Practical work on the descent method

The aim is to program in Java a descent method in the following particular case.

We consider a function f of two real variables and a domain D of the plane delimited by linear functions. We assume that f is convex and continuously differentiable on an open set containing D. We search the minimum of the function f on D.

The problem could be for example:

Minimize the function $f(x, y) = \exp(x + y) + x^2 + 2y^2$
in the domain defined by: $-2x + y \leq 0$
$-y \leq 0$
$x + y \leq 4$

It will be necessary to complete a Java class of a program.

The constraints of the problem are modeled as **negativity constraints**, in the form $ax + by + c \leq 0$. The gradient vector computed by the `getGradient()` method of the class `Constraint` is the vector (a, b).

If you want to see the result to achieve, download executableDescent.jar. You can then run the program with the command
`java -jar executableDescent.jar`.

To run the application, it is necessary to choose a problem by providing a number between 1 and 14; after validating the problem, the domain is displayed; you must then choose a starting point for the descent method inside the domain and then press the "Start" button. The descent method runs. During the method, for each point obtained during the method, the gradient vector is drawn in blue and the next direction to follow is in green. The trajectory generated by the descent method is drawn in red along the way.

You must start by saving the file descentToDo.jar on your computer.

After launching Eclipse, to create the project:

- in "File", do New then Java Project: this opens a window;
- in the "Project name" framework, put the name of your choice;
- if the "JRE" box in the window displays "jdk-11.0.19" to the right, go directly to the last instruction in this list;
- otherwise, do all of the following:
- in the JRE framework, select "Use a project specific JRE"; on the right, click on "Configure JREs..." (under the two small frames): this opens a window called "Preferences"
- in the "Preferences" window, click on the "Add" button: this opens a window called "Add JRE";
- in the "Add JRE" window, select "Standard VM" then click on "Next": this opens a new window called "Add JRE";
- in "JRE home", write /cal/softs/java/jdk-11.0.19/ then click on Finish: this returns to the "Preferences" window; then click on "Apply and close"; this returns to the "New Java Project" window;
- in the JRE framework, at the (selected) line "Use a project specific JRE", choose "jdk-11.0.19" using the black triangle pointed down;
- click on the "Finish" button at the bottom; this opens a "New module-info.java" window; in this window, choose **"Don't create" in order NOT TO CREATE A MODULE**.

If you work on your computer, maybe you must choose not to use any "module" in the page where the name of the project is given.

Then :

- click on src under the name of the project with the right button and choose "import";
- in the obtained window, in "General", choose "Archive File";
- after doing "Next", browse to choose descentToComplete.jar;
- click on finish.

You can try to run the method `main` which is in the class `Main` of the package `descent`.

The work to be done consists only of completing the class `Descent` of the package `descent.model`. In this class, six methods are to be completed (see the comments in the file to be completed for the description of these methods; these comments precede each of the methods to be completed): for the method `directionToFollowIfPOnCorner`, explanations are provided below on this page.

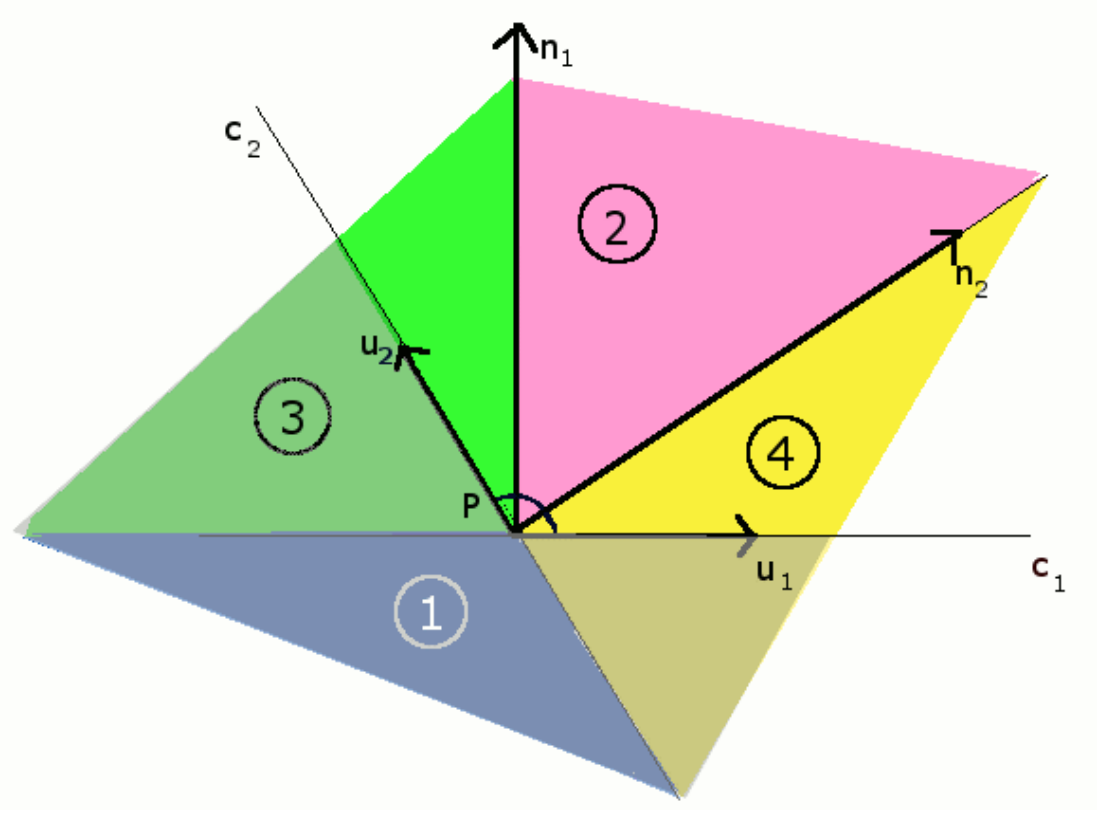
- the method `directionToFollowIfPInside`;
- the method `directionToFollowIfPOnEdge`;
- the method `directionToFollowIfPOnCorner`;
- the method `searchSecondPoint`;
- the method `dichotomy`;
- the method `KarushKuhnTucker`.

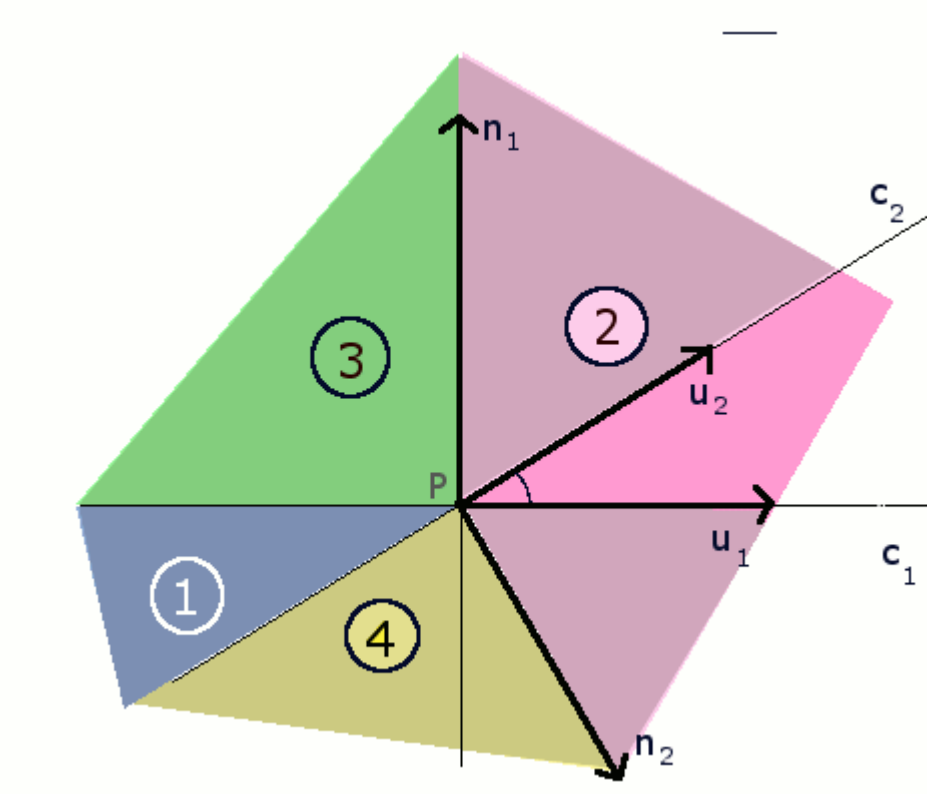
If the gradient norm of the studied function is lower than the attribute `threshold` of the class `descent`, the descent method is considered to be over; other cases of termination occur when the current point `P` is on an edge or on a corner of the domain.

It will be necessary to pay attention to the following attributes of the class `Descent`:

- the attribute `direction` must contain the direction to be followed by the descent method from the current point;
- the attribute `finished` should be set to `true` when you consider that the method is over.

Explanations for the function `directionToFollowIfPOnCorner`





In the drawings, two cases are represented: the case of an obtuse corner and the case of an acute corner. In both cases, the current point P is on the corner. The border lines of the constraints are called c_1 and c_2. The feasible domain lies between these two lines. The non-feasible domain is shaded.

The vector n_1 is equal to $-\nabla g_1(P)$ if the first constraint is $g_1(x, y) \leq 0$; the vector n_2 is equal to $-\nabla g_2(P)$ if the second constraint is $g_2(x, y) \leq 0$.

The vectors u_1 and u_2 are the unitary vectors of the borders and are located inside the feasible domain: **u_1 must be oriented towards the negative side of the constraint c_2** (it is necessary that the scalar product of

u_1 with n_2 is positive), **u_2 must be oriented towards the negative side of the constraint c_1** (it is necessary that the scalar product of u_2 with n_1 is positive).

Let ∇f be the gradient of f at the point P.

- *First case*: ∇f belongs to zone 1, in shaded blue. The steepest feasible direction is $-\nabla f$. The decomposition of ∇f with respect to the vectors u_1 and u_2 allows to detect this case.

- *Second case*: ∇f belongs to zone 2, in pink (shaded or not), the condition of Karush, Kuhn and Tucker is fulfilled. The decomposition of ∇f with respect to the vectors n_1 and n_2 allows to detect this case.

- *Third case*: we are not in the previous two cases and so ∇fbelongs to zone 3, green, or zone 4, yellow, shaded or not. In this case, the steepest feasible direction is given by one of the two vectors u_1 or u_2. Remind that the greater angle a direction d makes with ∇f (i.e. smaller the scalar product of d with ∇f is), the more d descends.

WARNING: do not change anything outside the Descent class.

You have a documentation on the project here.

The classes and methods useful for the work to be done are in particular the following:

- The class `descent.model.Descent` models the descent; it is in this class that the six methods to be completed can be found; explanations are in the file `Descent.java` to complete.

- The class `descent.model.Couple` models a couple of two elements of type `double`; it is used to model a point of the real plane or a vector of this same plane. The following methods in this class may be useful:

    - the method `double scalarProduct(Couple v)`: if `v1` and `v2` are two vectors of type `Couple`, then `v1.scalarProduct(v2)` gives the scalar product of `v1` with `v2`;

    - the method `double norm()`: if `v` is a vector of type `Couple`, then `v.norm()` gives the norm of `v`;

    - the method `boolean isPerpendicular(Couple v)`: if `v1` and `v2` are two vectors of type `Couple`, then `v1.isPerpendicular(v2)` is equal to `true` if `v1` and `v2` are perpendicular, and `false` otherwise;

    - the method `Couple mult(double t)`: if `v` is a vector of type `Couple` and if `t` is a `double`, then `v.mult(t)` return the vector of type `Couple` obtained by multiplying `v` by `t`; `v` is not modified by this multiplication;

    - the method `Couple add(Couple v)`: if `v1` and `v2` are two vectors of type `Couple`, then `v1.add(v2)` return the vector of type `Couple` obtained by adding `v1` to `v2`; `v1` is not modified by this addition;

    - the method `Couple decompose(Couple v, Couple v1, Couple v2)`: if `v`, `v1` and `v2` are three vectors of type `Couple`, then `decompose(v, v1, v2)` returns the two components of `v` on the basis formed by `v1` and `v2`; if `v1` and `v2` are parallel, the method return null.

    -

    -

- The class `descent.model.Constraint` models a linear constraint that is written: `coeffx * x + coeffy * y + constant <= 0`; this class also models a half-plane. The line of equation: `coeffx * x + coeffy * y + constant = 0` is the border line of the half-plane. The following methods in this class may be useful:
    - the method `Couple getGradient()`: if `c` is of type `Constraint`, then `c.getGradient()` returns the gradient of the function `(x, y) -> coeffx * x + coeffy * y + constant`, i.e. the vector of components `coeffx` and `coeffy;`
    - the method `Couple getUnitaryEdgeVector`: if `c` is of type `Constraint`, then `c.getUnitaryEdgeVector()` gives a unit vector of the border line.
- The class `descent.model.Domain` models a domain defined by a set of linear constraints. We suppose that there are not three constraints which intersect at the same point (otherwise the code might not work). The following methods in this class may be useful:
    - the method `Constraint isOnEdge(Couple P)`: if `D` is of type `Domain` and if `P` is a point of type `Couple`, then:
        - if `P` is on an edge of the domain, `D.isOnEdge(P)` returns a constraint (of type `Constraint`) corresponding to this edge; if `P` is on a corner of the domain, the method returns one of the two constraints corresponding to this corner;
        - if `P` is not on an edge of the domain, `D.isOnEdge(P)` returns `null`;
    - the method `Constraint[] isCorner(Couple P)`: if `D` is of type `Domain` and if `P` is a point of type `Couple`, then:
        - if `P` is on a corner of the domain, `D.isCorner(P)` returns the two constraints corresponding to this corner (in the form of an array of type `Constraint[]` with two components);
        - si `P` is not in a corner, `D.isCorner(P)` returns `null`.
- The abstract class `problem.Pb` models a problem, with the definition of the function `f` to be minimized and the set of constraints defining the domain. All problems are defined by classes inheriting from the class `problem.Pb`. The following methods in this class can be used:
    - the method `double gPrime(Couple P0, Couple d, double t)`: if `pb` is of type `Pb`, if $t \to P0 + td$ ($t > 0$) is the parametric equation of a half-line, then `pb.gPrime(P0, d, t)` returns the value of the derivative in `t` of the function `g: t -> g(t) = pb.f(P0 + td);`
    - the method `Couple gradientf(Couple P)`: if `pb` is of type `Pb`, if `P` is a point of type `Couple`, then `pb.gradientf(P)` returns the gradient of `f` at the point `P`.

When you have correctly implemented the method `directionToFollowIfPInside`, you will be able to see what happens with Problem 1 to see if it starts well; after implementing `directionToFollowIfPOnEdge`, you will be able to see what happens with Problem 1 to see if it goes well on one edge; after implementing `directionToFollowIfPOnCorner`, you will be able to check what happens with Problems 1, 4 and 6.
After implementing the method `searchSecondPoint`, you will be able to see what happens with Problem 3 (which does not reach a minimum in the domain).
After implementing the method `dichotomy`, you will be able to test all the problems.