Минимизация суммы квадратов

Сергей Кривохатский

СПбГУ, Современное программирование

21 октября 2022 г.

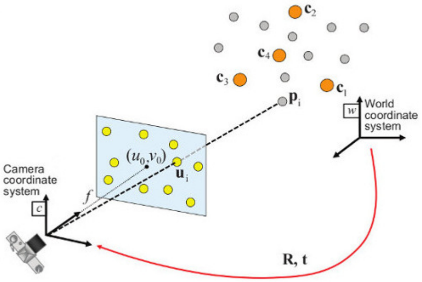
PnP

При наличии начального приближения задачу PnP можно решать путем минимизации суммы квадратов

$$\sum_i \left\| \pi(r, t, X_i) - x_i \right\|^2 \to \min$$

где

- r, t — позиция камеры
- x_i и X_i — 2D- и 3D-точки
- π — функция проекции

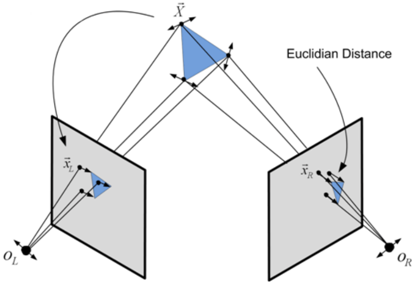


Bundle Adjustment

Качество восстановления сцены можно значительно улучшить путем оптимизации позиций камеры и координат 3D-точек

$$\sum_{j} \sum_{i \in V_j} \left\| \pi(r_j, t_j, X_i) - x_{ij} \right\|^2 \to \min$$

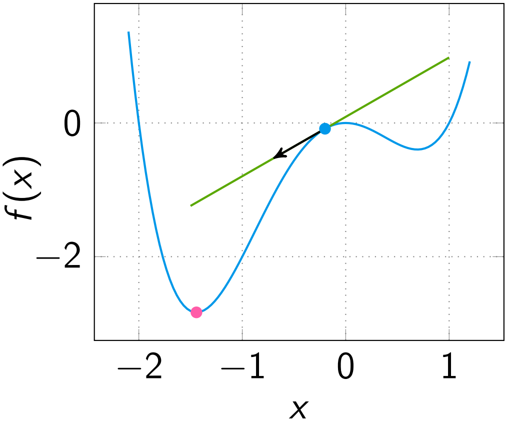
- r_j, t_j — позиция камеры в кадре j
- X_i — 3D-позиция i-й точки
- V_j — номера видимых в кадре j точек
- x_i — 2D-позиция i-й точки в кадре j
- π — функция проекции



Оптимизация на примере одномерной функции

Задача оптимизации

- дана $f : \mathbb{R} \to \mathbb{R}$
- хотим найти $x_{\min} = \arg\min_x f(x)$
- знаем начальное приближение x_0



Градиентный спуск

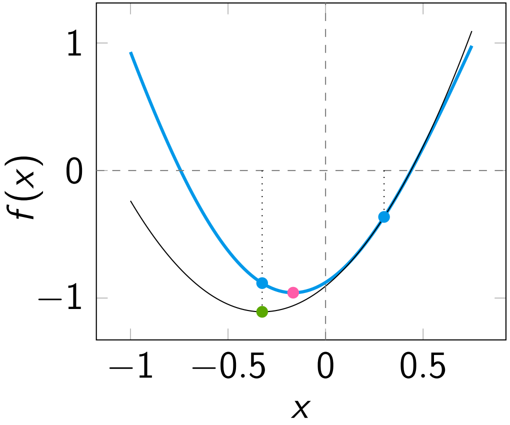
- повторяем $x_{k+1} = x_k - \alpha_{k+1} f'(x_k)$
- пока $\left|f'(x_k)\right| \geqslant \varepsilon$ (например)

Метод Ньютона для одномерной функции

Аппроксимируем $f(x)$ параболами и ходим в минимумы

$$f(x_k + \delta) \approx f(x_k) + f'(x_k)\delta + \frac{1}{2}f''(x_k)\delta^2$$

При определенных условиях скорость сходимости квадратичная



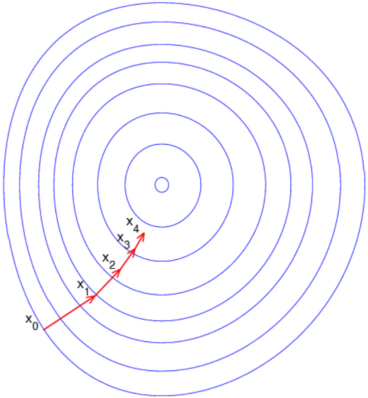
Оптимизация многомерной функции

Задача оптимизации

- дана $f : \mathbb{R}^n \to \mathbb{R}$
- хотим найти $x_{\min} = \arg\min_x f(x)$
- знаем начальное приближение x_0

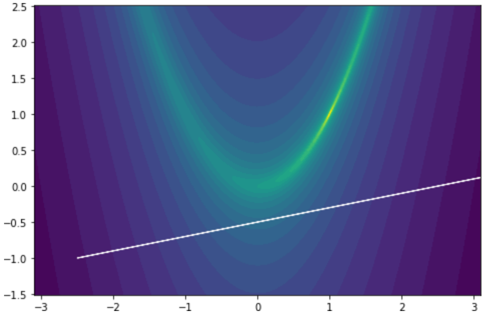
Градиентный спуск

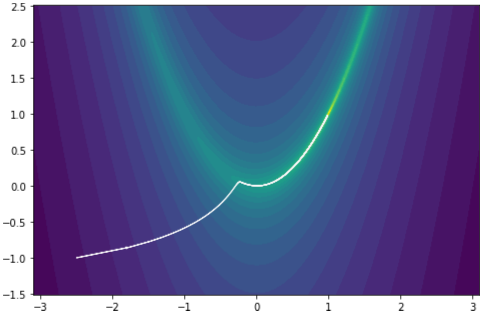
- повторяем $x_{k+1} = x_k - \alpha_{k+1} \nabla f(x_k)$
- пока $\left\| \nabla f(x_k) \right\| \geqslant \varepsilon$ (например)



Градиентный спуск

- обычно требует много коротких шагов, чтобы сойтись
- стоимость вычисления одного шага относительно небольшая





Метод Ньютона для многомерной функции I

- известна позиция x_k
- приближаем f в точке x_k квадратичной функцией

$$f(x_k+\delta) \approx f(x_k) + \delta^\intercal \nabla f(x_k) + \frac{1}{2}\delta^\intercal \boldsymbol{H}_f(x_k)\delta =: f_k(\delta)$$

- вычисляем $\delta_k = \arg\min_\delta f_k(\delta)$
- следующая позиция: $x_{k+1} = x_k + \delta_k$
- повторяем, пока не выполнится условие остановки

Метод Ньютона для многомерной функции II

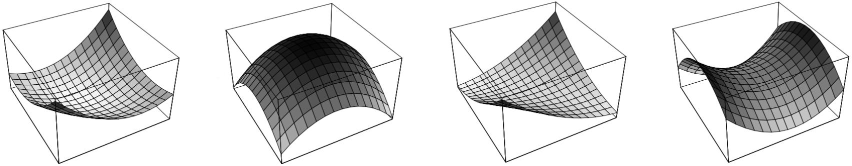
$$f_k(\delta) := f(x_k) + \delta^{\mathsf{T}} \nabla f(x_k) + \frac{1}{2}\delta^{\mathsf{T}} \boldsymbol{H}_f(x_k)\delta$$

- $\nabla f(x_k) = \begin{pmatrix} \dfrac{\partial f(x_k)}{\partial x_1} & \cdots & \dfrac{\partial f(x_k)}{\partial x_n} \end{pmatrix}^{\mathsf{T}} =: g_k$ — градиент
- $\boldsymbol{H}_f(x_k) = \begin{pmatrix} \dfrac{\partial^2 f(x_k)}{\partial x_1 \partial x_1} & \cdots & \dfrac{\partial^2 f(x_k)}{\partial x_1\, \partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f(x_k)}{\partial x_n\, \partial x_1} & \cdots & \dfrac{\partial^2 f(x_k)}{\partial x_n \partial x_n} \end{pmatrix} =: \boldsymbol{H}_k$ — матрица Гессе

Метод Ньютона для многомерной функции III

$$f_k(\delta) := f(x_k) + \delta^\intercal g_k + \frac{1}{2}\delta^\intercal \boldsymbol{H}_k \delta$$

- f_k имеет единственный минимум, если $\boldsymbol{H}_k$ — положительно определенная
- чтобы его найти, нужно решить СЛАУ: $\boldsymbol{H}_k \delta = -g_k$
- если $\boldsymbol{H}_k$ не положительно определенная, нам не повезло



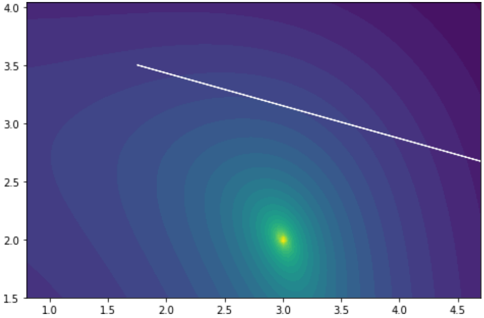
Метод Ньютона для многомерной функции IV

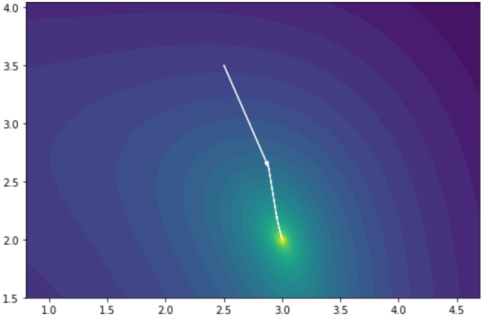
Получаем следующий алгоритм

- известна позиция x_k
- приближаем f в точке x_k квадратичной функцией $f_k(\delta)$
- вычисляем шаг, на котором f_k достигает минимума $\delta_k = -\boldsymbol{H}_k^{-1} g_k$
- следующая позиция: $x_{k+1} = x_k + \delta_k$
- повторяем, пока не выполнится условие остановки (какое-нибудь)

Метод Ньютона для многомерной функции V

- очень чувствителен к начальному приближению
- если все хорошо, очень быстро сходится
- стоимость вычисления одного шага относительно велика





Метод Гаусса — Ньютона I

- вычислять матрицу Гессе напрямую может быть сложно
- будем делать это приближенно
- мы знаем, что f имеет особый вид

$$f(x) = \sum_{i=1}^{m} r_i^2(x) = r(x)^\mathsf{T} r(x)$$

где $r(x) = \begin{pmatrix} r_1(x) & r_2(x) & \cdots & r_m(x) \end{pmatrix}^\mathsf{T}$

Метод Гаусса — Ньютона II

Распишем элемент матрицы Гессе

$$\frac{\partial^2 r^\intercal r}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\left(2r^\intercal \frac{\partial r}{\partial x_j}\right) = 2\frac{\partial r}{\partial x_i}^\intercal \frac{\partial r}{\partial x_j} + 2r^\intercal \frac{\partial^2 r}{\partial x_i \partial x_j}$$

Если элементы $r(x)$ достаточно малы либо близки к линейным функциям (т. е. их вторые производные малы)

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \approx 2\frac{\partial r}{\partial x_i}^\intercal \frac{\partial r}{\partial x_j}$$

Метод Гаусса — Ньютона III

Матрица Якóби функции $r(x)$

$$J_r(x) = \begin{pmatrix} \dfrac{\partial r(x)}{\partial x_1} & \cdots & \dfrac{\partial r(x)}{\partial x_n} \end{pmatrix} =$$

$$= \begin{pmatrix} \dfrac{\partial r_1(x)}{\partial x_1} & \cdots & \dfrac{\partial r_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial r_m(x)}{\partial x_1} & \cdots & \dfrac{\partial r_m(x)}{\partial x_n} \end{pmatrix}$$

Несложно заметить, что

$$H_f(x) \approx 2J_r(x)^\intercal J_r(x)$$

И что

$$\nabla f(x) = 2J_r(x)^\intercal r(x)$$

так как

$$\frac{\partial f}{\partial x_i} = 2\frac{\partial r}{\partial x_i}^\intercal r$$

Метод Гаусса — Ньютона IV

- обозначим $r_k := r(x_k)$ и $\boldsymbol{J}_k := \boldsymbol{J}_r(x_k)$
- f в точке x_k приближается как

$$f_k(\delta) \approx f(x_k) + \delta^\intercal g_k + \frac{1}{2}\delta^\intercal \boldsymbol{H}_k \delta \approx$$
$$\approx r_k^\intercal r_k + \delta^\intercal 2\boldsymbol{J}_k^\intercal r_k + \delta^\intercal \boldsymbol{J}_k^\intercal \boldsymbol{J}_k \delta$$

- шаг δ_k получается из решения линейной системы

$$\boldsymbol{J}_k^\intercal \boldsymbol{J}_k \delta_k = -\boldsymbol{J}_k^\intercal r_k$$

Метод Гаусса — Ньютона V

Получаем следующий алгоритм

- известна позиция x_k
- приближаем f в точке x_k квадратичной функцией: $f_k(\delta) \approx r_k^\intercal r_k + \delta^\intercal 2\boldsymbol{J}_k^\intercal r_k + \delta^\intercal \boldsymbol{J}_k^\intercal \boldsymbol{J}_k \delta$
- вычисляем шаг, минимизируя приближение: $\delta_k = -(\boldsymbol{J}_k^\intercal \boldsymbol{J}_k)^{-1} \boldsymbol{J}_k^\intercal r_k$
- следующая позиция: $x_{k+1} = x_k + \delta_k$
- повторяем, пока не выполнится условие остановки (какое-нибудь)

Алгоритм Левенберга — Марквардта I

Матрица $\boldsymbol{J}_k^\intercal \boldsymbol{J}_k$ может быть вырожденной. Левенберг в 1944 году предложил использовать регуляризацию $\boldsymbol{J}_k^\intercal \boldsymbol{J}_k + \lambda \mathbf{1}$

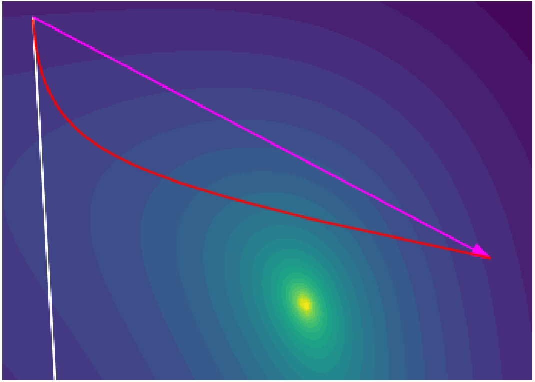
$$f_k(\delta) = f(x_k) + \delta^\intercal g_k + \delta^\intercal \boldsymbol{J}_k^\intercal \boldsymbol{J}_k \delta + \delta^\intercal \lambda \mathbf{1} \delta$$

Прибавляем параболоид с минимумом в x_k

Алгоритм Левенберга — Марквардта II

$$\delta_k = -(\boldsymbol{J}_k^\intercal \boldsymbol{J}_k + \lambda \mathbf{1})^{-1} g_k$$

- меньше λ — метод Гаусса — Ньютона
- больше λ — градиентный спуск с шагом $1/\lambda$



Алгоритм Левенберга — Марквардта III

- Марквардт в 1963 году предложил менять λ адаптивно от шага к шагу по следующему принципу

$$\lambda_{k+1} = \begin{cases} \frac{1}{\nu}\lambda_k & \text{если } f(x_k + \delta_k) < f(x_k) \\ \nu\lambda_k, & \text{иначе} \end{cases}$$

  где $\nu > 1$ — константа (часто берут 10)
- существует формулировка алгоритма подбора λ в терминах доверительного региона, которая обладает некоторыми преимуществами, см. *J. More, "The Levenberg-Marquardt Algorithm: Implementation and Theory", 1977*

Алгоритм Левенберга — Марквардта IV

- Флетчер в 1971 году предложил вместо $\mathbf{1}$ брать $\operatorname{diag} \boldsymbol{J}_k^\mathsf{T} \boldsymbol{J}_k$ — это улучшает сходимость в направлениях с малым абсолютным значением градиента
- может возникнуть проблема: если параметр x_i не влияет на функцию, матрица $\boldsymbol{J}_k^\mathsf{T} \boldsymbol{J}_k + \lambda_k \operatorname{diag} \boldsymbol{J}_k^\mathsf{T} \boldsymbol{J}_k$ становится вырожденной

Алгоритм Левенберга — Марквардта V

Шаг алгоритма Левенберга — Марквардта

$$\delta_k = -(\boldsymbol{J}_k^\mathsf{T} \boldsymbol{J}_k + \lambda_k \operatorname{diag} \boldsymbol{J}_k^\mathsf{T} \boldsymbol{J}_k)^{-1} \boldsymbol{J}_k^\mathsf{T} r_k$$

Вычислять обратную матрицу не стоит, лучше решить СЛАУ

$$(\boldsymbol{J}_k^\mathsf{T} \boldsymbol{J}_k + \lambda_k \operatorname{diag} \boldsymbol{J}_k^\mathsf{T} \boldsymbol{J}_k)\delta_k = -\boldsymbol{J}_k^\mathsf{T} r_k$$

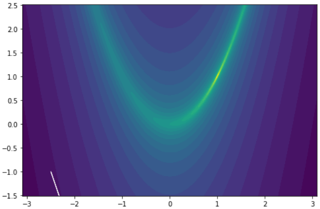
Матрица $\boldsymbol{J}_k$ часто имеет специфическую структуру, что может позволить подобрать наиболее эффективный метод решения, оптимизировать потребление памяти и т. д.

Алгоритм Левенберга — Марквардта VI

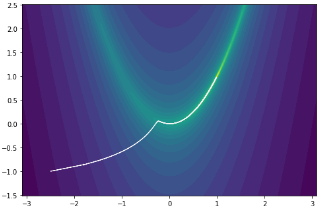
Получаем следующую схему алгоритма

- известна позиция x_k
- вычисляем шаг, решая систему уравнений $(\boldsymbol{J}_k^\mathsf{T}\boldsymbol{J}_k + \lambda_k \operatorname{diag} \boldsymbol{J}_k^\mathsf{T}\boldsymbol{J}_k)\delta_k = -\boldsymbol{J}_k^\mathsf{T} r_k$
- если $f(x_{k+1} + \delta_k) \geqslant f(x_k)$, увеличиваем λ, следующая позиция: $x_{k+1} = x_k$
- иначе уменьшаем значение λ, следующая позиция: $x_{k+1} = x_k + \delta_k$
- повторяем, пока не выполнится условие остановки (какое-нибудь)

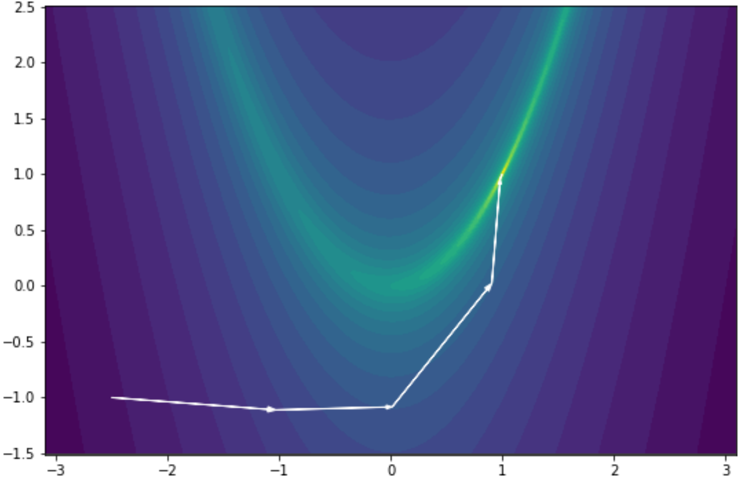
Алгоритм Левенберга — Марквардта VII



Метод Гаусса — Ньютона



Градиентный спуск



Метод Левенберга — Марквардта

Робастное оценивание на примере PnP I

- решение задачи PnP путем оптимизации

$$\arg\min_{r,t} \sum_i \left\| \pi(r, t, X_i) - x_i \right\|^2$$

- для удобства изменим обозначения

$$\sum_i \left\| \pi(r, t, X_i) - x_i \right\|^2 = \sum_i r_i^2(p) = r(p)^\mathsf{T} r(p)$$

- предположим, что при известной позиции камеры ошибки репроекции распределены нормально и независимо

$$r_i(p) \sim \mathcal{N}(0, \sigma)$$

Робастное оценивание на примере PnP II

- тогда позицию камеры можно найти с помощью метода максимального правдоподобия

$$\arg\max_p \prod_i \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{r_i(p)^2}{2\sigma^2}\right)$$

- если прологарифмировать, умножить на -1 и избавиться от констант, получим сумму квадратов

$$\arg\min_p \sum_i r_i^2(p)$$

Робастное оценивание на примере PnP III

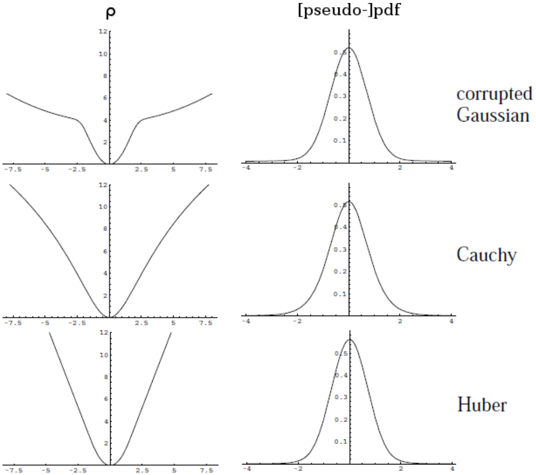
- выбросы среди 2D-точек могут заметно исказить результат
- для решения этой проблемы можно заменить квадратичную функцию на другую, которая даст более робастную оценку

$$\arg\min_{p} \sum_{i} \rho(r_i(p))$$

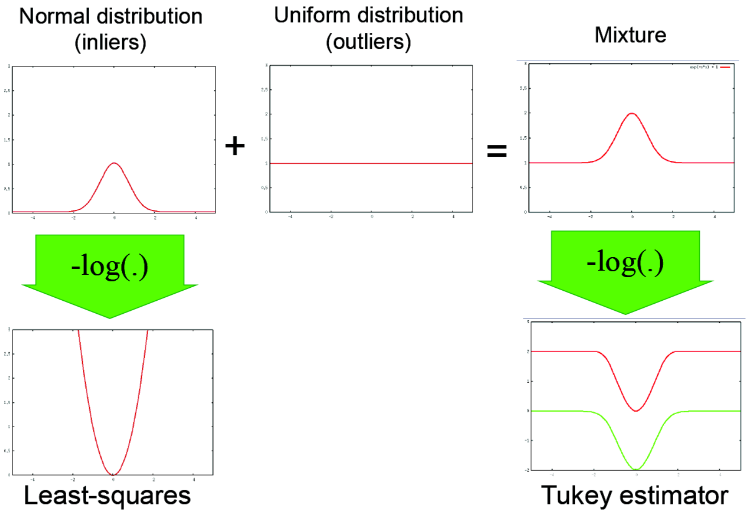
- такие оценки называются *оценками типа максимального правдоподобия* или *M-оценками*

M-оценки I

- ρ подбирается исходя из предположений о распределении ошибок измерений (2D-точек) или из эвристических соображений
- ρ может соответствовать «честной» плотности распределения измерений или «нечестной» (интеграл не равен единице, например)



М-оценки II



Normal distribution
(inliers)
+
Uniform distribution
(outliers)
=
Mixture
-log(.)
-log(.)
Least-squares
Tukey estimator

M-оценки III

- какие-то M-оценки (например, Тьюки) являются невыпуклыми, из-за чего могут возникнуть проблемы со сходимостью к глобальному минимуму, если начальное приближение плохое
- другие M-оценки (например, Хьюбера) являются выпуклыми и не имеют такого негативного влияния на сходимость
- на практике иногда полезно прогнать оптимизацию несколько раз: например, сначала с M-оценкой Хьюбера, потом с Тьюки
- *A General and Adaptive Robust Loss Function (CVPR 2019)*

М-оценки. Оптимизация I

- ρ — М-оценка
- $\psi := \rho'$ — функция влияния
- $w(e) := \dfrac{\psi(e)}{e}$ — весовая функция

$$f(p) := r(p)^\intercal r(p)$$

$$f_\rho(p) := \sum_{k=0}^{m} \rho(r_k(p))$$

M-оценки. Оптимизация II

Продифференцируем $\rho(r_k(p))$ по p_i

$$\frac{\partial \rho(r_k(p))}{\partial p_i} = \psi(r_k(p))\frac{\partial r_k(p)}{\partial p_i} = $$
$$= w(r_k(p))r_k(p)\frac{\partial r_k(p)}{\partial p_i}$$

$$\psi := \rho'$$
$$w(e) := \frac{\psi(e)}{e}$$

Значит градиент можно выразить таким образом

$$\nabla f_\rho(p) = \boldsymbol{J}_r^\intercal(p)\boldsymbol{W}_p r(p)$$

($\boldsymbol{W}_p$ — диагональная матрица с весами в точке p)

М-оценки. Оптимизация III

Продифференцируем еще раз, теперь по $\boldsymbol{p}_j$

$$\frac{\partial}{\partial \boldsymbol{p}_j}\left(\psi(r_k(p))\frac{\partial r_k(p)}{\partial \boldsymbol{p}_i}\right) = \frac{\partial r_k(p)}{\partial \boldsymbol{p}_j}\psi'(r_k(p))\frac{\partial r_k(p)}{\partial \boldsymbol{p}_i} + \psi(r_k(p))\frac{\partial^2 r_k(p)}{\partial \boldsymbol{p}_i \partial \boldsymbol{p}_j}$$

Значит матрицу Гессе можно приблизить таким образом

$$\boldsymbol{H}_{f_\rho}(p) \approx \boldsymbol{J}_r^\intercal(p)\,\mathrm{diag}\left(\psi'(r_1(p)), \psi'(r_2(p)), \dots\right)\boldsymbol{J}_r(p)$$

Но на практике обычно делают даже так

$$\boldsymbol{H}_{f_\rho}(p) \approx \boldsymbol{J}_r^\intercal(p)\boldsymbol{W}_p\boldsymbol{J}_r(p)$$

M-оценки. Оптимизация IV

- минимизацию $\sum \rho(r_k(p))$ сводят к шагам с минимизацией взвешенных сумм квадратов $r(p)^\intercal \boldsymbol{W}_p r(p)$
- на каждом шаге оптимизации производят перевзвешивание, т. е. пересчитывают весовую матрицу $\boldsymbol{W}$
- на практике параметры M-оценки (т. е. точная форма функции ρ) обычно переоцениваются на каждом шаге на основе текущего распределения r_k

Спасибо за внимание!