Tehnici de Optimizare
Laborator 3
– Probleme de minimizare cu constrângeri –

1 Introducere: Filtrare de trend

Fie seria de timp $\{y_t\}_{t=1}^n \in \mathbb{R}^n$, presupunem descompunerea:

$$y_t = x_t + \epsilon_t,$$

unde x_t reprezintă componenta-trend de variaţie lentă, iar ϵ_t componenta zgomot de variaţie rapidă. Scopul operaţiei de *filtrare a trend-ului* se rezumăla estimarea componentei x_t. Presupunem proprietatea implicită de "netezime" a componentei-trend.

Filtru Hodrick-Prescott. Primul model pe care îl analizăm pentru rezolvarea problemei.

$$\min_x \ \frac{1}{2}\sum_{t=1}^{n}(x_t - y_t)_2^2 + \rho\sum_{t=2}^{n-1}(x_{t-1} - 2x_t + x_{t+1})^2. \tag{1}$$

Observăm că funcţia obiectiv urmăreşte reducerea reziduului $\{y_t - x_t\}$ şi, în acelaşi timp, ajustarea "netezimii" lui x_t. Diferenţa de ordin 2: $x_{t-1} - 2x_t + x_{t+1}$ este nulă dacă şi numai dacă $\{x_{t-1}, x_t, x_{t+1}\}$ sunt coliniare. În forma restrânsă, funcţia obiectiv se rescrie: $\frac{1}{2}\|x - y\|_2^2 + \rho\|Dx\|_2^2$, unde $D \in \mathbb{R}^{(n-2)\times n}$

$$D = \begin{bmatrix} 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \vdots & \vdots & \vdots & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 & 1 \end{bmatrix}$$

Soluţia modelului HP are forma:

$$x_{HP}^* := \left(I + 2\rho D^T D\right)^{-1} y. \tag{2}$$

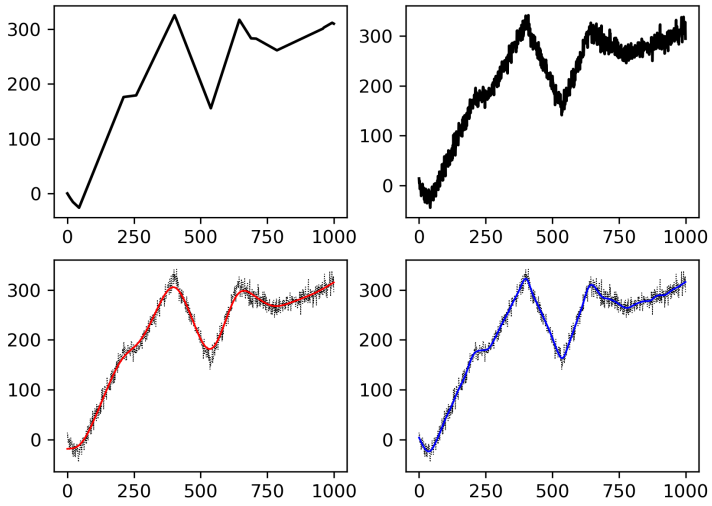
Eroarea relativă de estimare satisface:

$$\|y - x_{HP}^*\|_2 \leq \frac{32\rho}{1 + 32\rho}\|y\|_2$$

P1. Care este complexitatea calculului soluţiei x_{HP}^*?

Filtru ℓ_1. Pentru o reconstrucţie mai performantă, filtrul ℓ_1 înlocuieşte $\|\cdot\|_2^2$ cu $\|\cdot\|_1$. În acest model, se urmareşte o soluţie liniară pe porţiuni care poate reduce reziduul de estimare mai mult decât în cazul filtrului H-P, i.e.

$$x_{\ell_1}^* := \operatorname*{argmin}_x \ \frac{1}{2}\|x - y\|_2^2 + \rho\|Dx\|_1$$



În continuare, matricea D este matricea diferenţelor de ordin 2. În forma prezentă problema este nediferenţiabilă. Pentru a obţine o formă echivalentă rezolvabilă cu algoritmii de ordin I studiaţi la curs, reformulăm:

$$\begin{aligned} x^*_{\ell_1} :=& \operatorname*{argmin}_x \ \frac{1}{2}\|x-y\|_2^2 + \rho\|z\|_1 \\ & \text{s.l. } Dx = z. \end{aligned}$$

Funcţia Lagrangian pentru problema de mai sus devine:

$$\mathcal{L}(x,z,\mu) := \frac{1}{2}\|x-y\|_2^2 + \rho\|z\|_1 + \mu^T(Dx-z)$$

Observăm:

$$[z(x,\mu)]_i := \begin{cases} 0, & \text{dacă } \ -\rho \le \mu_i \le \rho \\ -\infty, & \text{dacă } \ |\mu_i| > \rho. \end{cases}$$

şi deducem condiţiile de optimalitate:

$$\begin{aligned} & \nabla_x \mathcal{L}(x^*_{\ell_1}, \mu^*) := x^*_{\ell_1} - y + D^T\mu^* = 0 \\ & -\rho \le \mu^*_i \le \rho. \end{aligned}$$

Problema duală Lagrange:

$$\begin{aligned} \mu^* = \arg\max_\mu \ & -\frac{1}{2}\|D^T\mu\|_2^2 + \mu^T D y \\ & \text{s.l. } \ -\rho \le \mu_i \le \rho \quad \forall i = 1, \cdots, n-2. \end{aligned} \tag{3}$$

Pe baza multiplicatorilor Lagrange optimi recuperăm: $x^*_{\ell_1} = y - D^T\mu^*$.

1.1 Metoda de Gradient Proiectat

Metoda Gradient reprezintă un algoritm de ordin I care, pornind dintr-un punct iniţial ales $x^0 \in Q$, generează un şir de iteraţii (vectori) $x^1, x^2, \cdots$ pe baza direcţiei antigradientului redus $-\mathcal{G}(\cdot)$. Vom presupune că f are gradient $L-$continuu Lipschitz.

Metoda Gradient(x^0, ϵ)

Iniţializează $k = 0$.

Cât timp $\|\nabla f(x^k)\| \leq \epsilon$:

1. Calculează $\nabla f(x^k)$
2. Actualizează: $x^{k+1} = \pi_Q(x^k - \alpha_k \nabla f(x^k))$
3. Set $k := k + 1$.

La linia 1, presupunem că un oracol de ordin I returnează gradientul ∇f evaluat în punctul curent x^k. Un aspect important al metodei este alegerea pasului $\alpha_k > 0$ dintre următoarele opţiuni:

- **Constant**: $\alpha_k = \alpha \in \left(0, \frac{2}{L}\right)$
- **Ideal**: $\alpha_k = \arg\min_{\alpha \geq 0} f(x^k - \alpha \nabla f(x^k))$
- **Adaptiv (Backtracking)**: alege $c > 0$, ajustează pasul α_k astfel încât să aibă loc relaţia de descreştere:

$$f(x^{k+1}) \leq f(x^k) - c\alpha_k \|\mathcal{G}(x^k)\|^2. \tag{4}$$

  Procedura de ajustare presupune alegerea $\rho \in (0, 1]$ şi actualizarea:

  (i) Alegem $c, \rho \in (0, 1), \alpha_{k,0} > 0$

  (ii) Cât timp $\alpha_{k,t}$ nu satisface (4) iterăm: $\alpha_{k,t+1} := \rho \alpha_{k,t}; \quad t := t + 1;$

2 Probleme propuse

1. Consideraţi problema filtrării trend-ului unei serii de timp $\{y_t\}_{t \geq 0}$. Seria de timp $\{y_t\}_{t \geq 0}$ este stocată în vectorul y:

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2}\|x - y\|^2 + \rho \|Dx\|_1$$

unde $\rho > 0$ şi D matricea de diferenţe definită mai sus.

a) Generaţi $\{y_t\}_{t \geq 0}$ respectând structura (parametrii la alegere):

- $y_t = x_t + z_t \quad x_{t+1} = x_t + v_t$
- z_t aleator după o distribuţie normală de medie 0 şi varianţă σ^2
- v_t este generat progresiv după regula: (i) se fixează probabilitatea p şi v_1 aleator; (ii) la indexul t avem $v_t = v_{t-1}$ cu probabilitatea p, altfel v_t generat aleator cu distribuţie uniformă; (iii) $t := t+1$.

  Afişaţi două figuri cu x_t şi y_t.

b) Cu datele de la $a)$ şi $n \geq 10^4$, calculaţi soluţia problemei folosind CVXPY. Variaţi valoarea lui ρ.

c) Cu datele de la $a)$ şi $n \geq 10^4$, calculaţi o aproximare a soluţiei folosind următoarea procedură:

  (i) Aplicaţi MGP (pas constant sau backtracking) pe problema duală (3), pentru a calcula o aproximare a multiplicatorilor optimi $\hat{\mu} \approx \mu^*$ cu acurateţe $\epsilon = 10^{-2}$, i.e. criteriul de oprire $\|\mathcal{G}(x^k)\| \leq \epsilon$;
  (ii) Recuperaţi soluţia aproximativă: $\hat{x}_{\ell_1} = y - D^T \hat{\mu}$.
  Variaţi valoarea lui ρ.

d) Scrieţi o rutină Python pentru calcularea eficientă a lui x^*_{HP} $(O(n))$. Indicaţie: folosim structura tridiagonală a lui D pentru rezolvarea sistemului (2).

e) Raportaţi o comparaţie (grafică sau numerică) a soluţiilor: $\hat{x}_{\ell_1}$ şi x^*_{HP}.

2.1 Ghid Python

Importare pachete necesare:

```
import numpy as np
from numpy import linalg
from numpy.linalg import norm
import numpy.random as random
import matplotlib.pyplot as plt
```

Exemplu generare vector după distribuţie normală:

```
z_t = random.normal(0, sigma, n)
```

Exemplu generare vector după distribuţie uniformă:

```
s_t = random.rand(n)
s_t = random.randn(n)
```

Exemplu calcul expresie gradient:

```
gradient = lambda x: np.matmul(D, np.matmul(D.T, x) - y_t)
```

Exemplu calcul proiecţie pe mulţimea $[-u, u]^n$:

```
projection = lambda x: np.maximum(-u, np.minimum(u,x))
```