

- (a) Rufen Sie sich das Entfaltungsproblem ins Gedächtnis. Nehmen Sie eine Poisson-Statistik für die Werte in \vec{g} an. Stellen Sie eine Likelihood-Funktion für die Verteilung der Zielvariable \vec{f} auf.

$$\vec{g} = A \cdot \vec{f}$$

m Messwerte/Bins der Messwerte $\rightarrow \vec{g}$
 n "wahre" Werte / Bins der "wahren" Werte $\rightarrow \vec{f}$

$\Rightarrow A$ ist $m \times n$ und die Responsematrix

Entfaltung mit der Likelihood-Methode:

Annahme: \vec{g} sind Poissonverteilt

Erwartungswert des i-ten Bins in \vec{g} ist:

$$\lambda_i = (A\vec{f})_i = \vec{a}_i \cdot \vec{f}; \text{ wobei } \vec{a}_i \text{ die i-te Zeile von } A \text{ ist}$$

$$\Leftrightarrow \vec{\lambda} = A\vec{f}$$

$$P(g_i | \vec{f}) = \frac{\lambda_i^{g_i}}{g_i!} \exp(-\lambda_i)$$

Likelihood:

$$L(\vec{f} | \vec{g}) = \prod_{i=1}^m P(g_i | \vec{f}) = \prod_{i=1}^m \frac{\lambda_i^{g_i}}{g_i!} \exp(-\lambda_i)$$

negative Log-Likelihood:

$$\begin{aligned} F(\vec{f} | \vec{g}) &= -\ln(L(\vec{f} | \vec{g})) \\ &= -\sum_{i=1}^m \ln(P(g_i | \vec{f})) \\ &= -\sum_{i=1}^m g_i \ln(\lambda_i) - \lambda_i - \ln(g_i!) \\ &= -\sum_{i=1}^m g_i (\ln((A\vec{f})_i) - (A\vec{f})_i - \ln(g_i!)) \end{aligned}$$

- (b) Fügen Sie nun den Tikhonov-Regularisierungs-Term $\frac{\tau}{2} ||\Gamma \cdot \vec{f}||^2$ hinzu. Berechnen Sie analytisch den Gradienten

$$(\nabla(-\log \mathcal{L}))(\vec{f})_i = \frac{\partial(-\log \mathcal{L})}{\partial f_i} \quad (1)$$

und die Hesse-Matrix

$$H[-\log \mathcal{L}](\vec{f})_{ij} = \frac{\partial^2(-\log \mathcal{L})}{\partial f_i \partial f_j} \quad (2)$$

Regularisierung mit Tikhonov:

$$F_{\text{reg}}(\vec{f} | \vec{g}) = F(\vec{f} | \vec{g}) + \frac{\gamma}{2} \|\Gamma \vec{f}\|^2$$

wobei $\gamma \triangleq$ Regularisierungskonstante

$\Gamma \triangleq$ Operatormatrix der zweiten Ableitung

$$\Gamma = \begin{pmatrix} -1 & 1 & 0 & \cdots & & \\ 1 & -2 & 1 & \cdots & & \\ 0 & 1 & \ddots & & & \\ \vdots & \vdots & & 1 & 0 & \\ & & & 1 & -2 & 1 \\ 0 & 1 & -1 & & & \end{pmatrix}_{n \times n}$$

$$F_{\text{reg}}(\vec{f}) = - \sum_{i=1}^m \left[g_i \ln \left((A\vec{f})_i \right) - (A\vec{f})_i - (\ln(g_i!)) \right] + \frac{\gamma}{2} \|\Gamma \vec{f}\|^2 \quad \text{mit} \quad \|\Gamma \vec{f}\|^2 = \sum_{i=1}^n (\Gamma \cdot \vec{f})_i^2 = \sum_{i=1}^n \left(\sum_{j=1}^n \Gamma_{ij} f_j \right)^2$$

Als Summenschreibweise (zum Ableiten):

$$F_{\text{reg}}(\vec{f}) = - \sum_{i=1}^m \left[g_i \ln \left(\sum_{j=1}^n A_{ij} f_j \right) - \left(\sum_{j=1}^n A_{ij} f_j \right) - (\ln(g_i!)) \right] + \frac{\gamma}{2} \left(\sum_{i=1}^n \left(\sum_{j=1}^n \Gamma_{ij} f_j \right)^2 \right)$$

$$\frac{\partial F_{\text{reg}}}{\partial f_k}(\vec{f}) = - \sum_{i=1}^m \left[g_i A_{ik} \left(\sum_{j=1}^n A_{ij} f_j \right)^{-1} - A_{ik} \right] + \frac{\gamma}{2} \left(\sum_{i=1}^n \Gamma_{ik} \left(\sum_{j=1}^n \Gamma_{ij} f_j \right) \right)$$

$$\frac{\partial F_{\text{reg}}}{\partial f_k \partial f_l}(\vec{f}) = - \sum_{i=1}^m \left[g_i A_{ik} A_{il} (-1) \left(\sum_{j=1}^n A_{ij} f_j \right)^{-2} \right] + \gamma \sum_{i=1}^n \Gamma_{ik} \Gamma_{il}$$

Wieder mit $(A\vec{f})$ und $(\Gamma \vec{f})$

$$\text{Likelihood: } F_{\text{reg}}(\vec{f}) = \sum_{i=1}^m \left[-g_i \ln \left((A\vec{f})_i \right) + (A\vec{f})_i + (\ln(g_i!)) \right] + \frac{\gamma}{2} \|\Gamma \vec{f}\|^2$$

$$\text{Gradient: } (\vec{\nabla} F_{\text{reg}})_k(\vec{f}) = \sum_{i=1}^m \left[-g_i A_{ik} \left((A\vec{f})_i \right)^{-1} + A_{ik} \right] + \gamma \sum_{i=1}^n \Gamma_{ik} (\Gamma \cdot \vec{f})_i$$

$$\text{Hesse-Matrix: } (H(F_{\text{reg}}))_{kl}(\vec{f}) = \sum_{i=1}^m \left[g_i A_{ik} A_{il} \left((A\vec{f})_i \right)^{-2} \right] + \gamma \sum_{i=1}^n \Gamma_{ik} \Gamma_{il}$$

Gradient als reine Matrix-Rechnung:

- erster Term:

$$m \Gamma \cdot \vec{f} + \gamma \vec{f} = \vec{f}$$

- erster Term:

$$\begin{aligned} \sum_{i=1}^m \left[-g_i A_{ik} \left((\vec{Af})_i \right)^{-1} + A_{ik} \right] &= -\sum_{i=1}^m A_{ik} \frac{g_{i,1}}{(\vec{Af})_{i,1}} + \sum_{i=1}^m A_{ik} \\ &= \left[A^T \left(\vec{g} \otimes (\vec{Af}) \right) \right]_k + \left(A^T \cdot \vec{\mathbb{1}}_m \right)_k \quad \text{wobei } \otimes \text{ die Elementweise Division ist} \\ &\quad \text{(Hadamard division)} \\ &= \left[A^T \left(\vec{\mathbb{1}}_m - \vec{g} \otimes (\vec{Af}) \right) \right]_k \quad \text{und } \vec{\mathbb{1}}_m = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}_{m \times 1} \end{aligned}$$

- zweiter Term:

$$\approx \sum_{i=1}^n \Gamma_{ik} (\Gamma \vec{f})_i = \tau (\Gamma^T (\Gamma \vec{f}))_k$$

- zusammen:

$$\begin{aligned} (\vec{\nabla} F_{reg})(\vec{f})_k &= \left[A^T (\vec{\mathbb{1}}_m - \vec{g} \otimes (\vec{Af})) \right]_k + \tau [\Gamma^T (\Gamma \vec{f})]_k \\ \Rightarrow \boxed{(\vec{\nabla} F_{reg})(\vec{f}) = A^T (\vec{\mathbb{1}}_m - \vec{g} \otimes (\vec{Af})) + \tau \Gamma^T (\Gamma \vec{f})} \end{aligned}$$

Hesse-Matrix:

- erster Term:

$$\begin{aligned} \sum_{i=1}^m \left[g_i A_{ik} A_{il} \left((\vec{Af})_i \right)^{-2} \right] &= \sum_{i=1}^m A_{ik} A_{il} \frac{g_i}{(\vec{Af})_i^2} \\ &= \left[A^T \text{diag}(\vec{g} \otimes (\vec{Af} \otimes \vec{Af})) A \right]_{kl} \quad \text{wobei } \otimes \text{ die Elementweise Multiplikation ist} \\ &\quad \text{(Hadamard product)} \end{aligned}$$

- zweiter Term:

$$\approx \sum_{i=1}^n \Gamma_{ik} \Gamma_{il} = \tau [\Gamma^T \Gamma]_{kl}$$

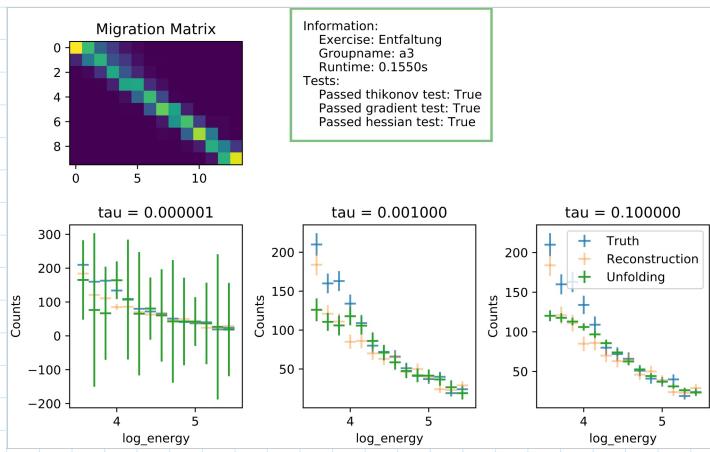
- zusammen:

$$\begin{aligned} [H(F_{reg})]_{kl}(\vec{f}) &= \left[A^T \text{diag}(\vec{g} \otimes (\vec{Af} \otimes \vec{Af})) A \right]_{kl} + \tau [\Gamma^T \Gamma]_{kl} \\ \Rightarrow \boxed{H(F_{reg})(\vec{f}) = A^T \text{diag}(\vec{g} \otimes (\vec{Af} \otimes \vec{Af})) A + \tau \Gamma^T \Gamma} \end{aligned}$$

- (c) Implementieren Sie zur Minimierung der Likelihood das Newton-Verfahren gemäß der Rechenvorschrift

$$f^{(k+1)} = f^{(k)} - [H[-\log \mathcal{L}](f) + \epsilon \mathbf{1}]^{-1} \cdot (\nabla(-\log \mathcal{L}))(f) \quad (3)$$

und nutzen Sie $\epsilon = 10^{-6}$. Brechen Sie das Verfahren ab, sobald $\|f^{(k+1)} - f^{(k)}\| < 10^{-10}$ ist.



Der Code ist weiter unten.

- (d) Erklären Sie die Effekte der Regularisierung. Welche Regularisierungsstärke erscheint angemessen? Warum?

Die gewählte Regularisierung unterdrückt große zweite Ableitungen.
Also werden starke Krümmungen "bestraft" und die Entfaltung wird "glatter".

Eine Regularisierungsstärke von 0.001 scheint hier am angemessendsten zu sein, da bei 0.000001 die Lösung stark oszilliert und die Fehler sehr groß sind.

Bei 0.1 sind zwar die Fehler sehr klein, allerdings passt die Lösung nicht mehr so gut über die "wahre" Verteilung wie bei 0.001.

project_a3/reconstruction/unfolding/newton.py
2021-12-19T16:16+01:00

```
1  """Unfolding Newton"""
2  import numpy as np
3  from collections import namedtuple
4
5  from numpy.lib.function_base import gradient
6
7
8  def C_thikonov(n_dims):
9
10     main_diag = -2*np.ones(shape=(n_dims,))
11     main_diag[[0, -1]] = -1
12
13     second_diag = np.ones(shape=(n_dims-1,))
14
15     C = np.diag(main_diag, k=0)
16     C += np.diag(second_diag, k=-1)
17     C += np.diag(second_diag, k=1)
18
19     return C
20
21
22 def llh_gradient(A, g, f, tau=0.0):
23
24     m, n = A.shape
25     C = C_thikonov(n)
26
27     # not simplified version:
28     # gradient = []
29     # for k in range(n):
30     #     gradient_k = np.sum(-g * A[:,k] * (A @ f)**(-1) + A[:,k])
31     #     gradient_k += tau * np.sum(C[:,k] * (C @ f))
32     #     gradient.append(gradient_k)
33     # gradient = np.array(gradient)
34
35     gradient = A.T @ (np.ones(m) - g / (A @ f)) + tau * C.T @ (C @ f)
36
37     return gradient
38
39
40 def llh_hessian(A, g, f, tau=0.0):
```

```

41
42     m, n = A.shape
43     C = C_thikonov(n)
44
45     # not simplified version:
46     # hessian = []
47     # for k in range(n):
48     #     hessian_k = []
49     #     for l in range(n):
50     #         hessian_kl = np.sum(g * A[:,k] * A[:,l] * (A @ f)**(-2))
51     #         hessian_kl += tau * np.sum(C[:,k] * C[:,l])
52     #         hessian_k.append(hessian_kl)
53     #     hessian.append(hessian_k)
54     # hessian = np.array(hessian)
55
56     hessian = A.T @ np.diag(g / (A @ f)**2) @ A + tau * C.T @ C
57
58     return hessian
59
60
61 def minimize(
62     fun_grad, fun_hess, x0, prec=1e-10, max_iter=1000, epsilon=1e-6
63 ):
64     x = np.copy(x0)
65     output = namedtuple(
66         "minimization_result", ["x", "success", "hess_inv",
67         "n_iterations"]
68     )
69
70     # remember to set success to True if precision is reached
71     # and to set n_iterations accordingly
72
73     for k in range(max_iter):
74         gradient = fun_grad(x)
75         hessian = fun_hess(x) + epsilon*np.identity(n=len(x))
76         hessian_inv = np.linalg.inv(hessian)
77
78         x_old = x
79         x = x - hessian_inv @ gradient
80
81         if np.linalg.norm(x - x_old) < prec:
82             success = True
83             n_iterations = k+1
84             break

```

```
84     else:
85         success = False
86         n_iterations = max_iter
87
88     return output(x=x, success=success, hess_inv=hessian_inv,
89                   n_iterations=n_iterations)
```