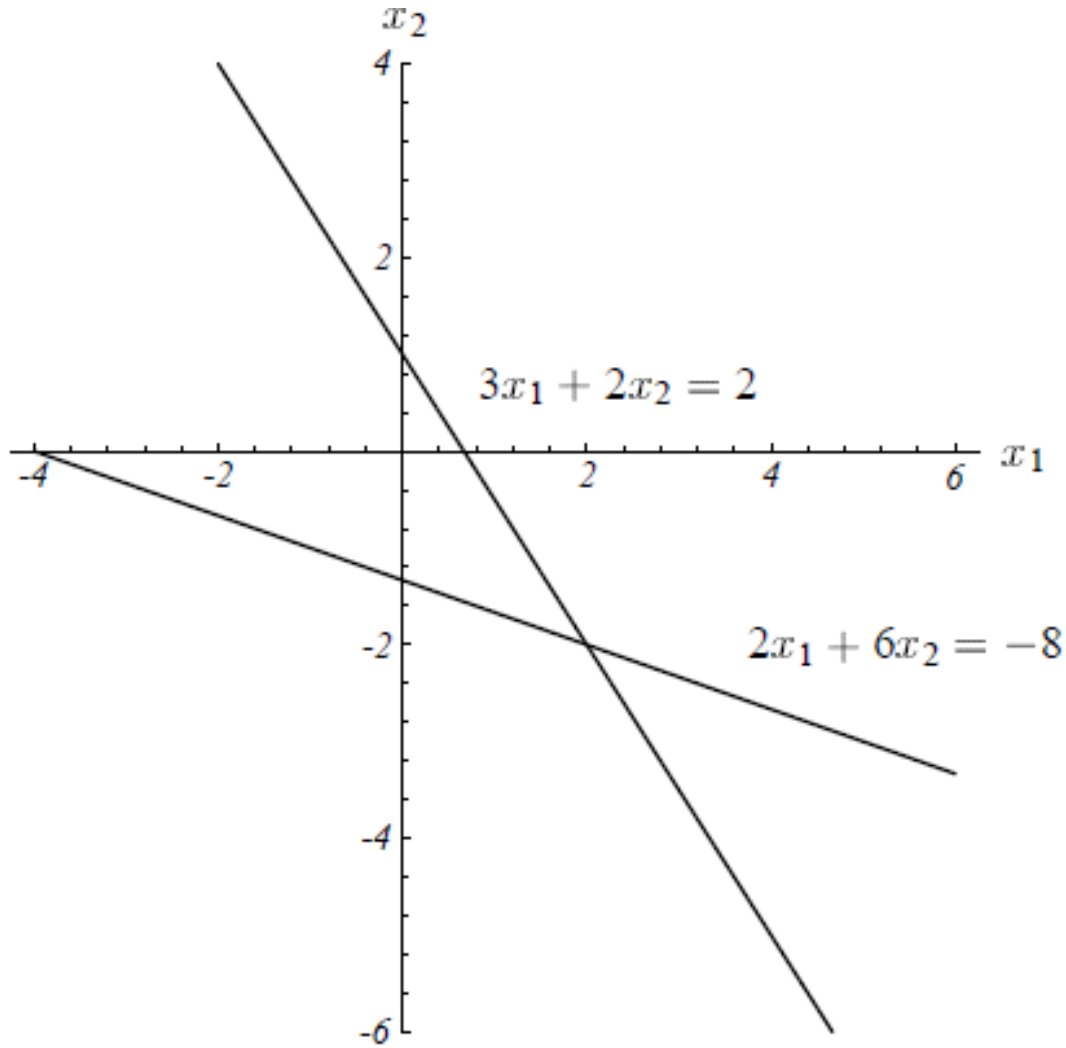


Figure 1: Sample two-dimensional linear system. The solution lies at the intersection of the lines.

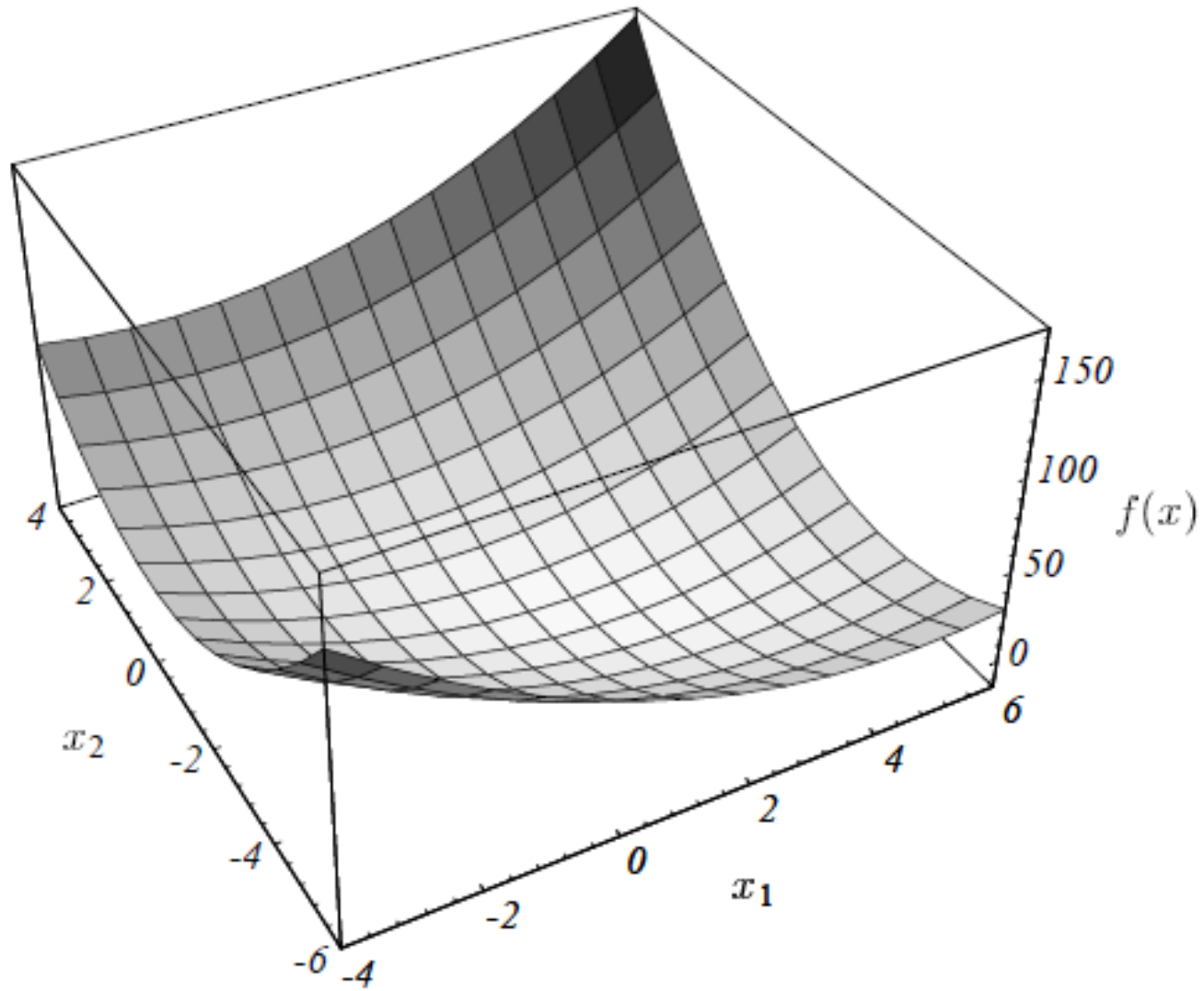


Figure 2: Graph of a quadratic form $f(x)$. The minimum point of this surface is the solution to $Ax = b$.

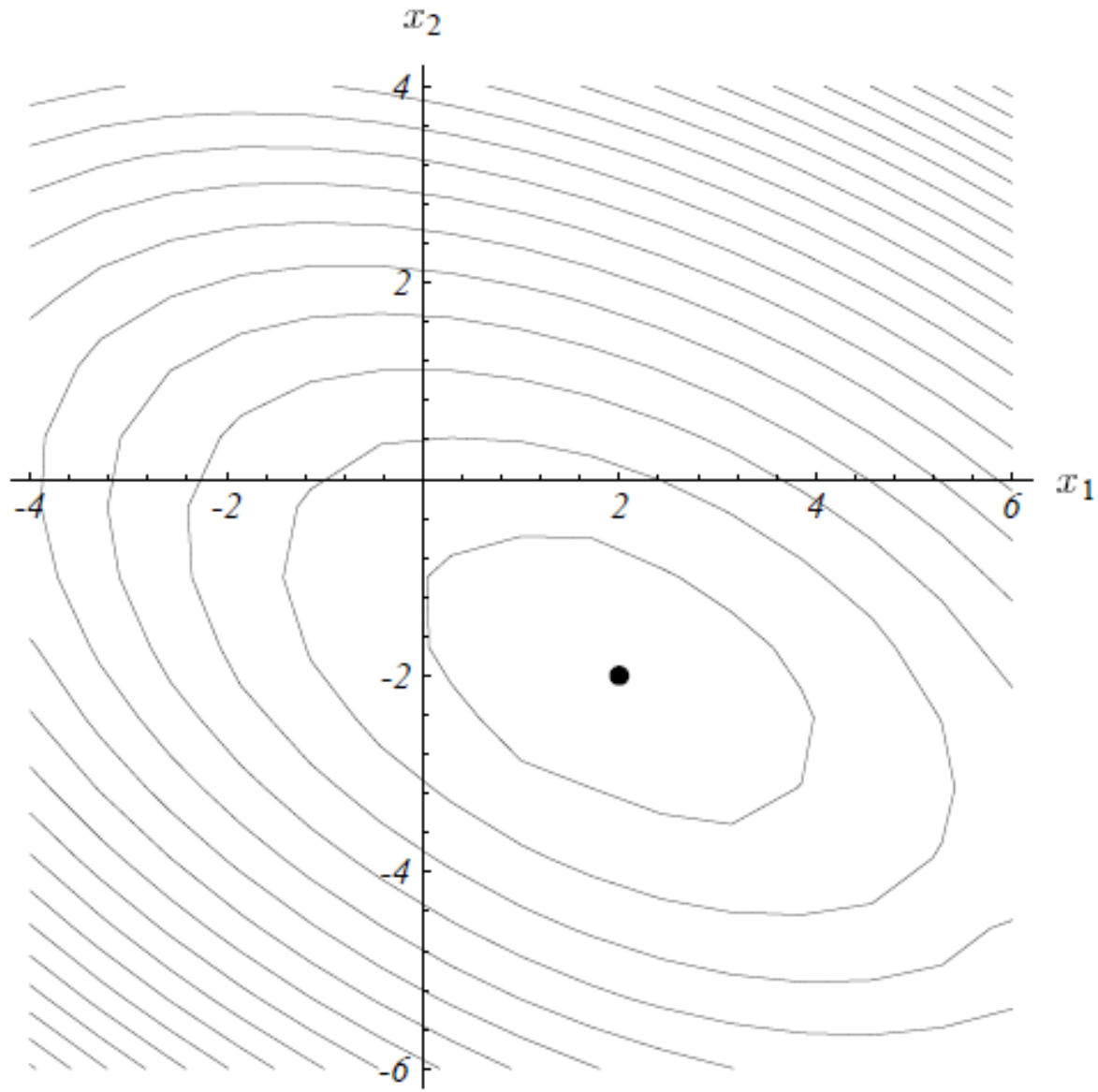


Figure 3: Contours of the quadratic form. Each ellipsoidal curve has constant $f(x)$.

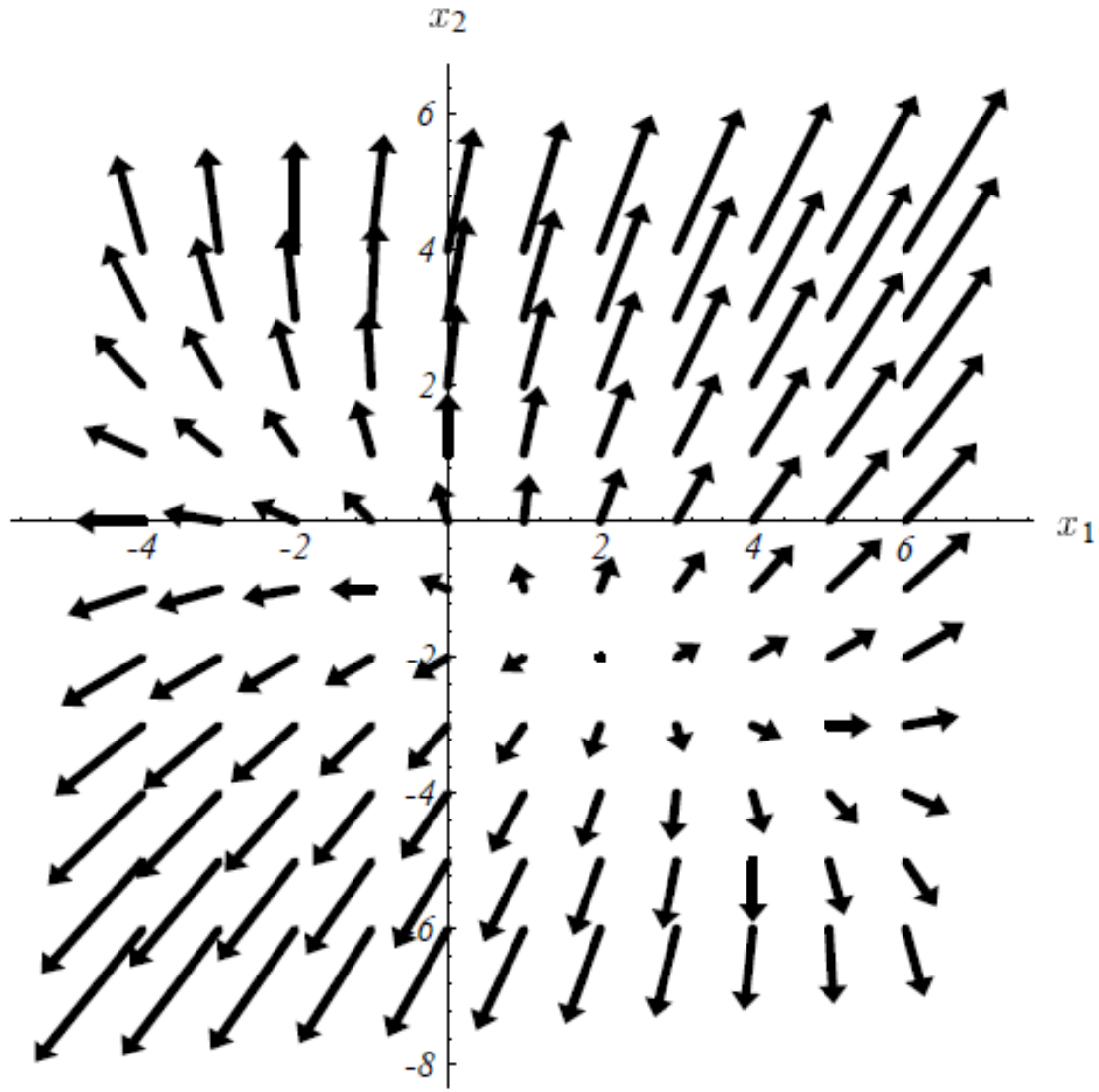


Figure 4: Gradient $f'(x)$ of the quadratic form. For every x, the gradient points in the direction of steepest increase of $f(x)$, and is orthogonal to the contour lines.

Because A is positive-definite, the surface defined by $f(x)$ is shaped like a paraboloid bowl. (I'll have more to say about this in a moment.)

The *gradient* of a quadratic form is defined to be

$$f'(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix}. \tag{5}$$

The gradient is a vector field that, for a given point x, points in the direction of greatest increase of $f(x)$. Figure 4 illustrates the gradient vectors for Equation 3 with the constants given in Equation 4. At the bottom of the paraboloid bowl, the gradient is zero. One can minimize $f(x)$ by setting $f'(x)$ equal to zero.

With a little bit of tedious math, one can apply Equation 5 to Equation 3, and derive

$$f'(x) = \frac{1}{2} A^T x + \frac{1}{2} Ax - b. \tag{6}$$

If A is symmetric, this equation reduces to

$$f'(x) = Ax - b. \tag{7}$$

Setting the gradient to zero, we obtain Equation 1, the linear system we wish to solve. Therefore, the solution to $Ax = b$ is a critical point of $f(x)$. If A is positive-definite as well as symmetric, then this

solution is a minimum of $f(x)$, so $Ax = b$ can be solved by finding an x that minimizes $f(x)$. (If A is not symmetric, then Equation 6 hints that CG will find a solution to the system $\frac{1}{2}(A^T + A)x = b$. Note that $\frac{1}{2}(A^T + A)$ is symmetric.)

Why do symmetric positive-definite matrices have this nice property? Consider the relationship between f at some arbitrary point p and at the solution point $x = A^{-1}b$. From Equation 3 one can show (Appendix C1) that if A is symmetric (be it positive-definite or not),

$$f(p) = f(x) + \frac{1}{2}(p - x)^T A(p - x). \tag{8}$$

If A is positive-definite as well, then by Inequality 2, the latter term is positive for all $p \neq x$. It follows that x is a global minimum of f.

Theorem. The vector $\boldsymbol{x}^*$ is a solution to the positive definite linear system $A\boldsymbol{x} = \boldsymbol{b}$ if and only if $\boldsymbol{x}^*$ produces the minimal value of $g(\boldsymbol{x}) = < \boldsymbol{x}, A\boldsymbol{x} > -2 < \boldsymbol{x}, \boldsymbol{b} >$.

Proof Let $\boldsymbol{x}$ and $\boldsymbol{v} \neq 0$ be fixed vectors and t a real number variable.

$$g(\boldsymbol{x} + t\boldsymbol{v}) = < \boldsymbol{x} + t\boldsymbol{v}, A(\boldsymbol{x} + t\boldsymbol{v}) > -2 < \boldsymbol{x} + t\boldsymbol{v}, \boldsymbol{b} >$$

$$= < \boldsymbol{x}, A\boldsymbol{x} > -2 < \boldsymbol{x}, b > +2t < \boldsymbol{v}, A\boldsymbol{x} > -2t < \boldsymbol{v}, \boldsymbol{b} > +t^2 < \boldsymbol{v}, A\boldsymbol{v} >$$

So $g(\boldsymbol{x} + t\boldsymbol{v}) = g(\boldsymbol{x}) - 2t < \boldsymbol{v}, \boldsymbol{b} - A\boldsymbol{x} > +t^2 < \boldsymbol{v}, A\boldsymbol{v} >$

Define $h(t) = g(\boldsymbol{x} + t\boldsymbol{v})$

Then $h(t)$ assumes a minimal value when $h'(t) = 0$.

$$h'(t) = -2 < \boldsymbol{v}, \boldsymbol{b} - A\boldsymbol{x} > +2t < \boldsymbol{v}, A\boldsymbol{v} >$$

The **minimum** occurs when $\hat{t} = \frac{<\boldsymbol{v}, \boldsymbol{b} - A\boldsymbol{x}>}{<\boldsymbol{v}, A\boldsymbol{v}>}$

$$h(\hat{t}) = g(\boldsymbol{x} + \hat{t}\boldsymbol{v}) = g(\boldsymbol{x}) - \frac{< \boldsymbol{v}, \boldsymbol{b} - A\boldsymbol{x} >^2}{< \boldsymbol{v}, A\boldsymbol{v} >}$$

For any vector $\boldsymbol{v} \neq \boldsymbol{0}$, we have $g(\boldsymbol{x} + \hat{t}\boldsymbol{v}) < g(\boldsymbol{x})$ unless $< \boldsymbol{v}, \boldsymbol{b} - A\boldsymbol{x} >= 0$.

Suppose $\boldsymbol{x}^*$ satisfies $A\boldsymbol{x}^* = \boldsymbol{b}$, then $< \boldsymbol{v}, \boldsymbol{b} - A\boldsymbol{x}^* >= 0$ for any $\boldsymbol{v}$. Thus $\boldsymbol{x}^*$ minimizes $g(\boldsymbol{x})$.

On the other hand, suppose that $\boldsymbol{x}^*$ is a vector minimizes $g(\boldsymbol{x})$. Then for any vector $\boldsymbol{v}$, $g(\boldsymbol{x}^* + \hat{t}\boldsymbol{v}) \geq g(\boldsymbol{x}^*)$. Thus $< \boldsymbol{v}, \boldsymbol{b} - A\boldsymbol{x}^* >= 0$. This implies that $\boldsymbol{b} - A\boldsymbol{x}^* = \boldsymbol{0}$.

Conjugate gradient

Let $A \in \mathbb{R}^{n \times n}$ be a large and sparse symmetric positive definite (s.p.d.) matrix. Consider the linear system

$$Ax = b$$

and the functional $F : \mathbb{R}^n \to \mathbb{R}$ with

$$F(x) = \frac{1}{2} x^T A x - b^T x. \tag{1}$$

Then it holds:

Theorem 1

For a vector x^* the following statements are equivalent:

$$(i) \quad F(x^*) < F(x), \text{ for all } x \neq x^*,$$

$$(ii) \quad Ax^* = b.$$

Proof: From assumption there exists $z_0 = A^{-1}b$ and $F(x)$ can be rewritten as

$$F(x) = \frac{1}{2}(x - z_0)^T A(x - z_0) - \frac{1}{2} z_0^T A z_0. \tag{2}$$

Since A is positive definite, $F(x)$ has a minimum at $x = z_0$ and only at $x = z_0$, it follows the assertion. ■

The solution of the linear system $Ax = b$ is equal to the solution of the minimization problem

$$\min F(x) \equiv \min \left(\frac{1}{2} x^T A x - b^T x \right).$$

The Method of Steepest Descent

a) Start with an arbitrary initial guess $\boldsymbol{x}^{(0)}$ to the solution $\boldsymbol{x}^*$ to $A\boldsymbol{x} = \boldsymbol{b}$

b) Let $\boldsymbol{v}^{(1)} = \boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{x}^{(0)}$.

Compute

$$t_1 = \frac{< \boldsymbol{v}^{(1)}, \boldsymbol{b} - A\boldsymbol{x}^{(0)} >}{< \boldsymbol{v}^{(1)}, A\boldsymbol{v}^{(1)} >}$$

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} + t_1 \boldsymbol{v}^{(1)}$$

Remark: the gradient of $g(\boldsymbol{x})$ is $\nabla g(\boldsymbol{x}) = 2(A\boldsymbol{x} - \boldsymbol{b}) = -2\boldsymbol{r}$. The direction of greatest decrease in the value of $g(\boldsymbol{x})$ is $-\nabla g(\boldsymbol{x})$.

c) $\boldsymbol{v}^{(2)} = \boldsymbol{r}^{(1)} = \boldsymbol{b} - A\boldsymbol{x}^{(1)}$

$$t_2 = \frac{< \boldsymbol{v}^{(2)}, \boldsymbol{b} - A\boldsymbol{x}^{(1)} >}{< \boldsymbol{v}^{(2)}, A\boldsymbol{v}^{(2)} >}$$

$$\boldsymbol{x}^{(2)} = \boldsymbol{x}^{(1)} + t_2 \boldsymbol{v}^{(2)}$$

d) Repeat the above process.

Remark: The Method of Steepest Descent does not lead to fastest convergence.

Definition: A set of nonzero vectors $\{\boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(n)}\}$ that satisfy $< \boldsymbol{v}^{(i)}, A\boldsymbol{v}^{(j)} > = 0$, if $i \neq j$ is said to be A-**orthogonal**.

Theorem. Let $\{\boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(n)}\}$ be an A-**orthogonal** set of nonzero vectors associated with the positive definite matrix A, and let $\boldsymbol{x}^{(0)}$ be arbitrary. Define

$$t_k = \frac{< \boldsymbol{v}^{(k)}, \boldsymbol{b} - A\boldsymbol{x}^{(k-1)} >}{< \boldsymbol{v}^{(k)}, A\boldsymbol{v}^{(k)} >}$$

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + t_1 \boldsymbol{v}^{(k)}$$

for $k = 1,2, \dots, n$. Then, assuming exact arithmetic, $A\boldsymbol{x}^{(n)} = b$.

Conjugate Gradient Method

Theorem. The residual vectors $\boldsymbol{r}^{(k)}$, where $k = 1,2,\dots,n$, for a conjugate direction method, satisfy the equation $< \boldsymbol{r}^{(k)}, \boldsymbol{v}^{(j)} >= 0$, for each $j = 1,2,\dots,k$.

Algorithm:

a) Start with an arbitrary initial guess $\boldsymbol{x}^{(0)}$ to the solution $\boldsymbol{x}^*$ to $A\boldsymbol{x} = \boldsymbol{b}$

Set

$$\boldsymbol{r}^{(0)} = \boldsymbol{b} - A\boldsymbol{x}^{(0)}$$

$$\boldsymbol{v}^{(1)} = \boldsymbol{r}^{(0)}$$

b) for $k = 1,2,3,\dots$

$$t_k = \frac{< \boldsymbol{r}^{(k-1)}, \boldsymbol{r}^{(k-1)} >}{< \boldsymbol{v}^{(k)}, A\boldsymbol{v}^{(k)} >}$$

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + t_k \boldsymbol{v}^{(k)}$$

$$\boldsymbol{r}^{(k)} = \boldsymbol{r}^{(k-1)} - t_k A\boldsymbol{v}^{(k)}$$

$$s_k = \frac{< \boldsymbol{r}^{(k)}, \boldsymbol{r}^{(k)} >}{< \boldsymbol{r}^{(k-1)}, \boldsymbol{r}^{(k-1)} >}$$

$$\boldsymbol{v}^{(k+1)} = \boldsymbol{r}^{(k)} + s_k \boldsymbol{v}^{(k)}$$

Numerical algorithm. In practice, when we apply the conjugate gradient algorithm, the formulas (13.11) of proposition 13.1.39 are programmed in the following way

$$\texttt{initialization} \quad \begin{cases} \texttt{initial choice } x_0 \\ r_0 = p_0 = b - Ax_0 \end{cases}$$

$$\texttt{iterations } k \geq 1 \quad \begin{cases} \alpha_{k-1} = \|r_{k-1}\|^2 / Ap_{k-1} \cdot p_{k-1} \\ x_k = x_{k-1} + \alpha_{k-1} p_{k-1} \\ r_k = r_{k-1} - \alpha_{k-1} Ap_{k-1} \\ \beta_{k-1} = \|r_k\|^2 / \|r_{k-1}\|^2 \\ p_k = r_k + \beta_{k-1} p_{k-1} \end{cases}$$

As soon as $r_k = 0$, the algorithm has converged, that is to say that x_k is the solution

Exercices

#1. Let $f(x,y) = 100(x^2 - y)^2 + (x-1)^2$ (this is called Rosenbrock's function).

Tracer cette fct avec Scilab

(a) What is the point of global minimum of f? (hint: the minimum value of f is 0).

(b) Find the formula for the gradient $\nabla f(x,y)$.

(c) Find $\nabla f(0.9, 0.9)$

(d) Find the angle between $-\nabla f(0.9, 0.9)$ and the vector going from $(0.9, 0.9)$ to the point of minimum at f. Note that the first is the direction of steepest descent, and the second is where we should really be going.

The algorithm

- Start with some x_0. Set $p_0 = r_0 = b - Ax_0$.
- For $k = 0, 1, 2, \ldots$
- $x_{k+1} = x_k + \alpha_k p_k, \quad \alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$
- $r_{k+1} = b - Ax_{k+1} = r_k - \alpha_k A p_k$
- $p_{k+1} = r_{k+1} + \beta_k p_k, \quad \beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$

Example

$\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

- Start with $\boldsymbol{x}_0 = 0$.
- $\boldsymbol{p}_0 = \boldsymbol{r}_0 = \boldsymbol{b} = [1, 0]^T$
- $\alpha_0 = \frac{\boldsymbol{r}_0^T \boldsymbol{r}_0}{\boldsymbol{p}_0^T \boldsymbol{A} \boldsymbol{p}_0} = \frac{1}{2}$, $\boldsymbol{x}_1 = \boldsymbol{x}_0 + \alpha_0 \boldsymbol{p}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix}$
- $\boldsymbol{r}_1 = \boldsymbol{r}_0 - \alpha_0 \boldsymbol{A} \boldsymbol{p}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \frac{1}{2} \begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1/2 \end{bmatrix}$, $\boldsymbol{r}_1^T \boldsymbol{r}_0 = 0$
- $\beta_0 = \frac{\boldsymbol{r}_1^T \boldsymbol{r}_1}{\boldsymbol{r}_0^T \boldsymbol{r}_0} = \frac{1}{4}$, $\boldsymbol{p}_1 = \boldsymbol{r}_1 + \beta_0 \boldsymbol{p}_0 = \begin{bmatrix} 0 \\ 1/2 \end{bmatrix} + \frac{1}{4} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/4 \\ 1/2 \end{bmatrix}$,
- $\alpha_1 = \frac{\boldsymbol{r}_1^T \boldsymbol{r}_1}{\boldsymbol{p}_1^T \boldsymbol{A} \boldsymbol{p}_1} = \frac{2}{3}$,

  $\boldsymbol{x}_2 = \boldsymbol{x}_1 + \alpha_1 \boldsymbol{p}_1 = \begin{bmatrix} 1/2 \\ 0 \end{bmatrix} + \frac{2}{3} \begin{bmatrix} 1/4 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 1/3 \end{bmatrix}$
- $\boldsymbol{r}_2 = 0$, exact solution.

```
function [x,numIter] = conjGrad(func,x,b,epsilon)
% Solves Ax = b by conjugate gradient method.
% USAGE: [x,numIter] = conjGrad(func,x,b,epsilon)
% INPUT:
% func    = handle of function that returns the vector A*v
% x       = starting solution vector
% b       = constant vector in A*x = b
% epsilon = error tolerance (default = 1.0e-9)
% OUTPUT:
% x       = solution vector
% numIter = number of iterations carried out

if nargin == 3; epsilon = 1.0e-9; end
n = length(b);
r = b - feval(func,x); s = r;
for numIter = 1:n
    u = feval(func,s);
    alpha = dot(s,r)/dot(s,u);
    x = x + alpha*s;
    r = b - feval(func,x);
    if sqrt(dot(r,r)) < epsilon
        return
    else
        beta = -dot(r,u)/dot(s,u);
        s = r + beta*s;
    end
end
error('Too many iterations')
```

CG Method: sample problem

- Sample problem:

$$\begin{bmatrix} 4 & -1 & 1 \\ -1 & 4 & -2 \\ 1 & -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ -1 \\ 5 \end{bmatrix}$$

- exact solution: $x_1 = 3$, $x_2 = x_3 = 1$.

Gram Schmidt orthogonalization

LEMME 1.21. *Soit* $(u_1, \ldots, u_k)$ *une famille orthogonale dans* E *et* $v \in E$. *Nous posons :*

$$u = v - \sum_{i=1}^{k} \frac{\langle u_i, v \rangle}{\|u_i\|^2} u_i.$$

Alors

(i) $u \perp u_1, \ldots, u_k$

(ii) $\mathbf{Vect}(u_1, \ldots, u_k, v) = \mathbf{Vect}(u_1, \ldots, u_k, u)$.

(iii) $u = 0$ *ssi* $v \in \mathbf{Vect}(u_1, \ldots, u_k)$.

THÉORÈME 1.22 (Orthogonalisation de Gram-Schmidt). *Soit* $(v_1, ..., v_m)$ *une famille libre dans* E. *On construit une suite de vecteurs* $u_1, \ldots, u_m$ *par :*

$$u_1 = v_1$$

$$u_{k+1} = v_{k+1} - \sum_{i=1}^{k} \frac{\langle u_i, v_{k+1} \rangle}{\|u_i\|^2} u_i.$$

(i) *La famille* $(u_1, \ldots, u_m)$ *ainsi construite est une base orthogonale pour* $\mathbf{Vect}(v_1, \ldots, v_m)$.

(ii) *Si on pose* $e_i = \frac{u_i}{\|u_i\|}$, *alors* $(e_1, \ldots, e_m)$ *est une base orthonormée pour* $\mathbf{Vect}(v_1, \ldots, v_m)$.

Variante : on peut normaliser $e_i = \frac{u_i}{\|u_i\|}$ pendant la construction et utiliser la formule

$$u_{k+1} = v_{k+1} - \sum_{i=1}^{k} \langle e_i, v_{k+1} \rangle e_i.$$

Bien que cette formule semble plus simple, le vecteurs normalisés e_i auront souvent une forme plus compliquée et le plus souvent on n'y gagne rien.

Gram Schmidt orthogonalization

Si la famille $(v_1, \ldots, v_m)$ n'est pas libre, on peut toujours appliquer le procédé de Gram-Schmidt. Pour chaque v_k qui est engendré par les vecteurs qui lui précèdent, on aura $u_k = 0$. Dans ce cas on peut enlever v_k de la famille et continuer avec le vecteur suivant.

COROLLAIRE 1.23. *Tout espace préhilbertien de dimension finie (= espace euclidien ou hermitien) admet un base orthonormée.*

Exercices

1. Implémenter ce procédé en créant une fonction `gramschmidt` prenant une matrice rectangulaire en entrée et, si la famille de vecteurs colonnes est libre, renvoyant une matrice de même taille avec les vecteurs orthonormaux issus du procédé ci-dessus. On veillera à tester si la matrice a au moins autant de lignes que de colonnes, et à afficher un message d'erreur si la famille n'est pas libre, c'est à dire si la norme d'un vecteur devient trop petite ($< 10^{-20}$ par exemple). Application : tester la procédure sur la matrice

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & -1 & 2 \\ 1 & 1 & 2 \end{pmatrix}$$

2. Cette méthode originelle est instable numériquement. Pour l'illustrer, considérer le cas

$$A = \begin{pmatrix} 1 & 1 & 1 \\ \varepsilon & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{pmatrix}$$

avec par exemple $\varepsilon = 10^{-10}$. Quel est le résultat exact ? Quel résultat donne votre procédure d'orthonormalisation ? Est-ce une famille orthogonale ? Cette anomalie vient du fait que du fait des erreurs d'arrondi, les vecteurs $\mathbf{u}_k$ ne sont pas orthogonaux. Cependant le procédé peut être stabilisé en l'implémentant de la manière suivante : Plutôt que de calculer $\mathbf{u}_k$ par $\mathbf{u}_k = \mathbf{v}_k - \text{proj}_{\mathbf{u}_1} \mathbf{v}_k - \text{proj}_{\mathbf{u}_2} \mathbf{v}_k - \cdots - \text{proj}_{\mathbf{u}_{k-1}} \mathbf{v}_k$, on le calcule par

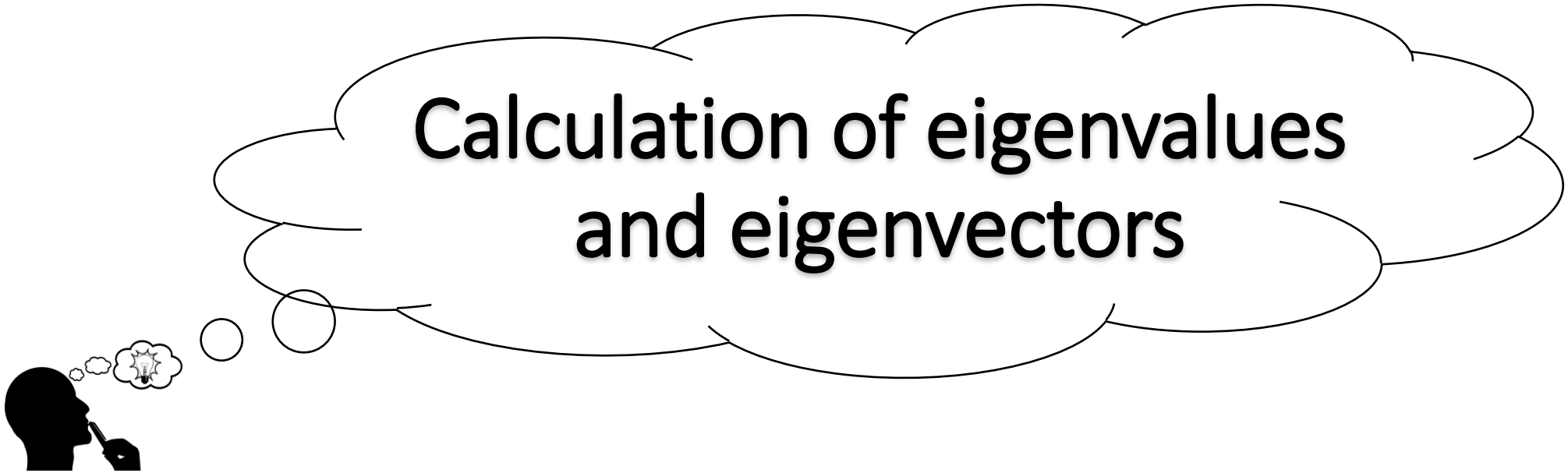
$$\begin{aligned} \mathbf{u}_k^{(1)} &= \mathbf{v}_k - \text{proj}_{\mathbf{u}_1} \mathbf{v}_k, \\ \mathbf{u}_k^{(2)} &= \mathbf{u}_k^{(1)} - \text{proj}_{\mathbf{u}_2} \mathbf{u}_k^{(1)}, \\ &\vdots \\ \mathbf{u}_k^{(k-2)} &= \mathbf{u}_k^{(k-3)} - \text{proj}_{\mathbf{u}_{k-2}} \mathbf{u}_k^{(k-3)}, \\ \mathbf{u}_k &= \mathbf{u}_k^{(k-2)} - \text{proj}_{\mathbf{u}_{k-1}} \mathbf{u}_k^{(k-2)}. \end{aligned}$$

Ce calcul donnerait le même résultat en arithmétique exacte mais est plus stable en arithmétique approchée.

Exercices

3. Implémenter ce second algorithme en modifiant très légèrement la fonction précédente et comparer le résultat qu'il fournit sur l'exemple ci-dessus. Etonnant, non ?

Calculation of eigenvalues and eigenvectors



Since the eigenvalues of a matrix A are the roots of its characteristic polynomial $\det(A - \lambda \mathrm{I})$. To calculate its eigenvalues, we could naively think that it is 'sufficient' to factorize its characteristic polynomial. It is nothing of the kind: we have known since Galois and Abel that we cannot calculate by elementary operations (addition, multiplication, extraction of roots) the roots of an arbitrary polynomial of degree greater than or equal to 5. To be convinced, we can notice that any polynomial of degree n,

$$P(\lambda) = (-1)^n \left(\lambda^n + a_1\lambda^{n-1} + a_2\lambda^{n-2} + \cdots + a_{n-1}\lambda + a_n\right),$$

is the characteristic polynomial (expanded with respect to the last column) of the matrix

$$A = \begin{pmatrix} -a_1 & -a_2 & \cdots & \cdots & -a_n \\ 1 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}.$$

Consequently, there cannot exist direct methods (that is to say which gives the result in a finite number of operations) for the calculation of eigenvalues! There therefore only exist iterative methods to calculate eigenvalues (and eigenvectors). We find that the practical calculation of eigenvalues and eigenvectors of a matrix is a much more difficult task than the solution of a linear system. Very fortunately, the case of real symmetric matrices (to which we limit ourselves as it is enough for our applications) is more simple than the case of non-self-adjoint matrices.

Some methods

- The power method
- The Givens–Householder method
- The Lanczos method

Some methods

- **The power method**
- The Givens–Householder method
- The Lanczos method

The power method

A very simple method to calculate the largest or the smallest (in modulus) eigenvalue of a matrix and an associated eigenvector is the power method. A limitation of the method is that the extreme eigenvalue that we calculate must be simple (or of multiplicity equal to 1, that is to say that the dimension of the corresponding eigensubspace is 1). Let A be a real symmetric matrix of order n, with eigenvalues $(\lambda_1, \ldots, \lambda_n)$ with $\lambda_n > |\lambda_i|$ for all $1 \leq i \leq n-1$. The power method to calculate the largest eigenvalue λ_n is defined by the algorithm below.

Numerical algorithm

1. Initialization: $x_0 \in \mathbb{R}^n$ such that $\|x_0\| = 1$.
2. Iterations: for $k \geq 1$
    1. $y_k = Ax_{k-1}$
    2. $x_k = y_k / \|y_k\|$
    3. convergence test: if $\|x_k - x_{k-1}\| \leq \varepsilon$, we stop.

In the convergence test ε is a small real number, typically equal to 10^{-6}. If $\delta_k = x_k - x_{k-1}$ is small, then x_k is an approximate eigenvector of A with approximate eigenvalue $\|y_k\|$ as $Ax_k - \|y_k\|x_k = A\delta_k$.

Proposition 13.2.1 *We assume that the matrix A is real symmetric, with eigenvalues $(\lambda_1, \ldots, \lambda_n)$, associated with an orthonormal basis of eigenvectors $(e_1, \ldots, e_n)$, and that the eigenvalue of largest modulus λ_n is simple and positive, that is to say that $|\lambda_1|, \ldots, |\lambda_{n-1}| < \lambda_n$. We assume also that the initial vector x_0 is not orthogonal to e_n. Then the power method converges, that is to say*

$$\lim_{k \to +\infty} \|y_k\| = \lambda_n, \qquad \lim_{k \to +\infty} x_k = x_\infty \quad \text{with } x_\infty = \pm e_n.$$

The rate of convergence is proportional to the ratio $|\lambda_{n-1}|/|\lambda_n|$

$$|\|y_k\| - \lambda_n| \leq C \left| \frac{\lambda_{n-1}}{\lambda_n} \right|^{2k}, \qquad \|x_k - x_\infty\| \leq C \left| \frac{\lambda_{n-1}}{\lambda_n} \right|^{k}.$$

Remark 13.2.2 The convergence of the sequence of approximate eigenvalues $\|y_k\|$ is more rapid than that of the approximate eigenvectors x_k (quadratic instead of linear). The power method also works for nonsymmetric matrices, but the convergence of $\|y_k\|$ is only linear in this case. •

In practice (and particularly for the calculation of eigenvalues from the discretization of an elliptic boundary value problem), we are above all interested in the **smallest** eigenvalue, in modulus, of A. We can adapt the preceding ideas, which gives the inverse power method whose algorithm is written below. We consider a real symmetric matrix A whose smallest eigenvalue in modulus is simple and strictly positive $0 < \lambda_1 < |\lambda_i|$ for all $2 \leq i \leq n$.

1. Initialization: $x_0 \in \mathbb{R}^n$ such that $\|x_0\| = 1$.
2. Iterations: for $k \geq 1$
    1. solve $Ay_k = x_{k-1}$
    2. $x_k = y_k / \|y_k\|$
    3. convergence test: if $\|x_k - x_{k-1}\| \leq \varepsilon$, we stop.

If $\delta_k = x_k - x_{k-1}$ is small, then x_{k-1} is an approximate eigenvector of the approximate eigenvalue $1/\|y_k\|$ as $Ax_{k-1} - x_{k-1}/\|y_k\| = -A\delta_k$.

Proposition 13.2.3 *We assume that the matrix A is real symmetric, with eigenvalues $(\lambda_1, \ldots, \lambda_n)$, associated with an orthonormal basis of eigenvectors $(e_1, \ldots, e_n)$, and that the eigenvalue of smallest modulus λ_1 is simple and strictly positive, that is to say $0 < \lambda_1 < |\lambda_2|, \cdots, |\lambda_n|$. We assume also that the initial vector x_0 is not orthogonal to e_1. Then the inverse power method converges, that is to say*

$$\lim_{k \to +\infty} \frac{1}{\|y_k\|} = |\lambda_1|, \qquad \lim_{k \to +\infty} x_k = x_\infty \quad \text{with } x_\infty = \pm e_1.$$

The rate of convergence is proportional to the ratio $\lambda_1/|\lambda_2|$

$$\left| \|y_k\|^{-1} - \lambda_1 \right| \leq C \left| \frac{\lambda_1}{\lambda_2} \right|^{2k}, \qquad \|x_k - x_\infty\| \leq C \left| \frac{\lambda_1}{\lambda_2} \right|^{k}.$$

Remark 13.2.4 The considerations of remark 13.2.2 also apply to the inverse power method. To accelerate the convergence, we can often translate the matrix A and replace it by $A - \sigma \mathrm{I}$ with σ an approximation of λ_1. •

TP Scilab

The power method

Example

Let's try this on the matrix $\begin{bmatrix} 7 & 4 & 1 \\ 4 & 4 & 4 \\ 1 & 4 & 7 \end{bmatrix}$ which has eigenvalues 12, 6 and 0 and

normalised eigenvectors $\frac{1}{\sqrt{3}}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$ and $\frac{1}{\sqrt{6}}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$.

Write a small MATLAB function:

```
function [x,lambda]=powermat1(A,x0,nit)
%  calculates the largest eigenvalue and corresponding eigenvector of
%  matrix A  by the power method using x0 as the starting vector and
%  carrying out nit interactions.
%

x = x0;

for n = 1:nit
   xnew = A*x;
   lambda = norm(xnew,inf)/norm(x,inf);
   fprintf('n = %4d  lambda = %g  x = %g %g %g \n', n, lambda, x');
   x=xnew;
end

x = x/norm(x); %normalise x
fprintf('n = %4d   normalised x = %g %g %g\n', n, x');

%end
```

Run it with initial guess $\mathbf{x}_0 = [1\ 2\ 3]^T$:

```
powermat1(A,[1 2 3]',10)
n =    1  lambda = 10  x = 1 2 3
n =    2  lambda = 10.8  x = 18 24 30
n =    3  lambda = 11.3333  x = 252 288 324
n =    4  lambda = 11.6471  x = 3240 3456 3672
n =    5  lambda = 11.8182  x = 40176 41472 42768
n =    6  lambda = 11.9077  x = 489888 497664 505440
n =    7  lambda = 11.9535  x = 5.92531e+006 5.97197e+006 6.01862e+006
n =    8  lambda = 11.9767  x = 7.13837e+007 7.16636e+007 7.19436e+007
n =    9  lambda = 11.9883  x = 8.58284e+008 8.59963e+008 8.61643e+008
n =   10  lambda = 11.9941  x = 1.03095e+010 1.03196e+010 1.03296e+010
n =   10   normalised x = 0.577068 0.57735 0.57763
```

You can see that $\mathbf{x}$ slowly "turns" towards the eigenvector and gets very close to $\lambda_1 = 12$ after 10 iterations.

Now try with starting vector $\mathbf{x}_0 = [0\ 1\ -1]^T$:

```
» powermat1(A,[0 1 -1]',10)
n =     1   lambda = 3   x = 0 1 -1
n =     2   lambda = 6   x = 3 0 -3
n =     3   lambda = 6   x = 18 0 -18
n =     4   lambda = 6   x = 108 0 -108
n =     5   lambda = 6   x = 648 0 -648
n =     6   lambda = 6   x = 3888 0 -3888
n =     7   lambda = 6   x = 23328 0 -23328
n =     8   lambda = 6   x = 139968 0 -139968
n =     9   lambda = 6   x = 839808 0 -839808
n =    10   lambda = 6   x = 5.03885e+006 0 -5.03885e+006
n =    10    normalised x = 0.707107 0 -0.707107
```

Note that it does not work in this case. We have found the second largest eigenvalue instead. The problem is that $\mathbf{x}^0$ is perpendicular to $\mathbf{u}_1$ and so $a_1 = 0$ in $\mathbf{x}^0 = a_1\mathbf{u}_1 + a_2\mathbf{u}_2 + \cdots + a_N\mathbf{u}_N$. So we must always have a component of $\mathbf{x}^0$ parallel to $\mathbf{u}_1$. A solution is to always try a couple of non parallel starting vectors.