Descent Methods

Part II Roadmap

- Part I – Linear Algebra (units 1-12) $Ac = b$

linearize

line search

- Part II – Optimization (units 13-20)
  - (units 13-16) Optimization -> Nonlinear Equations -> 1D roots/minima
  - (units 17-18) Computing/Avoiding Derivatives
  - (unit 19) Hack 1.0: "I give up" $H = I$ and J is mostly 0 (descent methods)
  - (unit 20) Hack 2.0: "It's an ODE!?" (adaptive learning rate and momentum)

Theory

Methods

Recall: Gradient (Unit 9)

- Consider the scalar (output) function $f(c)$ with multi-dimensional input c
- The Jacobian of $f(c)$ is $J(c) = \left(\frac{\partial f}{\partial c_1}(c) \quad \frac{\partial f}{\partial c_2}(c) \quad \cdots \quad \frac{\partial f}{\partial c_n}(c)\right)$
- The gradient of $f(c)$ is $\nabla f(c) = J^T(c) = \begin{pmatrix} \frac{\partial f}{\partial c_1}(c) \\ \frac{\partial f}{\partial c_2}(c) \\ \vdots \\ \frac{\partial f}{\partial c_n}(c) \end{pmatrix}$
- In 1D, both $J(c)$ and $\nabla f(c) = J^T(c)$ are the usual $f'(c)$

Gradient/Steepest Descent

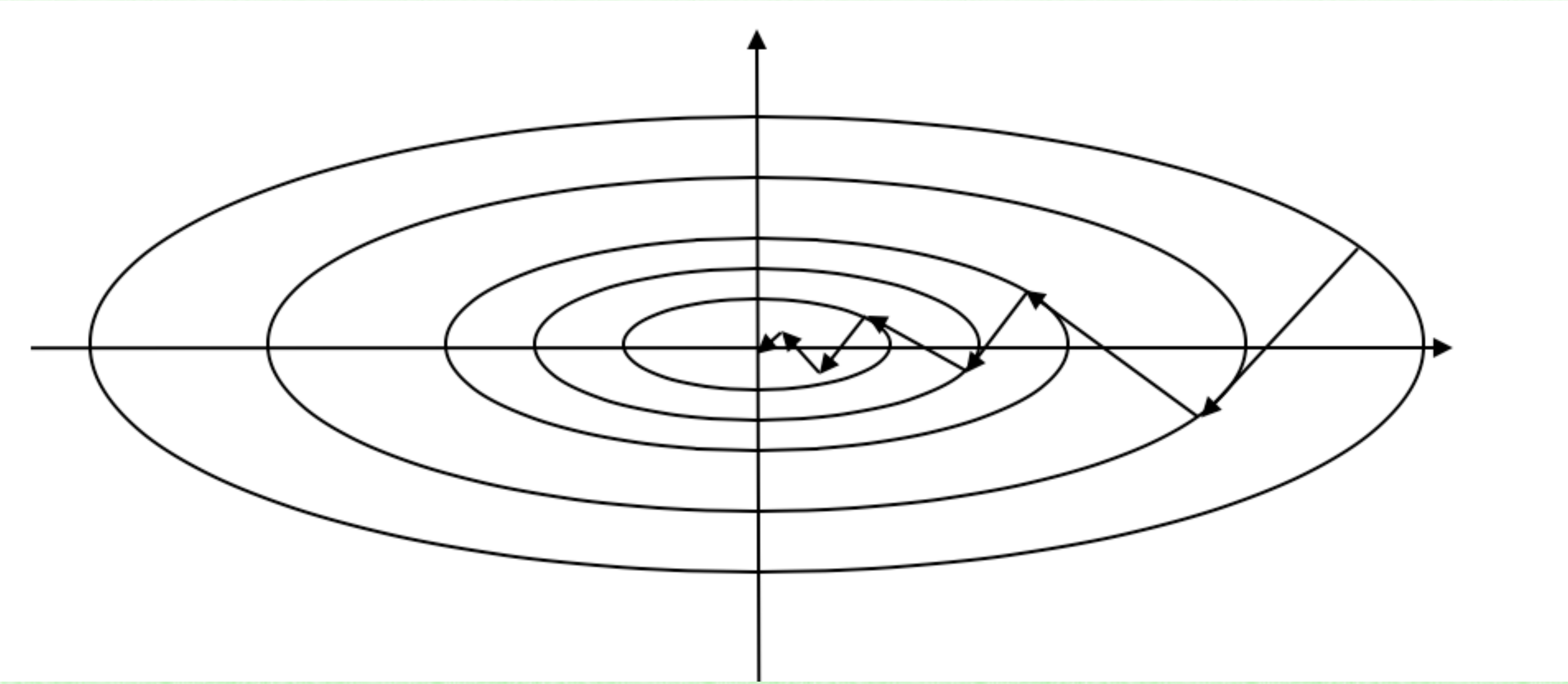
- Given a cost function $\hat{f}(c)$
  - $\nabla\hat{f}(c)$ is the direction in which $\hat{f}(c)$ increases the fastest
  - $-\nabla\hat{f}(c)$ is the direction in which $\hat{f}(c)$ decreases the fastest
- Thus, $-\nabla\hat{f}(c)$ is considered the <u>direction of steepest descent</u>
- Using $-\nabla\hat{f}(c)$ as the search direction is known as <u>steepest descent</u>
  - This can be thought of as always "walking in the steepest downhill direction"
  - However, never going uphill can lead to local minima
- Methods that use $-\nabla\hat{f}(c)$ in various ways are known as <u>gradient descent</u> methods

- Recall (Unit 18) approximating $H_{\hat{f}}^T \approx I$ in $H_{\hat{f}}^T(c^q)\Delta c^q = -J_{\hat{f}}^T(c^q)$ leads to steepest descent: $\Delta c^q = -J_{\hat{f}}^T(c^q) = -\nabla\hat{f}(c^q)$

Steepest Descent for Quadratic Forms

- Recall (Unit 9):
  - The <u>Quadratic Form</u> of a SPD $\tilde{A}$ is $f(c) = \frac{1}{2} c^T \tilde{A} c - \tilde{b}^T c + \tilde{c}$
  - Minimize $f(c)$ by finding critical points where $\nabla f(c) = \tilde{A}c - \tilde{b} = 0$
  - That is, solve $\tilde{A}c = \tilde{b}$ to find the critical point
- Recall (Unit 5):
  - Steepest descent search direction: $-\nabla f(c) = \tilde{b} - \tilde{A}c = r$
  - $r^q = b - Ac^q, \alpha^q = \frac{r^q \cdot r^q}{r^q \cdot A r^q}, c^{q+1} = c^q + \alpha^q r^q$ is iterated until r^q is small enough
  - The main drawback to steepest descent is that it repeatedly searches in the same directions too often, especially for higher condition number matrices
  - Because it takes far too long for steepest descent to converge, we instead advocated for Conjugate Gradients

Steepest Descent for Quadratic Forms



CG would (instead) solve this in 2 steps

Recall: Nonlinear Least Squares (Unit 18)

- Recall from Unit 13:
  - Determine parameters c that make $f(x, y, c) = 0$ best fit the training data, i.e. that make $\|f(x_i, y_i, c)\|_2^2 = f(x_i, y_i, c)^T f(x_i, y_i, c)$ close to zero for all i
  - Combining all (x_i, y_i), minimize $\hat{f}(c) = \frac{1}{2}\sum_i f(x_i, y_i, c)^T f(x_i, y_i, c)$

- Let m be the number of data points and $\widehat{m}$ be the output size of $f(x, y, c)$
- Define $\tilde{f}(c)$ by stacking the $\widehat{m}$ outputs of $f(x, y, c)$ consecutively m times, so that the vector valued output of $\tilde{f}(c)$ is length $m * \widehat{m}$
- Then, $\hat{f}(c) = \frac{1}{2}\sum_i f(x_i, y_i, c)^T f(x_i, y_i, c) = \frac{1}{2}\tilde{f}^T(c)\tilde{f}(c)$

Recall: Nonlinear Least Squares (Unit 18)

- Minimize $\hat{f}(c) = \frac{1}{2}\tilde{f}^T(c)\tilde{f}(c)$

- Jacobian matrix of $\tilde{f}$ is $J_{\tilde{f}}(c) = \begin{pmatrix} \frac{\partial \tilde{f}}{\partial c_1}(c) & \frac{\partial \tilde{f}}{\partial c_2}(c) & \cdots & \frac{\partial \tilde{f}}{\partial c_n}(c) \end{pmatrix}$

- Critical points of $\hat{f}(c)$ have $J_{\hat{f}}^T(c) = \begin{pmatrix} \tilde{f}^T(c)\frac{\partial \tilde{f}}{\partial c_1}(c) \\ \tilde{f}^T(c)\frac{\partial \tilde{f}}{\partial c_2}(c) \\ \vdots \\ \tilde{f}^T(c)\frac{\partial \tilde{f}}{\partial c_n}(c) \end{pmatrix} = J_{\tilde{f}}^T(c)\tilde{f}(c) = 0$

Steepest Descent for Nonlinear Least Squares

- Search direction $-\nabla\hat{f}(c) = -J_{\hat{f}}^T(c) = -J_{\tilde{f}}^T(c)\tilde{f}(c) = \begin{pmatrix} -\tilde{f}^T(c)\dfrac{\partial\tilde{f}}{\partial c_1}(c) \\ -\tilde{f}^T(c)\dfrac{\partial\tilde{f}}{\partial c_2}(c) \\ \vdots \\ -\tilde{f}^T(c)\dfrac{\partial\tilde{f}}{\partial c_n}(c) \end{pmatrix}$
- Recall that $\tilde{f}(c)$ is constructed by stacking the $\widehat{m}$ outputs of $f(x_i, y_i, c)$ consequtively m times, once for each data point (x_i, y_i)
- Thus, each of the n terms of the form $-\tilde{f}^T(c)\dfrac{\partial\tilde{f}}{\partial c_k}(c)$ is a (potentially expensive) sum through $m * \widehat{m}$ terms (recall: m is the amount of training data)

Descent Options for Nonlinear Least Squares

- When there is a lot of data, m can be extremely large
  - This is exacerbated when the $\frac{\partial \tilde{f}}{\partial c_k}$ are expensive to compute

- Using all the data is called Batch Gradient Descent

- When only a (typically small) subset of the data is used to compute the search direction (ignoring the rest of the data), this is called Mini-Batch Gradient Descent

- When only a single data point is used to compute the search direction (chosen randomly/sequentially), this is called Stochastic Gradient Descent (SGD)