TP5 - Variationnel

February 1, 2024

William Liaw

1 Débruitage par régularisation quadratique

1.1 Comment utiliser l'outil `resoud_quad_fourier` pour trouver le minimiseur de cette énergie (voir le programme `minimisation_quadratique`)?

On cherche à retrouver u comme minimiseur de l'énergie:

$$\begin{aligned}
&\|u-v\|^2+\lambda\|\nabla u\|^2\\
&\|\delta*u-v\|^2+\lambda(\|K_x*u\|^2+\|K_y*u\|^2)\\
&\|\delta*u-v\|^2+\|\sqrt{\lambda}K_y*u-0\|^2+\|\sqrt{\lambda}K_x*u-0\|^2\\
&\|\sqrt{\lambda}K_x*u-0\|^2+\|\sqrt{\lambda}K_y*u-0\|^2+\|\delta*u-v\|^2\\
&\sum_{i=1}^{3}\|K_i*u-V_i\|^2
\end{aligned}$$

Ainsi, minimiser l'expression originale de l'énergie revient à trouver une image qui minimise $\sum_{i=1}^{3}\|K_i*u-V_i\|^2$, pour $K=(\sqrt{\lambda}K_x,\sqrt{\lambda}K_y,\delta)$ et $V=(0,0,v)$. En prenant la transformée de Fourier de la dernière expression, on obtient: $\sum_{\omega\in f}\sum_{i=1}^{3}|\hat{K}_i\cdot\hat{u}-\hat{V}_i|^2 \Rightarrow \sum_{i=1}^{3}\sum_{\omega\in f}|\hat{K}_i\cdot\hat{u}-\hat{V}_i|^2$. En analysant le terme dans la sommation:

$$\begin{aligned}
&|\hat{K}_i\cdot\hat{u}-\hat{V}_i|^2\\
&|\hat{K}_i|^2\cdot|\hat{u}|^2-2\mathbb{R}(\hat{V}_i\overline{\hat{K}_i}\overline{\hat{u}})+|\hat{V}_i|^2\\
&|\hat{K}_i|^2\left(|\hat{u}|^2-2\mathbb{R}\left(\frac{\hat{V}_i\overline{\hat{K}_i}\overline{\hat{u}}}{|\hat{K}_i|^2}\right)+\frac{|\hat{V}_i|^2}{|\hat{K}_i|^2}\right)
\end{aligned}$$

En supprimant les termes indépendants de u, on peut voir que la minimisation de l'expression originale de l'énergie revient à minimiser: $|\hat{u}|^2-2\mathbb{R}\left(\frac{\hat{V}_i\overline{\hat{K}_i}}{|\hat{K}_i|^2}\overline{\hat{u}}\right)+|\gamma|^2=|\hat{u}-\gamma|^2,\quad \overline{\gamma}=\frac{\hat{V}_i\overline{\hat{K}_i}}{|\hat{K}_i|^2}$. Ainsi:

$$\widehat{u_{\min}} = \gamma$$

$$\therefore u_{\min} = \mathbb{R}\left(\mathscr{F}^{-}\left(\frac{\hat{V}_i\overline{\hat{K}_i}}{|\hat{K}_i|^2}\right)\right)$$

1.2 Décrire le résultat de ce débruitage lorsque λ est très grand ou très petit.

Lorsque la valeur de λ est très petite, le processus de débruitage donne des résultats peu significatifs: l'image reste bruitée. À mesure que nous accroissons la valeur de λ, la qualité de l'image restaurée se rapproche de celle de l'image parfaite, représentée par u. Cependant, en continuant à augmenter λ, l'image obtenue devient de plus en plus floue.

L'augmentation de λ correspond à un renforcement du poids du terme de régularisation. Cela se traduit par une préférence croissante pour une image résultante présentant des caractéristiques statistiques naturelles, plutôt que de rester strictement fidèle aux données d'origine. En d'autres termes, une pondération plus importante de ce terme favorise une régularité visuelle accrue au détriment de la fidélité stricte aux données initiales.

1.3 Après avoir ajouté un bruit d'écart type $\sigma = 5$ à l'image de lena, trouver (par dichotomie) le paramètre λ pour lequel $\|\tilde{u} - v\|^2 \sim \|u - v\|^2$. C'est-à-dire le paramètre pour lequel l'image reconstruite $\tilde{u}$ est à la même distance de l'image parfaite u que ne l'est l'image dégradée.

Par le biais de la méthode de dichotomie, notre démarche a débuté en définissant un intervalle initial de de manière suffisamment étendue. À chaque itération, nous avons calculé la différence $\|\tilde{u} - v\|^2 - \|u - v\|^2$ pour les valeurs de λ: λ_{left} et λ_{right}, ainsi qu'à la moyenne entre ces deux valeurs. Ensuite, nous avons remplacé l'une des valeurs de λ_{left} et λ_{right} par cette moyenne. Grâce à cet algorithme itératif, nous avons pu déterminer que la valeur optimale de λ est égale à 3.207. Cela suggère que, dans le contexte d'optimisation, cette valeur spécifique de λ minimise la différence entre les images $\tilde{u}$ et u.

[ ]:
```
def dichotomic_search_zero(
    function, left_bound, right_bound, tolerance=1e-6, max_iterations=1000
):
    for iteration in range(max_iterations):
        mid_point = (left_bound + right_bound) / 2

        # Evaluate the function at the mid-point and its neighbors
        f_mid = function(mid_point)
        f_left = function(left_bound)
        f_right = function(right_bound)

        # Check if the minimum is on the left or right side of the interval
        if f_left * f_mid <= 0:
            right_bound = mid_point
        else:
            left_bound = mid_point
```

```
        # Check for convergence
        if abs(f_right - f_left) < tolerance:
            break

    # Return the minimum value and the argument at which it occurs
    min_value = function((left_bound + right_bound) / 2)
    min_argument = (left_bound + right_bound) / 2
    return min_value, min_argument
```

[ ]:
```
im = imread("lena.tif")
imb = degrade_image(im, 5**2)

t = norm2(im - imb) ** 2
function = lambda lamb: norm2(minimisation_quadratique(imb, lamb) - imb) ** 2 -␣
 ↪t

min_error, min_lambda = dichotomic_search_zero(
    function=function,
    left_bound=1,
    right_bound=1e3,
    tolerance=1e-6,
    max_iterations=1000,
)

print(f"Best lambda: {min_lambda:.3f}")
viewimage(minimisation_quadratique(imb, min_lambda))
```

1.4 Écrire un algorithme pour trouver le paramètre λ tel que $\|\tilde{u} - u\|^2$ soit minimale. (dans le cadre de ce TP on connaît l'image parfaite u). Commentaires?

Une fois de plus, par le biais de la méthode de dichotomie, notre démarche a débuté en définissant un intervalle initial de de manière suffisamment étendue. À chaque itération, nous avons calculé la différence $\|\tilde{u} - u\|^2$ pour les valeurs de λ: λ_{left} et λ_{right}, ainsi qu'à la moyenne entre ces deux valeurs. Ensuite, nous avons remplacé l'une des valeurs de λ_{left} et λ_{right} par cette moyenne. Grâce à cet algorithme itératif, nous avons pu déterminer que la valeur optimale de λ est égale à 1.122. Cela suggère que, dans le contexte d'optimisation, cette valeur spécifique de λ minimise la différence entre les images $\tilde{u}$ et u.

[ ]:
```
def dichotomic_minimization(
    function, left_bound, right_bound, tolerance=1e-6, max_iterations=1000
):
    x = []
    y = []
    for iteration in range(max_iterations):
        mid_point = (left_bound + right_bound) / 2
```

```
        # Evaluate the function at the mid-point and its neighbors
        f_mid = function(mid_point)
        f_left = function(left_bound)
        f_right = function(right_bound)
        if f_right <= f_mid:
            left_bound = mid_point
        else:
            right_bound = mid_point

        # Check for convergence
        if abs(f_right - f_left) < tolerance:
            break

    # Return the minimum value and the argument at which it occurs
    min_value = function((left_bound + right_bound) / 2)
    min_argument = (left_bound + right_bound) / 2
    return min_value, min_argument
```

[ ]:
```
im = imread("lena.tif")
imb = degrade_image(im, 5**2)

function = lambda lamb: norm2(minimisation_quadratique(imb, lamb) - im) ** 2

min_error, min_lambda = dichotomic_minimization(
    function=function,
    left_bound=1,
    right_bound=1e3,
    tolerance=1e-6,
    max_iterations=1000,
)

print(f"Best lambda: {min_lambda:.3f}")
viewimage(minimisation_quadratique(imb, min_lambda))
```

L'analyse des résultats révèle que la deuxième méthode offre une image améliorée et une erreur plus réduite $\|\tilde{u} - u\|^2$. D'un point de vue géométrique, on peut expliquer cette différence: la première méthode recherche l'intersection entre les cercles de rayon σ^2 centrés en u et v, tandis que la deuxième méthode trace $\tilde{u} = v(\lambda)$, une ligne à partir de v, et trouve la projection orthogonale de u. Cette approche semble mieux capturer la structure géométrique de l'espace des solutions, conduisant ainsi à des résultats plus précis et à une réduction de l'erreur.

2 Débruitage par variation totale

2.1 Descente de gradient

[ ]:
```
im = imread("lena.tif")
imb = degrade_image(im, 5**2)

u, energ01 = minimise_TV_gradient(imb, 3, 0.1, 100)
u, energ02 = minimise_TV_gradient(imb, 3, 0.2, 100)
u, energ04 = minimise_TV_gradient(imb, 3, 0.4, 100)
u, energ06 = minimise_TV_gradient(imb, 3, 0.6, 100)
u, energ08 = minimise_TV_gradient(imb, 3, 0.8, 100)
u, energ10 = minimise_TV_gradient(imb, 3, 1.0, 100)

plt.plot(np.log(energ01), label="0.1")
plt.plot(np.log(energ02), label="0.2")
plt.plot(np.log(energ04), label="0.4")
plt.plot(np.log(energ06), label="0.6")
plt.plot(np.log(energ08), label="0.8")
plt.plot(np.log(energ10), label="1.0")

plt.gca().spines["top"].set_alpha(0.0)
plt.gca().spines["bottom"].set_alpha(0.3)
plt.gca().spines["right"].set_alpha(0.0)
plt.gca().spines["left"].set_alpha(0.3)

plt.grid()
plt.legend()
plt.show()
```

Il est évident que nous n'atteignons toujours pas le même minimum d'énergie. Nous faisons face à des problèmes numériques lors de la minimisation de la variation totale par descente de gradient à pas constant. Plus spécifiquement, lorsque nous optons pour une valeur de pas très grande, l'algorithme rencontre des difficultés à converger vers le minimum recherché. D'un autre côté, une taille de pas excessivement petite entraîne une convergence plus lente de l'algorithme.

Il est d'une importance primordiale de bien comprendre le problème en question afin de choisir une taille de pas optimale: ni trop grande pour garantir la convergence de l'algorithme, ni trop petite pour permettre à l'algorithme de tirer le meilleur parti de chaque itération. Trouver le bon équilibre dans le choix de la taille du pas est essentiel pour assurer une convergence stable et efficace du processus d'optimisation.

2.2 Projection Chambolle

[ ]:
```
im = imread("lena.tif")
imb = degrade_image(im, 5**2)
lamb = 3
```

```
u_grad, energ_quad = minimise_TV_gradient(imb, lamb, 0.6, 100)
u_chamb = vartotale_Chambolle(imb, lamb, 100)

function = lambda u: norm2(u - im) ** 2
print(function(u_grad))
print(function(u_chamb))
```

Nous constatons que les deux méthodes donnent des résultats similaires en comparant $\|\tilde{u}_{\text{grad}} - v\|^2$ et $\|\tilde{u}_{\text{chamb}} - v\|^2$. Cependant, il est à noter que la méthode de Chambolle présente une efficacité computationnelle nettement plus rapide.

3 Comparaison

[ ]:
```
im = imread("lena.tif")
imb = degrade_image(im, 5**2)

errvt = []
erreur = []
vk = np.concatenate((np.linspace(0.1, 3, 50), np.linspace(3, 60, 120)))

for k in vk:
    restq = minimisation_quadratique(imb, k)
    restva = vartotale_Chambolle(imb, k)
    erreur.append(norm2(im - restq))
    errvt.append(norm2(restva - im))

plt.plot(vk, erreur, label="Quadratique")
plt.plot(vk, errvt, label="Chambolle")

plt.gca().spines["top"].set_alpha(0.0)
plt.gca().spines["bottom"].set_alpha(0.3)
plt.gca().spines["right"].set_alpha(0.0)
plt.gca().spines["left"].set_alpha(0.3)

plt.grid()
plt.legend()
plt.show()

print(vk[np.argmin(erreur)])
print(vk[np.argmin(errvt)])
```

Les paramètres optimaux trouvés pour la minimisation quadratique étaient $\lambda = 1.165$, et pour la variation totale $\lambda = 41.319$.

En conclusion, il semble que, du point de vue quantitatif, la méthode de variation totale soit plus efficace. En premier lieu, elle s'avère au moins aussi rapide à calculer que la méthode de minimisation quadratique par la méthode de Chambolle, évitant ainsi le besoin de calcul des transformées de Fourier. En outre, l'erreur obtenue par cette méthode est 1.199 fois plus petite que celle obtenue

par l'autre approche.

D'un point de vue qualitatif, les résultats révèlent une nuance plus subtile. Bien que la méthode de variation totale réduise de manière évidente le bruit dans l'image, il semble que nous ayons sacrifié un peu de contraste (même si cela ne se manifeste pas clairement sur les histogrammes).