Exercises in Convex Optimization
Lecture 3

Michel Baes, Patrick Cheridito

October 7, 2018

Lower semi-continuous functions Different equivalent definitions exist for the notion of lower semi-continuity of a function. In the present exercise, you can use the definition given in the lecture, that is: a function is lower semi-continuous iff its epigraph is closed.

1. Show that every continuous function $f : \mathbb{R}^n \to \mathbb{R}$ is lower semi-continuous.
2. Are all convex functions from $\mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ lower semi-continuous? If yes, prove it. If not, give a counterexample.
3. Let $X \subseteq \mathbb{R}^n$ be a compact set, $f : X \to \mathbb{R}$ and $g : X \to \mathbb{R}$ be continuous functions. Show that the perturbation function $\phi(b) := \min\{f(x) : g(x) \geq b, x \in X\}$ is lower semi-continuous.

`Lecture_3/LowerSemiContinuous`

Dual of the norm cone Let $||\cdot||$ be a norm on $\mathbb{R}^n$, and $||y||_* := \sup_{||x||=1}\langle y, x\rangle$ be its dual norm with respect to a scalar product $\langle\cdot,\cdot\rangle$. Considering the scalar product:

$$\left\langle \begin{pmatrix} s \\ y \end{pmatrix}, \begin{pmatrix} t \\ x \end{pmatrix} \right\rangle := st + \langle y, x\rangle$$

on $\mathbb{R} \times \mathbb{R}^n$, prove that the dual cone of:

$$K := \{(t, x) \in \mathbb{R} \times \mathbb{R}^n : t \geq ||x||\}$$

is:

$$K^* := \{(s, y) \in \mathbb{R} \times \mathbb{R}^n : s \geq ||y||_*\}.$$

`Lecture_3/DualNormCone`

Automatic backward differentiation [1] Assume that a function $f : \mathbb{R}^n \to \mathbb{R}$ is given as in Algorithm [1], where every step of the loop consists of a *simple* operation g_i. In general, the operation g_i uses the data $(v_{1-n}, \cdots, v_0) \equiv (x_1, \ldots, x_n)$ and what is already computed $(v_1, ..., v_{i-1})$ to compute v_i, which can then be used in subsequent operations. The last

Algorithm 1 Function evaluation

Input: $x \in \mathbb{R}^n$
Output: $f(x) \in \mathbb{R}$
Begin
  for $i := 1, 2, \ldots, n,$
    $v_{i-n} := x_i$
  end for
  for $i := 1, 2, \ldots, m,$
    $v_i := g_i(v_{1-n}, v_{2-n}, \ldots, v_{i-1})$
  end for
  $f(x) := v_m$
end

operation computes v_m, which is our output $f(x)$. Usually, g_i takes only very few inputs and its differential can be computed in closed form.

For differentiating this function, we can apply the backward mode of automatic differentiation described below in Algorithm [2]. The key to efficiency in this algorithm is, of course, that the inner loop (in the variable j) runs in reality only through the indices j of those variables v_j that actually influence v_i.

1. Prove that this algorithm indeed gives the gradient of f in x.
   Hint: you can represent the value of f at x in function of v in $m+1$ different ways. For instance, $f(x) = \Phi_{m+1}(v_{1-n}, \ldots, v_m) := v_m$, or $f(x) = \Phi_m(v_{1-n}, \ldots, v_{m-1}) := g_m(v_1, \ldots, v_{m-1})$, or $f(x) = \Phi_{m-1}(v_{1-n}, \ldots, v_{m-2}) := g_m(v_{1-n}, \ldots, v_{m-2}, g_{m-1}(v_{1-n}, \ldots, v_{m-2}))$, etc... Now, what does $\bar{v}_j$ represents at iteration i?
2. Use the Automatic Differentiation procedure to compute the gradient of the function implemented in `Nesterov_Chebysev.m`.
   Write and test a Matlab code and compare its output and computation time with the more standard procedure where you approximate each component of the gradient by $(f(x + te_i) - f(x))/t$, where $t = \sqrt{\epsilon_{\text{mach}}}$ and ϵ_{mach} is the machine precision of the float-point system you are using. Alternatively, you can use Python or R, but you would need to rewrite `Nesterov_Chebysev.m` in the language you wish to use; note that vector components are numbered from 1 in Matlab and in R, but from 0 in Python.

`Lecture_3/AutomaticDifferentiation`

[1] This method is also known as the *Backpropagation Algorithm* in the context of Artificial Neural Networks

Algorithm 2 Function and gradient evaluation

Input: $x \in \mathbb{R}^n$
Output: $f(x) \in \mathbb{R}, \nabla f(x) \in \mathbb{R}^n$
Begin
//Function evaluation:
for $i := 1, 2, \ldots, n,$
$v_{i-n} := x_i$
end for
for $i := 1, 2, \ldots, m,$
$v_i := g_i(v_{1-n}, v_{2-n}, \ldots, v_{i-1})$
end for
$f(x) := v_m$
end //Adjoint derivative computation:
Begin
for $i := 1, 2, \ldots, n+m,$
$\bar{v}_{i-n} := 0$
end for
$\bar{v}_m := 1$
//Backwards loop:
for $i := m, m-1, \ldots, 1,$
for $j := -n+1, -n+2, \ldots, i-1,$
$\bar{v}_j := \bar{v}_j + \bar{v}_i \frac{\partial g_i}{\partial v_j}(v_{1-n}, v_{2-n}, \ldots, v_{i-1})$
end for
end for
for $i := 1, 2, \ldots, n,$
$\nabla f_i(x) := \bar{v}_{i-n}$
end for
end