Non-convex inverse problems: programming exercises

Irène Waldspurger

waldspurger@ceremade.dauphine.fr

January and February 2023

1 Convexification for low-rank matrix recovery

In this exercise, we will try to recover low-rank matrices through nuclear norm minimization.

1. We first consider the problem of matrix completion, for matrices of size $d \times d$ (for some $d \in \mathbb{N}^*$) and rank 1 :

$$\begin{aligned} &\text{recover } X_0 \in \mathbb{R}^{d\times d} \\ &\quad \text{from } (X_0)_{ij}, \forall (i,j) \in \Omega, \\ &\text{knowing that } \operatorname{rank}(X_0) = 1. \end{aligned} \qquad \text{(Matrix Completion)}$$

   We will perform tests for random matrices X_0, generated as

$$X_0 = (u_i v_j)_{1\le i,j\le d},$$

   where $u_1, \dots, u_d, v_1, \dots, v_d$ are independent random variables, with uniform distribution in $[-1; 1]$.

   a) Write a function which, given d and m, generates a random $X_0 \in \mathbb{R}^{d\times d}$ as above, and a random subset Ω of $\{1, \dots, d\}^2$ containing m elements chosen uniformly at random.

b) Write a function which, given d, $((X_0)_{ij})_{(i,j)\in\Omega}$ and Ω, returns the solution X_{cvx} of the following convex problem :

$$\begin{aligned} &\text{minimize } ||X||_* \\ &\quad \text{for } X \in \mathbb{R}^{d\times d}, \\ &\text{such that } X_{ij} = (X_0)_{ij}, \forall (i,j) \in \Omega. \end{aligned} \tag{Convex MC}$$

[See below for indications on how to solve such problems in Julia and Python.]

c) Test the previous function for a random $X_0 \in \mathbb{R}^{10\times 10}$ and Ω of size 50. Check whether X_{cvx} is equal to X_0 or not. [1]
[Hint : run the test several times. If your implementation is correct, X_{cvx} and X_0 should be equal in roughly half of the trials.]

d) For $d = 10$ and each $m = 20, 30, ..., 100$, try to solve 10 random instances of (Matrix Completion) using (Convex MC). For each m, compute the empirical probability that $X_{cvx} = X_0$. What is the smallest m for which the probability is above 50% ? Which percentage of entries of X_0 does it correspond to ?

e) Same question for $d = 20$ and $m = 80, 100, ..., 200$, then for $d = 40$ and $m = 270, 310, ..., 390$.

2. In this question, we consider phase retrieval problems :

$$\begin{aligned} &\text{reconstruct } x_0 \in \mathbb{C}^d \\ &\text{from } |\langle x_0, v_j\rangle|, \forall j = 1, \dots, m, \end{aligned} \tag{Phase Retrieval}$$

where $(v_1, \dots, v_m)$ is a (known) family of *measurement vectors*.
We will perform tests for random problems, where $x_0, v_1, \dots, v_m \in \mathbb{C}^d$ are generated according to independent standard normal complex distributions [2].

a) Write a function which, given $(|\langle x_0, v_j\rangle|)_{j\le m}$ and $(v_j)_{j\le m}$, computes

1. You can consider that $X_{cvx} = X_0$ if $||X_{cvx} - X_0||_F \le 0.01||X_0||_F$. This is an arbitrary but reasonable rule.

2. A random variable z follows a standard normal complex distribution if $\mathrm{Re}(z)$ and $\mathrm{Im}(z)$ are independent normal variables, both with mean 0 and variance 1/2.

the solution X_{cvx} of the convex relaxation

$$\begin{aligned} &\text{minimize } \operatorname{Tr}(X) \\ &\text{for } X \in \mathbb{C}^{d\times d} \\ &\text{such that } v_j^* X v_j = |\langle x_0, v_j\rangle|^2, \forall j = 1,\dots,m, \\ &X \succeq 0. \end{aligned} \tag{PhaseLift}$$

b) As seen during the lecture, if the relaxation is tight, then

$$X_{cvx} = x_0 x_0^*.$$

Assuming this equality holds, write a function which, given X_{cvx}, computes x_0.

c) Show that, for any $z_1, z_2 \in \mathbb{C}^d$, the distance up to global phase between z_1 and z_2 satisfies

$$\operatorname{dist}(z_1, z_2) \overset{def}{=} \min_{\phi\in\mathbb{R}} ||e^{i\phi} z_1 - z_2||_2 = \sqrt{||z_1||^2 - 2|\langle z_1, z_2\rangle| + ||z_2||^2}.$$

Write a function to compute this distance.

d) Using the functions you just implemented, try to solve a random phase retrieval problem with $d = 10$ and $m = 50$. Is the solution x_{cvx} you obtain equal[3] to x_0 ?
[Hint : if your implementation is correct, x_{cvx} and x_0 should almost always be equal.]

e) For $d = 10$ and each $m = 25, 30, \dots, 50$, try to solve 10 random instances of (Phase Retrieval) using (PhaseLift). For each m, compute the empirical probability that $x_{cvx} = x_0$. What is the smallest m for which the probability is above 50% ?
[Hint : pass a time limit of (for instance) 5 seconds to the (PhaseLift) solver ; otherwise, some instances will take a long time.]

f) Same question for $d = 20$ and $m = 50, 60, \dots, 100$.

3. As before, we declare that x_{cvx} and x_0 are equal if $\operatorname{dist}(x_{cvx}, x_0) \le 0.01||x_0||_2$

1.1 Semidefinite programming in Julia and Python

Let us explain how to solve the following problem :

$$\begin{aligned} \text{minimize } & ||X||_* \\ \text{over all } & X \in \mathbb{R}^{d_1 \times d_2} \\ \text{such that } & \mathrm{Tr}(XA_i^T) = y_i, \forall i \leq m, \end{aligned} \tag{1}$$

where $A_1, \dots, A_m \in \mathbb{R}^{d_1 \times d_2}$ and $y_1, \dots, y_m \in \mathbb{R}$ are given.

1.1.1 In Julia, with Convex.jl and SCS.jl

Several Julia packages allow to solve Problem (1). Here, we propose to use Convex.jl and SCS.jl.

```
using Convex, SCS
```

Convex.jl provides an interface to describe optimization problems and call a solver ; SCS.jl is a particular solver.
A possible code for solving Problem (1) with these packages would be as follows :

```
1 X = Variable(d1,d2)
2 for k=1:m
3   add_constraint!(X, tr(X * A[k]') == y[k])
4 end
5 problem = minimize(nuclearnorm(X))
6 solve!(problem,SCS.Optimizer)
7 X_sol = evaluate(X)
```

A complete example using this code is available at https://www.ceremade.dauphine.fr/~waldspurger/tds/22_23_s2/M2/tps/sdp_example.jl.
Line 1 declares the type of the unknown X. Here, it is a matrix with size $d_1 \times d_2$ and real coefficients. For complex coefficients, one would use

```
X = ComplexVariable(d1,d2)
```

Lines 2 to 4 declare the constraints which must be satisfied by X. Many other types of constraints exist. For instance, if $d_1 = d_2$, it is possible to require X to be semidefinite positive using

```
add_constraint!(X, X in :SDP)
```

Line 5 declares the objective function. Line 6 calls the SCS solver and Line 7 returns the optimal X found by the solver.
It is possible to pass options to the solver. To avoid information display, one would use

```
solve!(problem,SCS.Optimizer; silent_solver=true)
```

To set a time limit of 5 seconds, it would be

```
solve!(problem,Convex.MOI.OptimizerWithAttributes(
  SCS.Optimizer, "time_limit_secs" => 5.))
```

1.1.2 In Python, with CVXPY

In Python, we propose to solve Problem (1) using CVXPY. This package provides an interface to define convex optimization problems and pass them to solvers.

```
import cvxpy as cp
```

A possible code for solving Problem (1) using CVXPY is as follows :

```
1 X = cp.Variable((d1,d2))
2 constraints = [cp.trace(X @ A[:,:,k].T) == y[k]
3                for k in range(m)]
4 objective = cp.Minimize(cp.norm(X,"nuc"))
5 problem = cp.Problem(objective,constraints)
6 problem.solve(solver=cp.SCS)
7 return X.value
```

A complete example using this code is available at https://www.ceremade.dauphine.fr/~waldspurger/tds/22_23_s2/M2/tps/sdp_example.py.
Line 1 declares the variable X, here a variable of size $d_1 \times d_2$ with real coordinates. To declare a matrix of size $d_1 \times d_2$ with complex coordinates, one would have used

```
X = cp.Variable((d1,d2),complex=True)
```

And to additionally constrain X to be Hermitian,

```
X = cp.Variable((d,d),hermitian=True)
```

Lines 2 and 3 declare the list of constraints. Other types of constraints than linear are possible. For instance, to constrain a symmetric or Hermitian matrix to be semidefinite positive, one can add

```
constraints.append(X >> 0)
```

Line 4 declares the objective function. Line 5 and 6 define the problem and call the solver[4]. Line 7 returns the optimal X found by the solver.
To set a time limit for the solver, use

```
problem.solve(solver=cp.SCS,time_limit_secs=5.)
```

4. Here, the solver is SCS so as to match the code proposed for Julia users, but other solvers are of course possible.

2 Gradient descent on non-convex objectives

We consider the function

$$\begin{array}{rccc} f: & \mathbb{R}^2 & \to & \mathbb{R} \\ & (x,y) & \to & 4x^4+4x^2y^2+3y^4-6x^3-2y^3+x^2-3y^2. \end{array}$$

[Note : we consider this specific function because it has a nice landscape of critical points, but it is not especially interesting otherwise.]

1. Compute its gradient.
2. Write functions `f`, `grad_fx` and `grad_fy` which compute the value of f and the two components of its gradient at a given point.
3. Choose several points p_0 uniformly at random in the square $[-2;2]\times[-2;2]$. For each one of them, run 400 steps of gradient descent with stepsize $\frac{1}{150}$, starting at p_0. Check that each run converges towards one of the following three points :

$$M_1=(1,0), \quad M_2=(0,1), \quad M_3=\left(0,-\frac{1}{2}\right).$$

4. Show that M_1, M_2, M_3 are second-order critical points of f.
5. Show that $P_1=(0,0)$ and $P_2=(1/8,0)$ are first-order, but not second-order critical points of f.

It is possible to show that f has only two other first-order critical points,

$$P_3 \approx (0.275,-0.465) \quad \text{and} \quad P_4 \approx (0.899, 0.396),$$

which are also not second-order critical points.

6. What is the global minimum of f ?
7. a) For each $p\in[-2;2]$, we define

$$\texttt{limit} = k \text{ for } k\in\{1,2,3\}$$

if the 400-th gradient descent iterate, with starting point p, is at distance at most 0.1 from M_k. We define

$$\texttt{limit} = 0 \text{ otherwise.}$$

Compute `limit` for all points on a grid with spacing 0.01 in $[-2;2]\times[-2;2]$. Display it as an image.

b) On the same plot, display the negative gradient as a vector field, at each point of a grid with spacing 0.2. Normalize each gradient to improve the readibility of the figure.
[In Python, you can plot a vector field using `quiver(U,V,X,Y)` from the `matplotlib.pyplot` library, where `U`, `V` are the origins of the arrows and `X`, `Y` their coordinates. This function is also available in the Julia `PyPlot` module.]

8. Repeat the last question on a grid with spacing 0.001 (0.025 for the gradient) in $[-0.1, 0.3] \times [-0.2, 0.2]$.
9. Comment the pictures : what do they look like in the neighborhood of critical points ? Which signs indicate whether a critical point is first or second-order ? Using the information visible on the pictures, can you draw the shape of the gradient descent trajectories, in the different regions of the plane ?