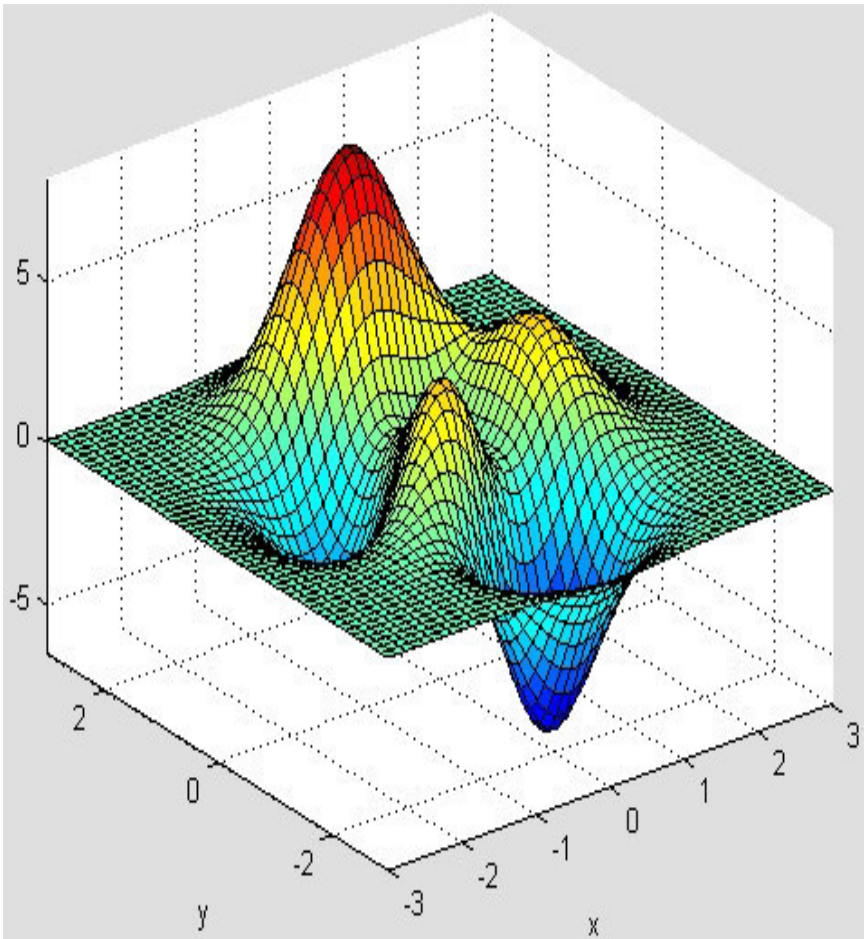
Unconstrained Nonlinear Optimisation

Georgy Gimel'farb

COMPSCI 369 Computational Science

Nature is nonlinear...

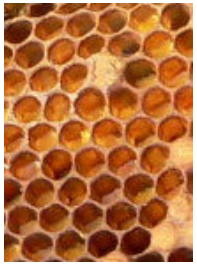
4896kj.com/journeying/wp-content/uploads /2007/06/

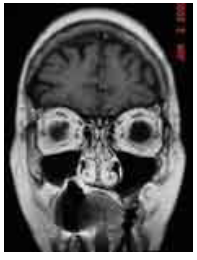


www.mathworks.com/matlabcentral/fx_files/

Why Optimisation?

- Many physical laws are formulated in terms of maximum or minimum of energy, entropy, or other object properties
- Optimisation is essential almost everywhere (in natural life, science, engineering, economics, business, etc).
  - Bees minimise the average amount of material per cell
  - Humans aim to the maximum effect under limited resources
- Many scientific and technological applications require large-scale optimisation with $10^4..10^6$ or more variables:
  - Medical imaging and computer assisted diagnostics
  - Shape design of mechanical objects
  - Control of industrial processes
  - Robotics and autonomous navigation, etc.

Extremum of a Function

One of most important applied problems: to find maximum or minimum value of a function $f(\mathbf{x})$ under constraints $\mathbf{x} \in \mathbf{X}$

- $f(\mathbf{x}) \equiv f(x_1, \ldots, x_n)$ is a scalar function of n-dimensional vector argument
- $\mathbf{X}$ is a certain subset of n-dimensional vector space $\mathbb{R}_n$

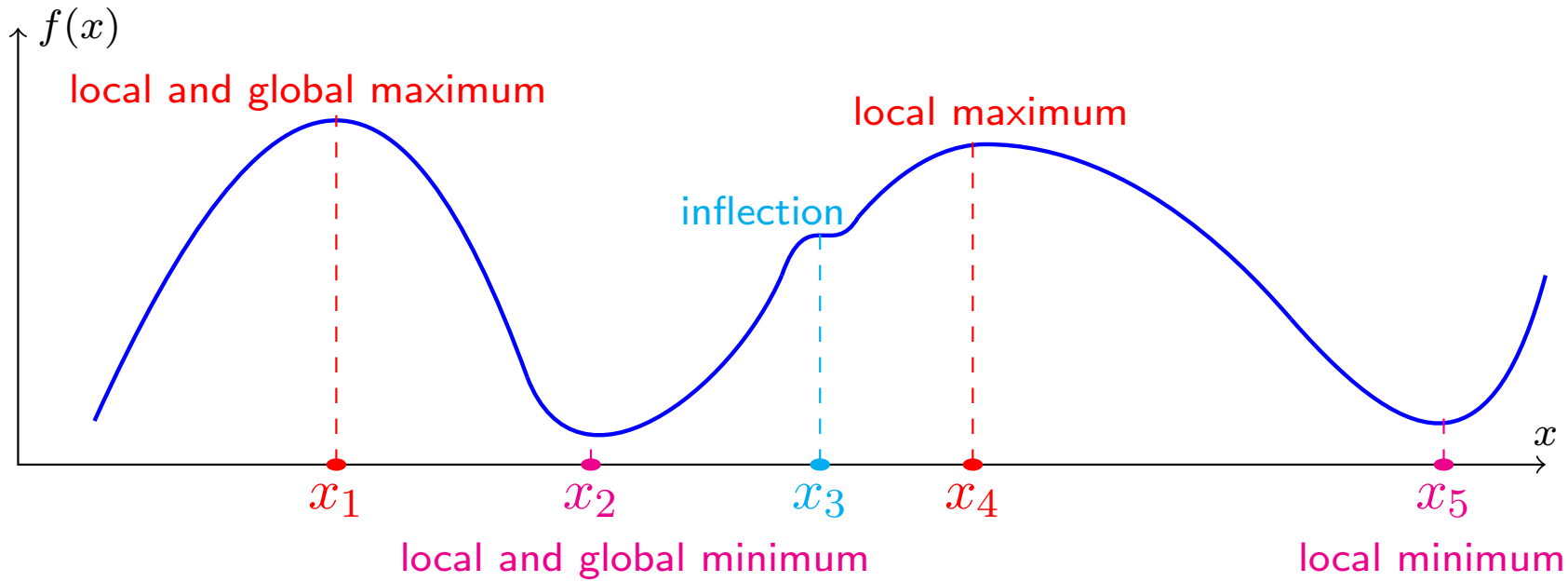
Unconstrained optimisation: if $\mathbf{X} = \mathbb{R}_n$

- Maximum / minimum function value: $f^* = \left\{\begin{matrix}\max\\ \min\end{matrix}\right\} f(\mathbf{x})$
- Maximiser / minimiser: $\mathbf{x}^* = \arg\left\{\begin{matrix}\max\\ \min\end{matrix}\right\} f(\mathbf{x})$

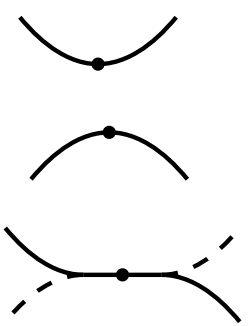
Constrained optimisation: if $\mathbf{X} \subset \mathbb{R}_n$

- Maximum / minimum function value: $f^* = \left\{\begin{matrix}\max\\ \min\end{matrix}\right\}_{\mathbf{x}\in\mathbf{X}} f(\mathbf{x})$
- Maximiser / minimiser: $\mathbf{x}^* = \arg\left\{\begin{matrix}\max\\ \min\end{matrix}\right\}_{\mathbf{x}\in\mathbf{X}} f(\mathbf{x})$

Univariate Function (Function of One Variable)



- Minimum: $\frac{df(x)}{dx} = 0;\ \frac{d^2f(x)}{dx^2} > 0$
- Maximum: $\frac{df(x)}{dx} = 0;\ \frac{d^2f(x)}{dx^2} < 0$
- Inflection point: $\frac{df(x)}{dx} = 0;\ \frac{d^2f(x)}{dx^2} = 0$



Functions of Many Variables $f(\mathbf{x})$

- Unconditional local critical (or stationary) point: where the gradient of f is zero: $\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n}\right]^{\mathsf{T}} = \mathbf{0}$
- Its properties depend on the Hessian matrix of the 2^{nd} derivatives:

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{bmatrix}$$

  - Local minimum: if the Hessian is positive definite (the quadratic form $\mathbf{e}^{\mathsf{T}}\mathbf{H}(\mathbf{x})\mathbf{e} > 0$ for any $\mathbf{e} \neq \mathbf{0}$)
  - Local maximum: if the Hessian is negative definite (the quadratic form $\mathbf{e}^{\mathsf{T}}\mathbf{H}(\mathbf{x})\mathbf{e} < 0$ for any $\mathbf{e} \neq \mathbf{0}$)
  - Saddle point: if the Hessian is indefinite

Quadratic Function $f(\mathbf{x}) = \mathbf{a}^\mathsf{T}\mathbf{x} + \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{H}\mathbf{x}$ of n Variables

$$f(\mathbf{x}) = \overbrace{a_1x_1 + \ldots + a_nx_n}^{\mathbf{a}^\mathsf{T}\mathbf{x}} + \overbrace{\frac{1}{2}\left(\begin{array}{l} H_{11}x_1^2 \quad + H_{12}x_1x_2 + \ldots + H_{1n}x_1x_n \\ + H_{21}x_2x_1 + H_{22}x_2^2 \quad + \ldots + H_{2n}x_2x_n \\ \ldots + H_{n1}x_nx_1 + H_{n2}x_nx_2 + \ldots + H_{nn}x_n^2 \end{array}\right)}^{\frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{H}\mathbf{x}}$$

$$= \sum_{i=1}^{n}\left(a_ix_i + \frac{H_{ii}}{2}x_i^2 + \sum_{j=i+1}^{n} H_{ij}x_ix_j\right) \quad \text{where} \quad H_{ij} = H_{ji}$$

- Gradient $\nabla f(\mathbf{x}) = \mathbf{a} + \mathbf{H}\mathbf{x}$:

$$\begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} H_{11} & H_{12} & \ldots & H_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ H_{n1} & H_{n2} & \ldots & H_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

- Hessian $\frac{\partial \nabla f(\mathbf{x})}{\partial \mathbf{x}} \equiv \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{i,j=1}^{n} \equiv [H_{ij}]_{i,j=1}^{n} \equiv \mathbf{H}$

Sylvester's Criterion: Sufficient Condition of an Extremum

Symmetric $n \times n$ matrix $\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}$ is:

- Positive definite if the determinants of all its upper-left submatrices are positive:

$$a_{11} > 0; \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} > 0; \dots; \begin{vmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{vmatrix} > 0$$

- Negative definite if sign-alternate (starting from $a_{11} < 0$):

$$-a_{11} > 0; \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} > 0; \dots; (-1)^n \begin{vmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{vmatrix} > 0$$

- Indefinite in all other cases

Quadratic Function: An Example with No Local Extremum

$$f(x_1, x_2, x_3) = 5x_1 - 3x_2 + x_3 + 4x_1^2 - 2x_1x_2 + 10x_1x_3 - x_2^2 + 3x_2x_3 + 9x_3^2$$

$$\frac{\partial f}{\partial x_1} = 5 + 8x_1 - 2x_2 + 10x_3 \qquad \frac{\partial^2 f}{\partial x_1^2} = 8 \qquad \frac{\partial^2 f}{\partial x_1 \partial x_2} = -2 \qquad \frac{\partial^2 f}{\partial x_1 \partial x_3} = 10$$

$$\frac{\partial f}{\partial x_2} = -3 - 2x_1 - 2x_2 + 3x_3 \qquad \frac{\partial^2 f}{\partial x_1 \partial x_2} = -2 \qquad \frac{\partial^2 f}{\partial x_2^2} = -2 \qquad \frac{\partial^2 f}{\partial x_2 \partial x_3} = 3$$

$$\frac{\partial f}{\partial x_3} = 1 + 10x_1 + 3x_2 + 18x_3 \qquad \frac{\partial^2 f}{\partial x_1 \partial x_3} = 10 \qquad \frac{\partial^2 f}{\partial x_2 \partial x_3} = 3 \qquad \frac{\partial^2 f}{\partial x_3^2} = 18$$

Gradient:

$$\nabla f(x_1, x_2, x_3) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 5 \\ -3 \\ 1 \end{bmatrix} + \underbrace{\begin{bmatrix} 8 & -2 & 10 \\ -2 & -2 & 3 \\ 10 & 3 & 18 \end{bmatrix}}_{\text{the indefinite Hessian } \mathbf{H}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

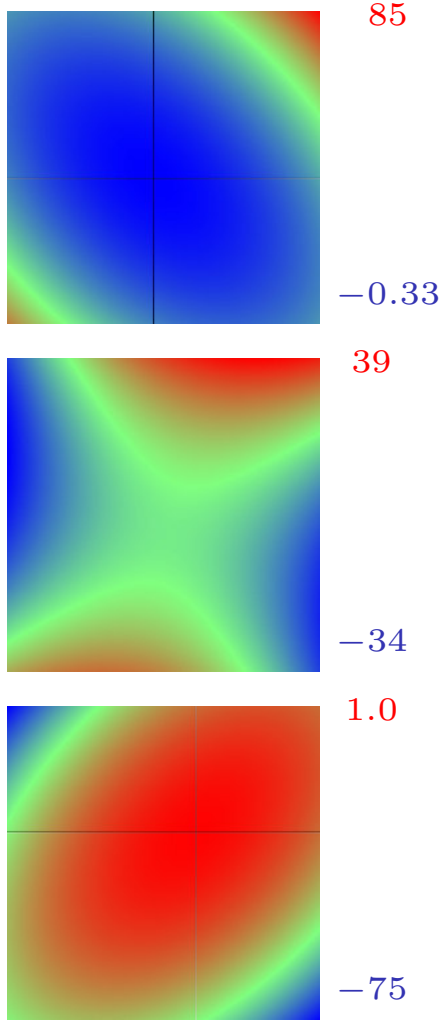
Sylvester's criterion:

$$H_{11} = 8 > 0; \quad \begin{vmatrix} 8 & -2 \\ -2 & -2 \end{vmatrix} = -12 < 0; \quad \begin{vmatrix} 8 & -2 & 10 \\ -2 & -2 & 3 \\ 10 & 3 & 18 \end{vmatrix} = -120 < 0$$

Quadratic Functions of Two Variables

- $f(x,y) = x^2 - xy + y^2 + x - y$:
  the positive definite Hessian $\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$ as
  $2 > 0; \begin{vmatrix} 2 & 1 \\ 1 & 2 \end{vmatrix} = 2 \cdot 2 - 1 \cdot 1 = 4 - 1 = 3 > 0$
- $f(x,y) = -x^2 - xy + y^2 + x - y$:
  the indefinite Hessian $\begin{bmatrix} -2 & -1 \\ -1 & 2 \end{bmatrix}$ as
  $-2 < 0; \begin{vmatrix} -2 & 1 \\ 1 & 2 \end{vmatrix} = -2 \cdot 2 - 1 \cdot 1 = -4 - 1 = -5 < 0$
- $f(x,y) = --x^2 - xy - y^2 + x - y$:
  the negative definite Hessian $\begin{bmatrix} -2 & -1 \\ -1 & -2 \end{bmatrix}$ as
  $-2 < 0;$
  $\begin{vmatrix} -2 & -1 \\ -1 & -2 \end{vmatrix} = -2 \cdot (-2) - (-1) \cdot (-1) = 4 - 1 = 3 > 0$

Colour-coded range
$[f_{\min}, f_{\max}]$



Equivalent Definitions of a Positive Definite Matrix

Symmetric $n \times n$ matrix $\mathbf{A}$ is positive definite if

1. All eigenvalues of $\mathbf{A}$ are positive (> 0), or
2. Choleski decomposition $\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$ exists where $\mathbf{L}$ is a lower triangular matrix with $l_{ii} > 0$, or
3. Decomposition $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^{\mathsf{T}}$ exists where $\mathbf{L}$ is a lower triangular matrix with $l_{ii} = 1$ and $\mathbf{D}$ is a diagonal matrix with $d_i > 0$, or
4. All pivots in Gaussian elimination without pivoting are positive (> 0).

For small matrices, the Sylvester's condition can be readily checked, but generally Conditions 2 or 3 are the most efficient and yield easy solutions to linear systems with coefficients $\mathbf{A}$

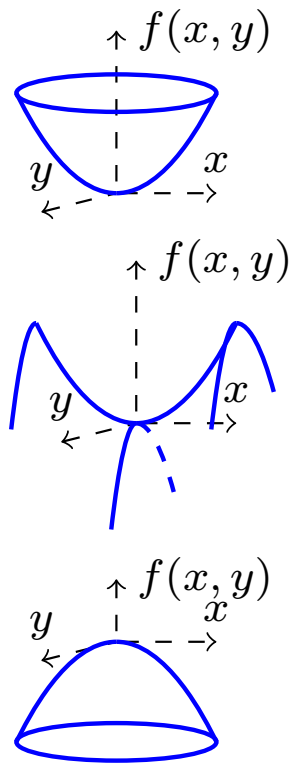
Extrema of $f(x, y) = ax^2 + by^2;\ a \neq 0;\ b \neq 0$

Gradient $\nabla f(x, y) = 0 \Rightarrow \frac{\partial f(x,y)}{\partial x} = 2ax = 0; \frac{\partial f(x,y)}{\partial y} = 2by = 0$

$\Rightarrow$ Single extremum at the point $[0, 0]^\mathsf{T}$

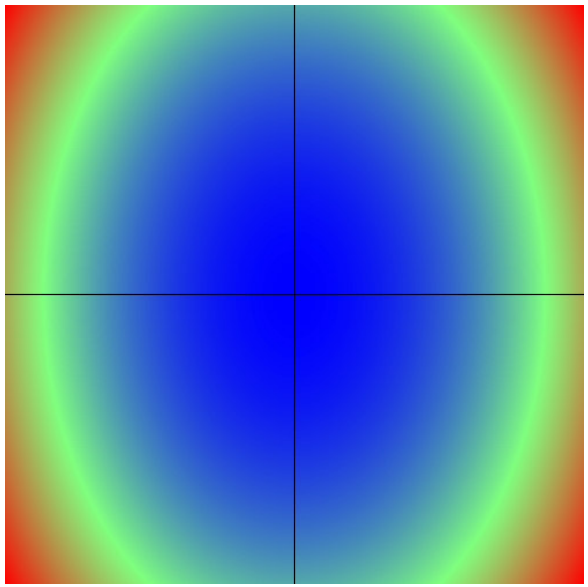
Hessian $\mathbf{H} = \begin{bmatrix} 2a & 0 \\ 0 & 2b \end{bmatrix}$

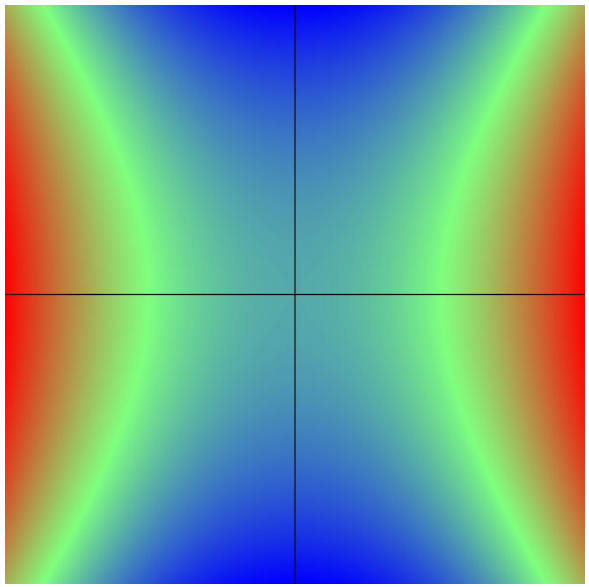
- Function $f(x, y)$ has the minimum in $[0, 0]^\mathsf{T}$ if $a > 0$ and $b > 0$: an elliptic paraboloid (the Sylvester's criterion: $2a > 0$ and $4ab > 0$)
- If $a > 0;\ b < 0$ or $a < 0;\ b > 0$, there is no extremum: a hyperbolic paraboloid with a saddle point $[0, 0]^\mathsf{T}$ (the Sylvester's criterion: $2a > 0$ and $4ab < 0$ or $2a < 0$ and $4ab < 0$)
- Function $f(x, y)$ has the maximum in $[0, 0]^\mathsf{T}$ if $a < 0$ and $b < 0$: an elliptic paraboloid (the Sylvester's criterion: $2a < 0$ and $4ab > 0$)

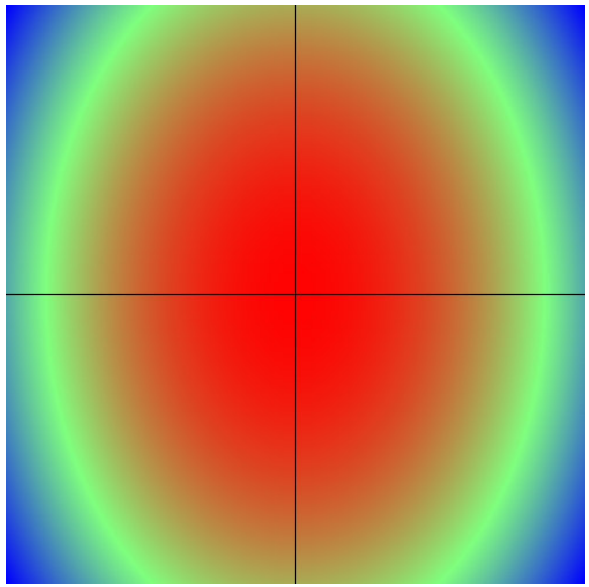


Colour-Coded Graphs of $f(x, y) = ax^2 + by^2;\ a \neq 0;\ b \neq 0$

Function range $[0.0, 3.75]$ $[-1.25, 2.5]$ $[-3.75, 0.0]$



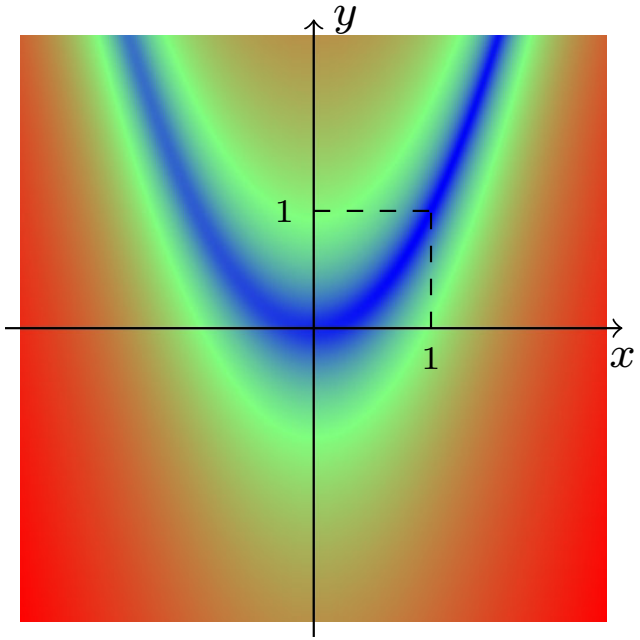




$f(x, y) = 0.1x^2 + 0.05y^2$ $f(x, y) = 0.1x^2 - 0.05y^2$ $f(x, y) = -0.1x^2 - 0.05y^2$

- The single potentially extremum point $[x = 0, y = 0]$ such that $\nabla f(0, 0) = \mathbf{0}$
- (left) Minimum – the positive definite $\mathbf{H} = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.1 \end{bmatrix}$
- (middle) Saddle – the indefinite $\mathbf{H} = \begin{bmatrix} 0.2 & 0 \\ 0 & -0.1 \end{bmatrix}$
- (right) Maximum – the negative definite $\mathbf{H} = \begin{bmatrix} -0.2 & 0 \\ 0 & -0.1 \end{bmatrix}$

A More Complex Case: the Rosenbrock's "Banana" Test Function



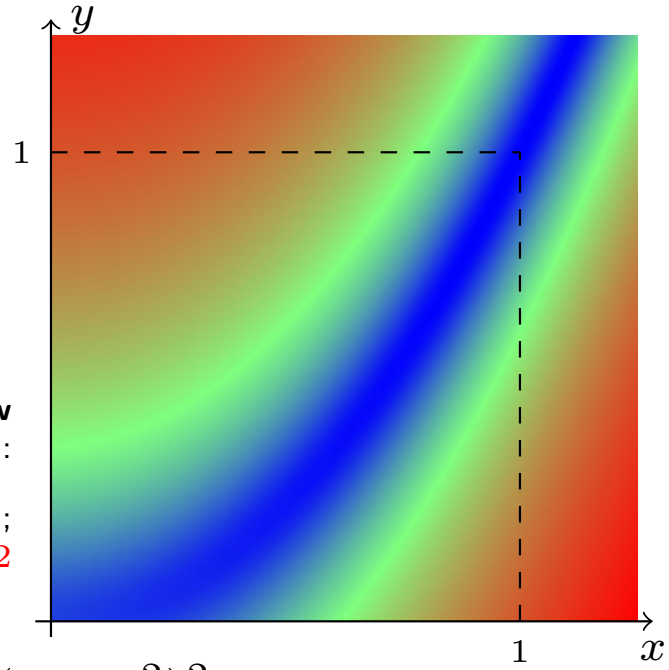
Logarithmic colour scales

Left window
$-2.5 \leq x, y \leq 2.5$:

$f(1,1) = 0$; $f(0,0) = 1$;
$f(-2.5,-2.5) = 7668.5$

Right window
$0 \leq x, y \leq 1.25$:

$f(1,1) = 0$; $f(0,0) = 1$;
$f(1.25,0) = 244.2$



- Bivariate case: $f(x,y) = (1-x)^2 + 100(y-x^2)^2$

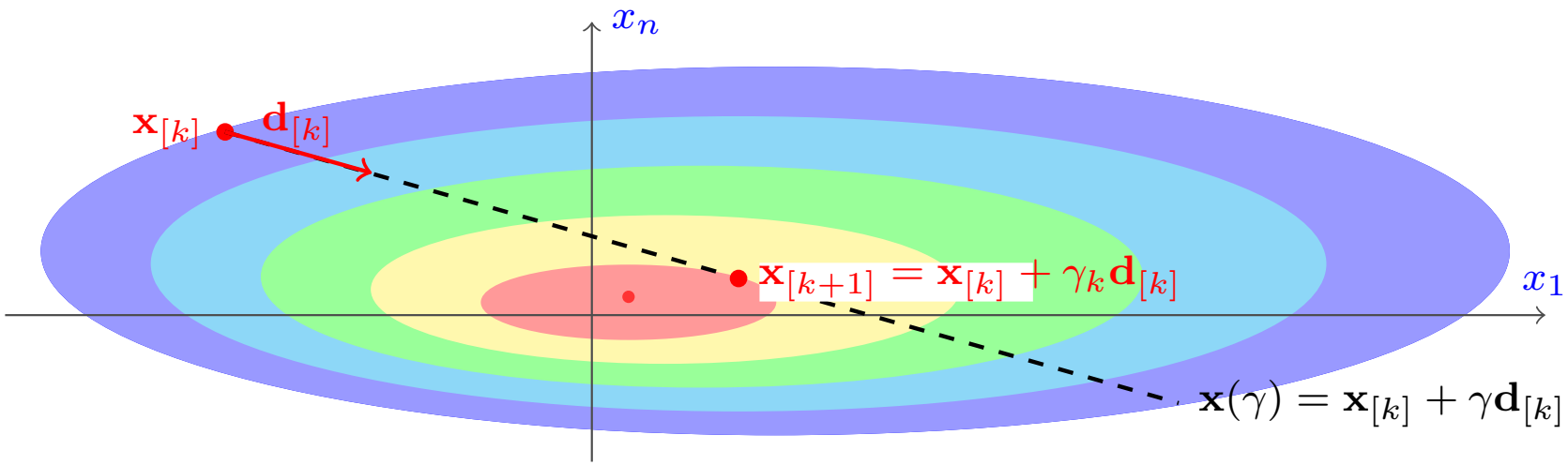
  $\nabla f = \begin{bmatrix} -2(1-x) - 400x(y-x^2) \\ 200(y-x^2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow$ Single minimum $f(1,1) = 0$

- Multivariate extension: $f(x_1, \ldots, x_n) = \sum_{i=1}^{n-1} \left[(1-x_i)^2 + 100 \left(x_{i+1} - x_i^2 \right)^2 \right]$
  - For $n = 3$: Single minimum at $[1,1,1]$
  - For $4 \leq n \leq 7$: Global minimum at all ones $(1,1,\ldots,1)$ and local minimum near $(-1,1,\ldots,1)$

Line Search for a Maximal Point

Find a maximiser of $f(\mathbf{x})$ along a direction $\mathbf{d}_{[k]}$ from a point $\mathbf{x}_{[k]}$, i.e. along the line $\mathbf{x}(\gamma) = \mathbf{x}_{[k]} + \gamma \mathbf{d}_{[k]}$:

$$\begin{cases} \gamma_{[k]} & = & \arg\max\limits_{\gamma \in \mathbb{R}} f(\mathbf{x}(\gamma)) \equiv \arg\max\limits_{\gamma \in \mathbb{R}} f\left(\mathbf{x}_{[k]} + \gamma \mathbf{d}_{[k]}\right) \\ \mathbf{x}_{[k+1]} & = & \mathbf{x}_{[k]} + \gamma_{[k]} \mathbf{d}_{[k]} \end{cases}$$



Line search is used repeatedly in many multivariate search methods

Line Search for a Maximal Point: An Example

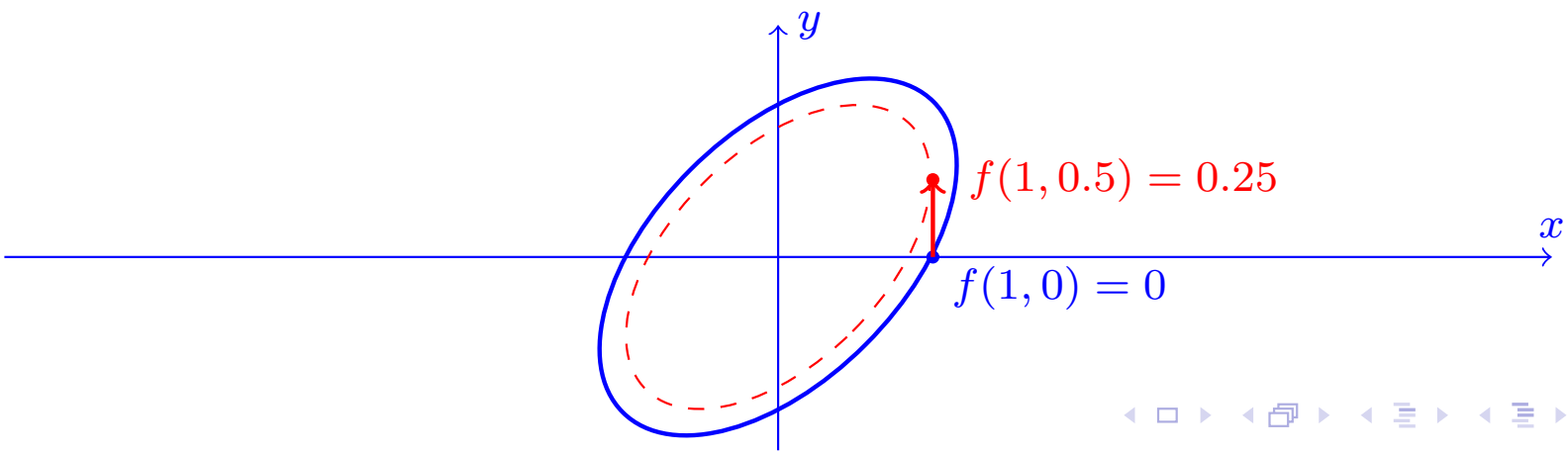
Find a maximiser of the quadratic function

$$f(x, y) = -x^2 + xy - y^2 + 1$$

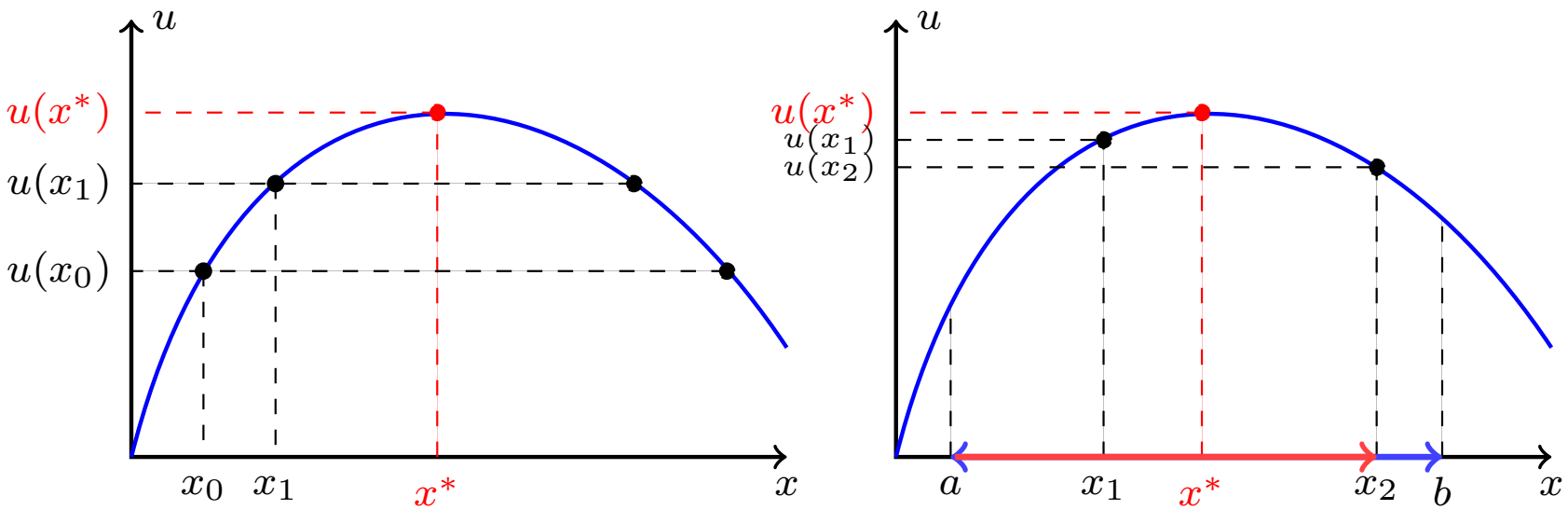
along a direction $[\delta_x = 0, \delta_y = 1]$ from the point $[x^\circ = 1, y^\circ = 0]$

Maximising along the line $[x^\circ + \gamma\delta_x, y^\circ + \gamma\delta_y]$, i.e. the line $[1 + 0 \cdot \gamma, 0 + 1 \cdot \gamma] = [1, \gamma]$ in this example:

$$\begin{aligned} f(1,\gamma) &= -1 + \gamma - \gamma^2 + 1 = \gamma - \gamma^2 \\ \gamma^* &= \arg\max_{\gamma \in \mathbb{R}} \{\gamma - \gamma^2\} = 0.5 \Leftarrow \frac{d(\gamma - \gamma^2)}{d\gamma} = 1 - 2\gamma = 0 \\ x^* = 1 \quad ; \quad y^* &= 0.5 \Rightarrow f(1, 0.5) = 0.25 > f(1, 0) = 0 \end{aligned}$$



Maximising an Univariate Unimodal Function $u(x)$



Properties of the maximiser $x^* = \arg\max_x u(x)$

- If $x_0 < x_1 < x^*$ or $x_0 > x_1 > x^*$, then $u(x_0) < u(x_1) < u(x^*)$
- If $a \le x^* \le b$ and $a \le x_1 < x_2 < b$ or $a < x_1 < x_2 \le b$, then

$$\left.\begin{array}{lcl} u(x_1) < u(x_2) & \Rightarrow & x_1 < x^* \le b \\ u(x_1) = u(x_2) & \Rightarrow & x_1 < x^* < x_2 \\ u(x_1) > u(x_2) & \Rightarrow & a \le x^* < x_2 \end{array}\right\}$$ Therefore, the search interval $a \le x^* \le b$ is reduced

Golden Section Search

Initialisation: an interval $[a_0, b_0]$; $a_0 \leq x^* \leq b_0$, containing the maximiser x^* of a unimodal function $u(x)$
Iteration (step) $i = 0, 1, 2, \ldots$, of reducing the search interval:

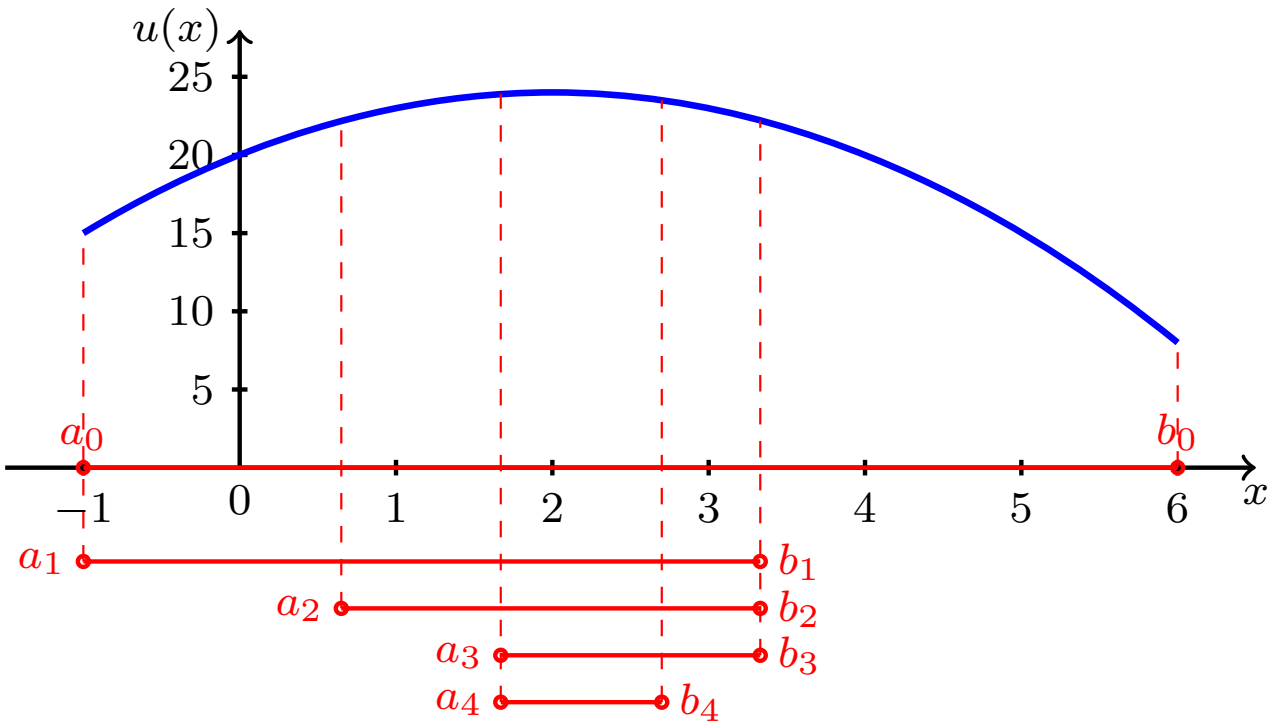
- Evaluate $u(x_i)$ and $u(\bar{x}_i)$ at symmetric internal points of the interval $[a_i, b_i]$ (below $\tau = \frac{1+\sqrt{5}}{2}$ is the Greek golden section ratio):

$$\begin{cases} x_i = a_i + (b_i - a_i)(2 - \tau) \approx a_i + 0.382(b_i - a_i) \\ \bar{x}_i = a_i + (b_i - a_i)(\tau - 1) \approx a_i + 0.618(b_i - a_i) \end{cases}$$

  - If $u(x_i) > u(\bar{x}_i)$, then $a_{i+1} \leftarrow a_i$ and $b_{i+1} \leftarrow \bar{x}_i$
  - If $u(x_i) < u(\bar{x}_i)$, then $a_{i+1} \leftarrow x_i$ and $b_{i+1} \leftarrow b_i$
  - If $u(x_i) = u(\bar{x}_i)$, then $a_0 \leftarrow x_i$; $b_0 \leftarrow \bar{x}_i$, and initiate the search again from the new interval $[a_0, b_0]$ and $i = 0$

Stopping rule: Proceed until the interval $[a_0, b_0]$ is sufficiently small, or the next point is within the resolution distance of the last point

Golden Section Search: A Simple Example



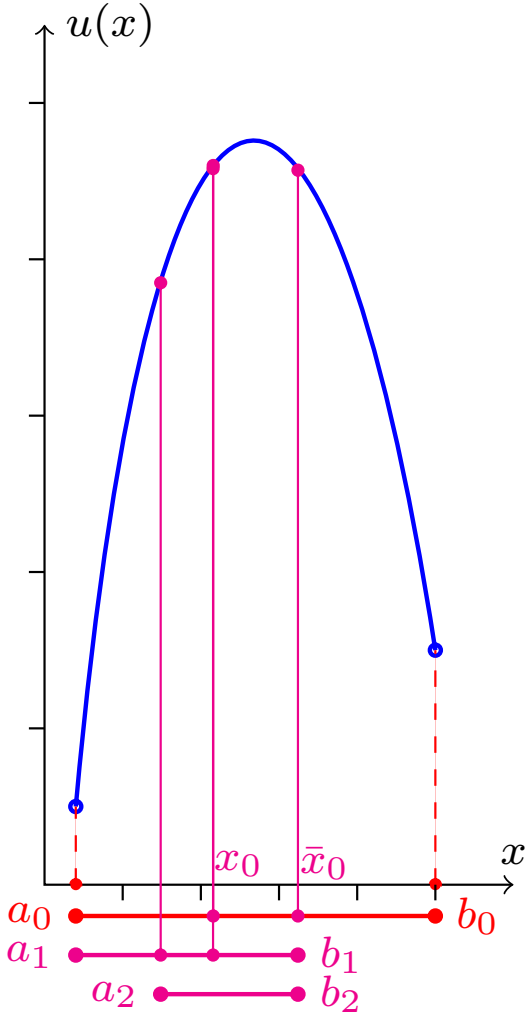
$$u(x) = 20 + 4x - x^2$$

$$x^* = 2;\ u(x^*) = 24$$

i	0	1	2	3	4	5	6	7	8	9	10
a_i	-1.0	-1.00	0.65	1.67	1.67	1.67	1.92	1.92	1.92	1.97	1.97
b_i	6.0	3.33	3.33	3.33	2.70	2.31	2.31	2.16	2.06	2.06	2.03
$u(a_i)$	15.	15.0	22.2	23.9	23.9	23.9	23.9_9	23.9_9	23.9_9	24.0	24.0
$u(b_i)$	8.	22.2	22.2	22.2	23.5	23.9	23.9_1	23.9_8	23.9_9	23.9_9	24.0

Golden Section vs. Fibonacci Search (optional)

Golden section search: an example



- Golden section search is less efficient than the Fibonacci search: for $i = 1, 2, \ldots, n-1$,

$$x_i = a_i + (b_i - a_i)F_{n-i}/F_{n+2-i}$$
$$\bar{x}_i = a_i + (b_i - a_i)F_{n+1-i}/F_{n+2-i}$$

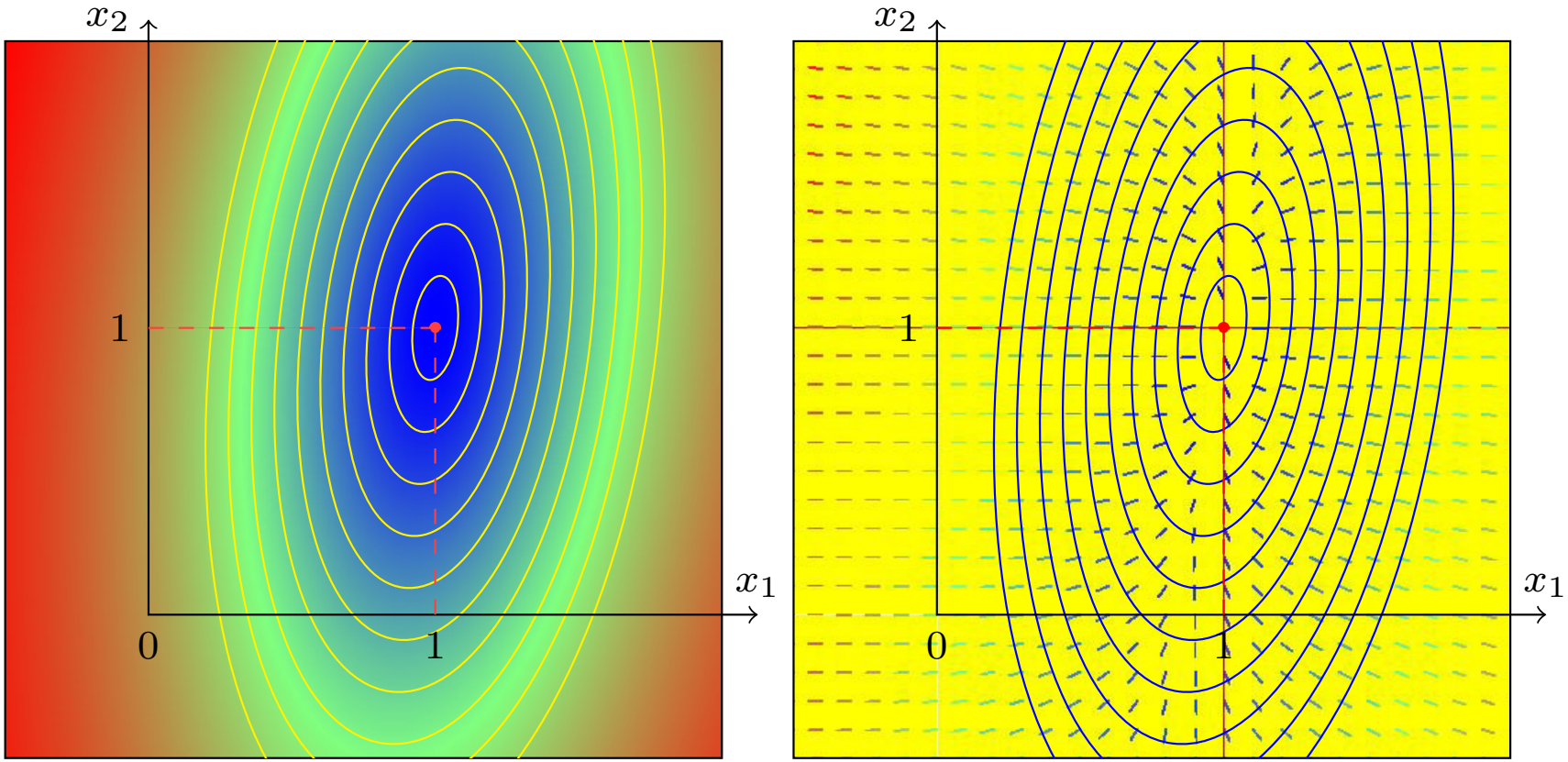
  where F_k is the Fibonacci number: $F_0 = 0$; $F_1 = 1$; $F_k = F_{k-1} + F_{k-2}$, $k = 2, 3, \ldots$

  - Fibonacci search minimises the maximal interval of uncertainty about the maximiser x^* (in that sense it is optimal)
  - But the number of points n to be evaluated in the Fibonacci search has to be prescribed

- Search for the root x^* of the first derivative, $\frac{du}{dx}(x^*) = 0$, be it available, is even more efficient

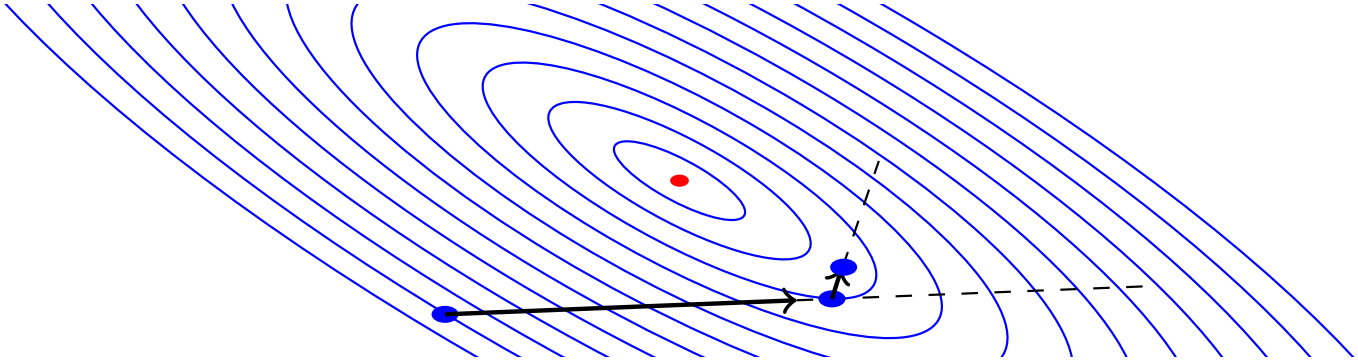
Gradient of a Function of Several Variables

Gradient vector $\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right]^{\mathsf{T}}$ is directed to the greatest slope of the function f at any point.



$f(x_1, x_2) = 10x_1^2 - 2x_1x_2 + 2x_2^2 - 18x_1 - 2x_2$; $\frac{\partial f}{\partial x_1} = 20x_1 - 2x_2 - 18$; $\frac{\partial f}{\partial x_2} = -2x_1 + 4x_2 - 2$

Gradient Search



Gradient methods for seeking a maximum (minimum) for f:

1. Evaluate the gradient at an initial point
2. Move along the gradient direction for a computable distance
3. Repeat this process until the maximum (minimum) is found

If exact partial derivatives are unknown, the gradients may be numerically approximated: for a small $\varepsilon > 0$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_1, \ldots, x_{i-1}, x_i + \varepsilon, x_{i+1}, \ldots, x_n) - f(x_1, \ldots, x_{i-1}, x_i - \varepsilon, x_{i+1}, \ldots, x_n)}{2\varepsilon}$$

- Approximation errors can make the methods less attractive

Gradient Maximisation: The Steepest Ascent

Initialisation: Select an initial point $\mathbf{x}_0$
Iterative steps $i = 0, 1, \ldots$:

1. Compute the gradient $\nabla f(\mathbf{x})$ at the point $\mathbf{x}_i$
2. Draw a line $\mathbf{x}_i + t\nabla f(\mathbf{x}_i)$ through $\mathbf{x}_i$ in the gradient direction
3. Select the point $\mathbf{x}_{i+1}$ on this line yielding the largest value for f of all points on the line:

$$f(\mathbf{x}_{i+1}) = \max_{t \in (0,\infty)} \{f(\mathbf{x}) : \mathbf{x} = \mathbf{x}_i + t\nabla f(\mathbf{x}_i)\}$$

Stopping rule: Terminate if the gradient is small $(|\nabla f(\mathbf{x}_i)| \approx \mathbf{0})$ or successive points are close to each other $(|\mathbf{x}_{i+1} - \mathbf{x}_i| \approx \mathbf{0})$

Gradient Minimisation: The Steepest Descent

Initialisation: Select an initial point $\mathbf{x}_0$
Iterative steps $i = 0, 1, \ldots$:

1. Compute the gradient $\nabla f(\mathbf{x})$ at the point $\mathbf{x}_i$
2. Draw a line $\mathbf{x}_i - t\nabla f(\mathbf{x}_i)$ in the opposite gradient direction
3. Select the point $\mathbf{x}_{i+1}$ on this line yielding the smallest value for f of all points on the line:

$$f(\mathbf{x}_{i+1}) = \min_{t \in (0,\infty)} \{f(\mathbf{x}) : \mathbf{x} = \mathbf{x}_i - t\nabla f(\mathbf{x}_i)\}$$

Stopping rule: Terminate if the gradient is small $(|\nabla f(\mathbf{x}_i)| \approx \mathbf{0})$ or successive points are close to each other $(|\mathbf{x}_{i+1} - \mathbf{x}_i| \approx \mathbf{0})$

Finding the Best Point Along the Gradient Direction

Search for the extremum point along the line $f(\mathbf{x}) \pm t \cdot \nabla f(\mathbf{x})$

If the derivatives ∇f are computable and
the function f is well-behaving **then**:

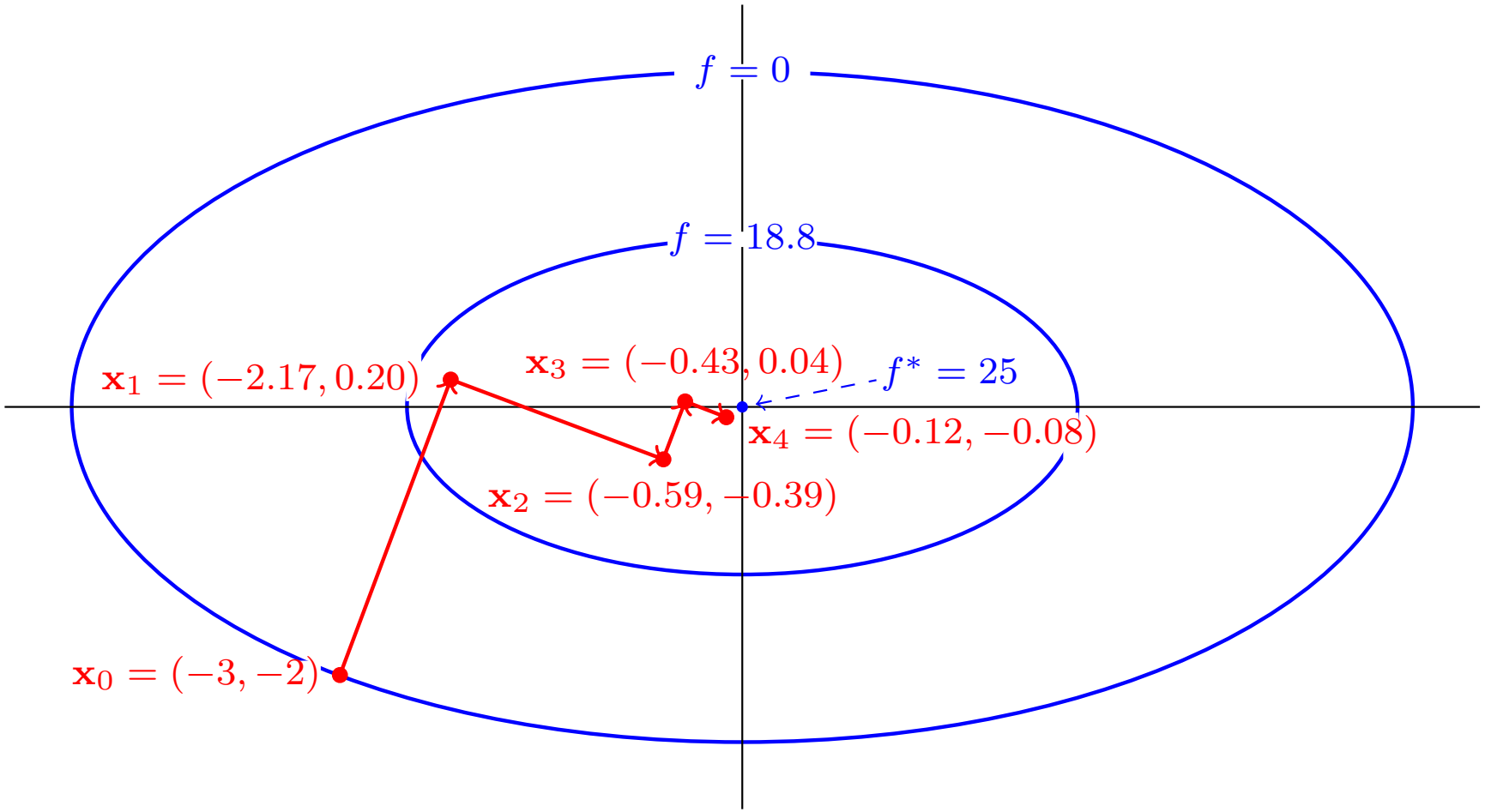
1. Substitute $\mathbf{x}_0 \pm t \cdot \nabla f(\mathbf{x}_0)$ into the equation for f,
2. Differentiate with respect to t, and
3. Set the derivative equal to zero to find t

else if the function $u(t) \equiv f\left(\mathbf{x}_0 \pm t \nabla f(\mathbf{x}_0)\right)$ is unimodal
then use any one-dimensional line search, e.g.

- the golden section search or
- the Fibonacci section search

Steepest Ascent: An Example

$f(\mathbf{x}) \equiv f(x, y) = 25 - x^2 - 4y^2$; $\nabla f(\mathbf{x}) = (-2x, -8y)$; $\mathbf{x}_0 = (-3, -2)$:
$f(\mathbf{x}_0) = 0$; $f(\mathbf{x}_1) = 20.1$; $f(\mathbf{x}_2) = 24.0$; $f(\mathbf{x}_3) = 24.8$; $f(\mathbf{x}_4) = 24.9$; ...



Steepest Descent: $f(x,y) = 10x^2 - 2xy + 2y^2 - 18x - 2y$

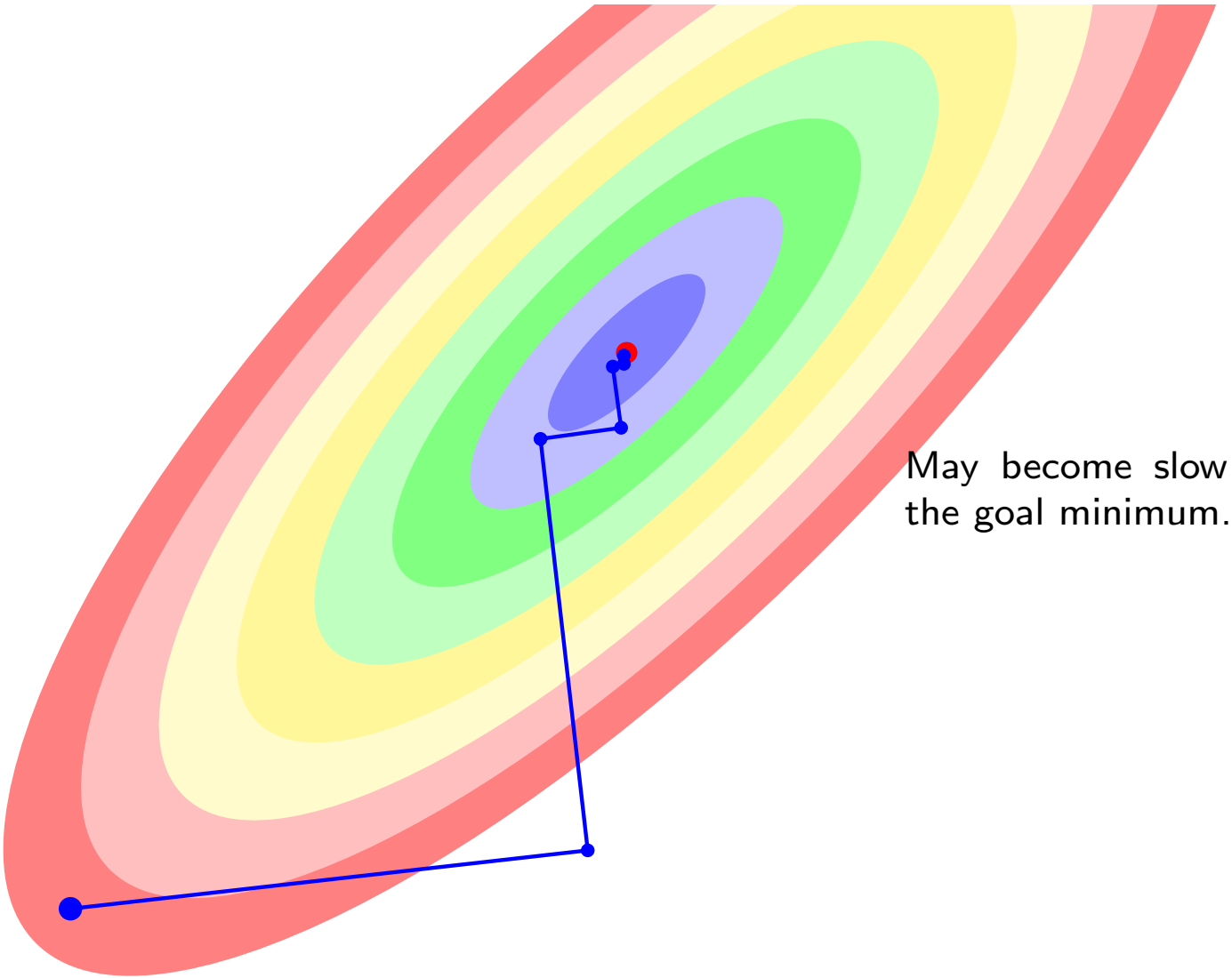
$$\begin{cases} \left.\frac{\partial f}{\partial x}\right|_{x_i,y_i} \equiv u_i &= 20x - 2y - 18 \\ \left.\frac{\partial f}{\partial y}\right|_{x_i,y_i} \equiv v_i &= -2x + 4y - 2 \end{cases} \Rightarrow \begin{cases} x_{i+1} &= x_i - t_i u_i \\ y_{i+1} &= y_i - t_i v_i \end{cases}$$

$$f(x_i + tu_i, y_i + tv_i) = f(x_i, y_i) - t\left(u_i^2 + v_i^2\right) + t^2\left(au_i^2 + bu_iv_i + cv_i^2\right)$$

$$\Rightarrow t_i = \arg\min_t \{f(x_i + tu_i, y_i + tv_i)\} = \frac{1}{2}\frac{u_i^2+v_i^2}{au_i^2+bu_iv_i+cv_i^2}$$

i	x_i	y_i	$f(x_i,y_i)$	u_i	v_i	x_{i+1}	y_{i+1}
0	−1.0000	−1.0000	30.0000	−36.0000	−4.0000	0.8589	−0.7935
1	0.8589	−0.7935	−3.8741	0.7657	−6.8917	0.6937	0.6937
2	0.6937	0.6937	−9.0618	−5.5134	−0.6126	0.9784	0.7253
3	0.9784	0.7253	−9.8563	0.1173	−1.0555	0.9531	0.9531
4	0.9531	0.9531	−9.9780	−0.8444	−0.0938	0.9967	0.9579
5	0.9967	0.9579	−9.9966	0.0180	−0.1616	0.9928	0.9928
6	0.9928	0.9928	−9.9995	−0.1293	−0.0144	0.9995	0.9936
7	0.9995	0.9936	−9.9999	0.0028	−0.0248	0.9989	0.9989
8	0.9989	0.9989	−10.0000	−0.0198	−0.0022	0.9999	0.9990
9	0.9999	0.9990	−10.0000	0.0004	−0.0038	0.9998	0.9998

Steepest Descent: $f(x, y) = 10x^2 - 2xy + 2y^2 - 18x - 2y$

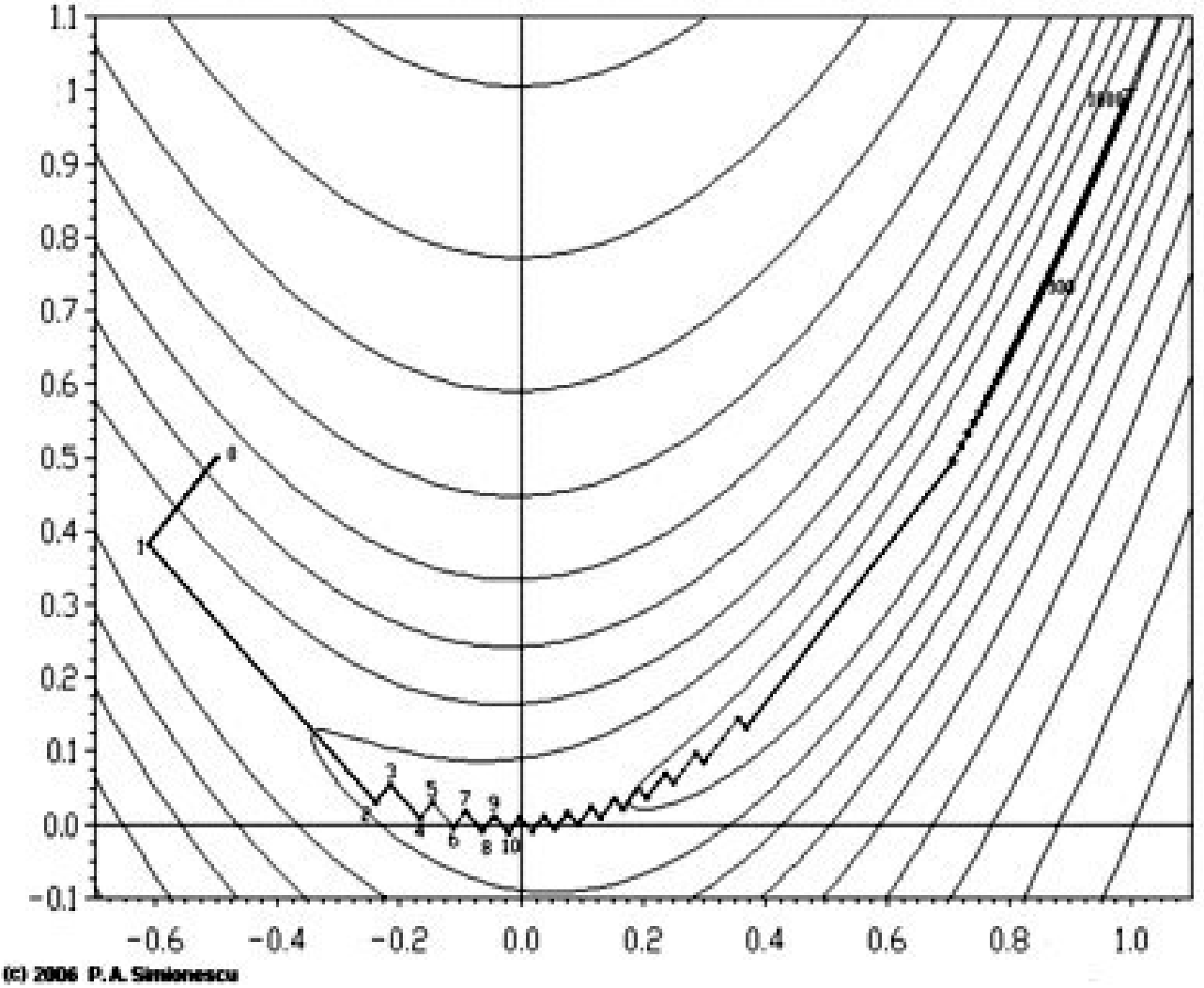


May become slow near the goal minimum...

Steepest Descent: Rosenbrock's "Banana"

$$f(x, y) = (1 - x)^2 + 100\left(y - x^2\right)^2$$

http://en.wikipedia.org/wiki/Gradient_descent



Unconstrained Optimisation: An Informal Sketch

Minimum or maximum of a function $f(\mathbf{x})$ of many variables

- The goal: $\mathbf{x}^* = \arg\{\min^{\max}\} f(\mathbf{x})$; $f^* \equiv f(\mathbf{x}^*) = \{\min^{\max}\} f(\mathbf{x})$
- Mostly: a multi-modal $f(\mathbf{x})$
  - Modes correspond to the gradient's roots $\nabla f(\mathbf{x}^\circ) = \mathbf{0}$
  - Most optimistic case: an analytical solution of this system
  - The optimum may not be unique: $f^* = f(\mathbf{x}_1^*) = \ldots = f(\mathbf{x}_k^*)$

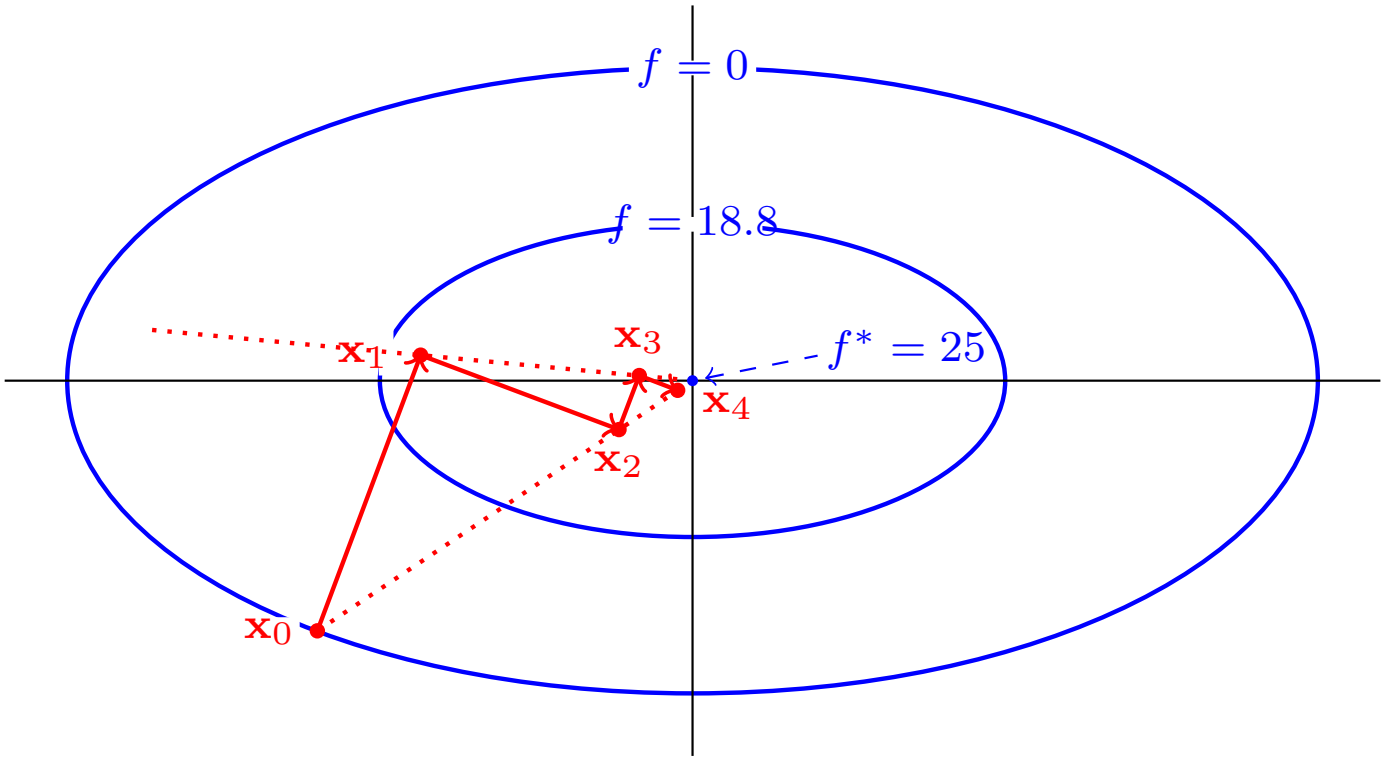
Global optimisation is typically a hard problem

- Feasible solutions - only for specific functions

Local optimisation: find the optimiser $\mathbf{x}^\circ$ close to a known $\mathbf{x}_0$

- Most optimistic case: $f(\mathbf{x})$; $\nabla f(\mathbf{x})$ – an analytical solution
- Typically – an iterative linear search starting from $\mathbf{x}_0$
  - Analytical or numerical selection of direction $\mathbf{s}_i$ at each step i
  - Analytical or numerical step, e.g. $t^\circ = \arg\{\min^{\max}\}_t f(\mathbf{x}_i + t\mathbf{s}_i)$
  - Accelerated linear search with interdependent $\mathbf{s}_i$ and $\mathbf{s}_j$; $j < i$

Accelerated Gradient Search



- Once $i > 2$, for i odd the point $\mathbf{x}_i$ is found by gradient search from $\mathbf{x}_{i-1}$, and $\mathbf{x}_{i+1}$ is found by an accelerated step by maximising over the line through $\mathbf{x}_i$ and $\mathbf{x}_{i-2}$
- Global maximum of a negative definite quadratic function of n variables is provably found after $2n - 1$ steps of this procedure

Direct Search Methods

- If both the gradient and Hessian of $f(\mathbf{x})$ are too complicated to compute but f can be evaluated at any point $\mathbf{x} \in \mathbb{R}_n$

Pattern search of K. Hooke and T. A. Jeeves

- For $i = 1, \ldots, n$ sequentially:
  - If $f(x_1, \ldots, x_i + \varepsilon_i, \ldots, x_n) > f(x_1, \ldots, x_i, \ldots, x_n)$, replace $x_i \leftarrow x_i + \varepsilon$
  - Else if $f(x_1, \ldots, x_i - \varepsilon_i, \ldots, x_n) > f(x_1, \ldots, x_i, \ldots, x_n)$, replace $x_i \leftarrow x_i - \varepsilon$
- Repeat this cycle of perturbations until no perturbations about $\mathbf{x}_j$ bring about an improvement
- Halve the pre-defined perturbation sizes ε_i and repeat the process while the next point brings an improvement over $\mathbf{x}_j$

Sectioning and Accelerated Rosenbrock's Search

One-at-a-time search, or sectioning, from an initial point $\mathbf{x}_0$

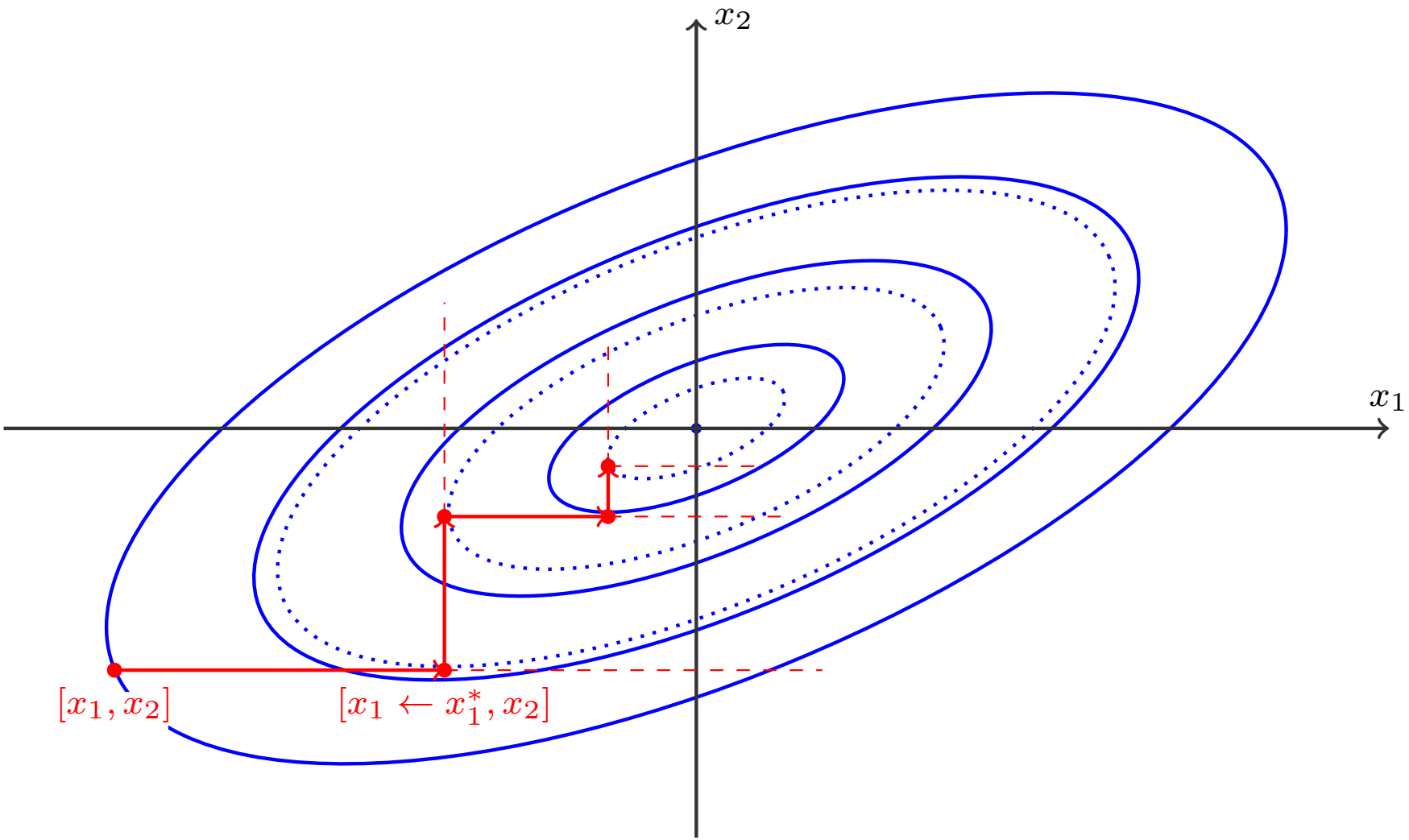
- For $i = 1, \ldots, n$ sequentially, search for the maximum in the direction of the variable x_i by one of the one-dimensional search methods and replace x_i by the maximiser x_i^*: $x_i \leftarrow x_i^*$
- Repeat this cycle of one-dimensional searches until the steps $x_i - x_i^*$; $i = 1, \ldots, n$ become less than a given threshold

Convergence rate is usually too slow and the search may halt far from the optimum

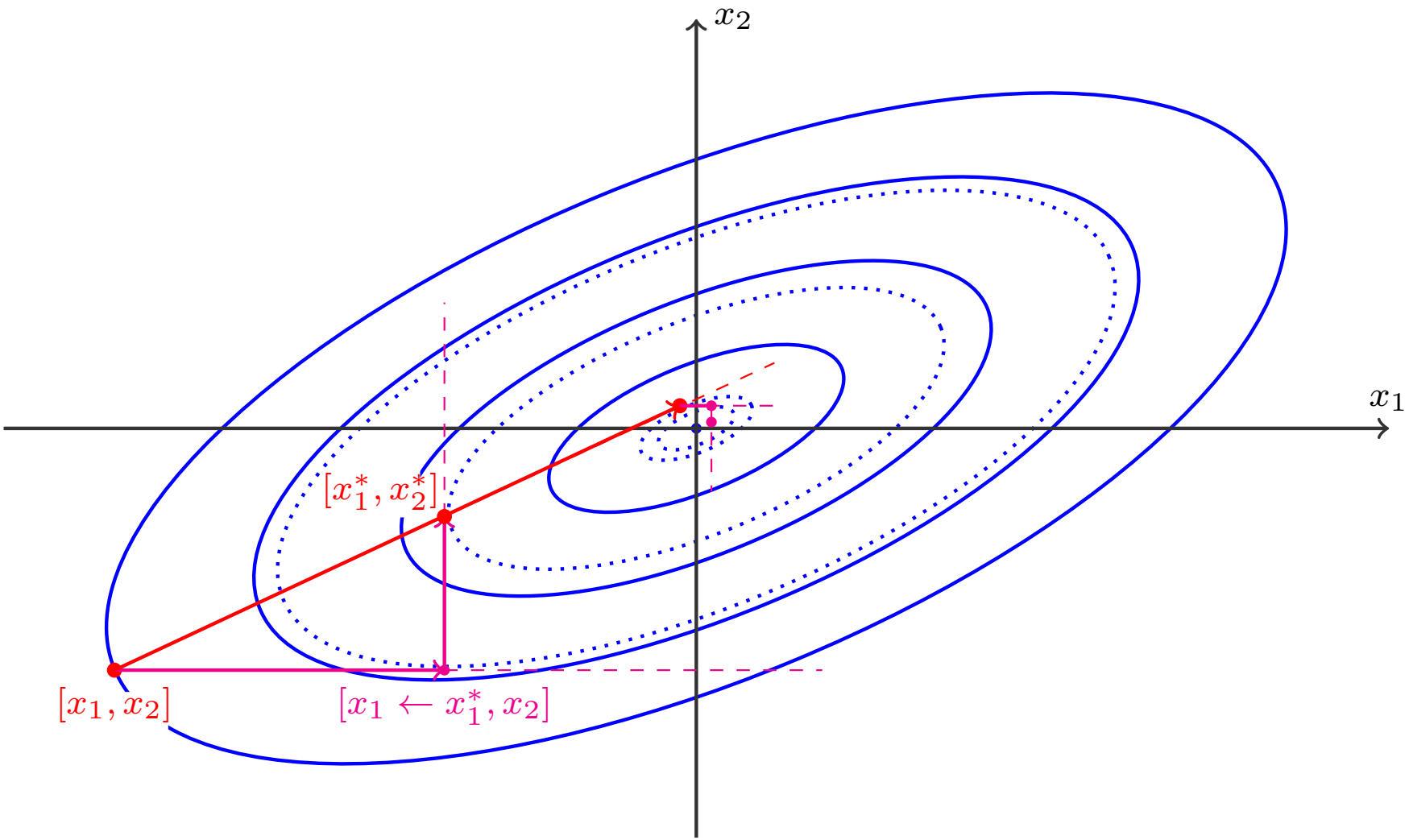
Accelerated search of H. H. Rosenbrock

- Use one-at-a-time search from $\mathbf{x}_0$ to find the next point $\mathbf{x}_1^*$ and the direction $\boldsymbol{\delta}$ with components $\delta_i = x_{1:i}^* - x_{0:i}$
- Search for the maximum in the direction $\boldsymbol{\delta}$ and replace $\mathbf{x}_0$ by the maximiser $\mathbf{x}_1$ found
- Repeat this cycle until $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$ are closer than a threshold

One-at-a-time Search: An Example



Rosenbrock's Search: An Example



Coordinate descent

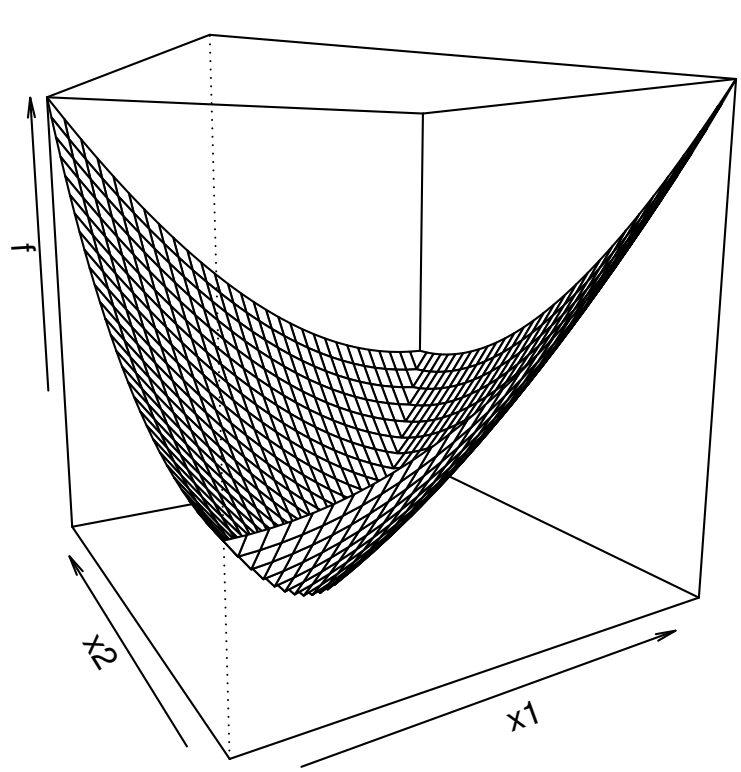
We've seen some pretty sophisticated methods

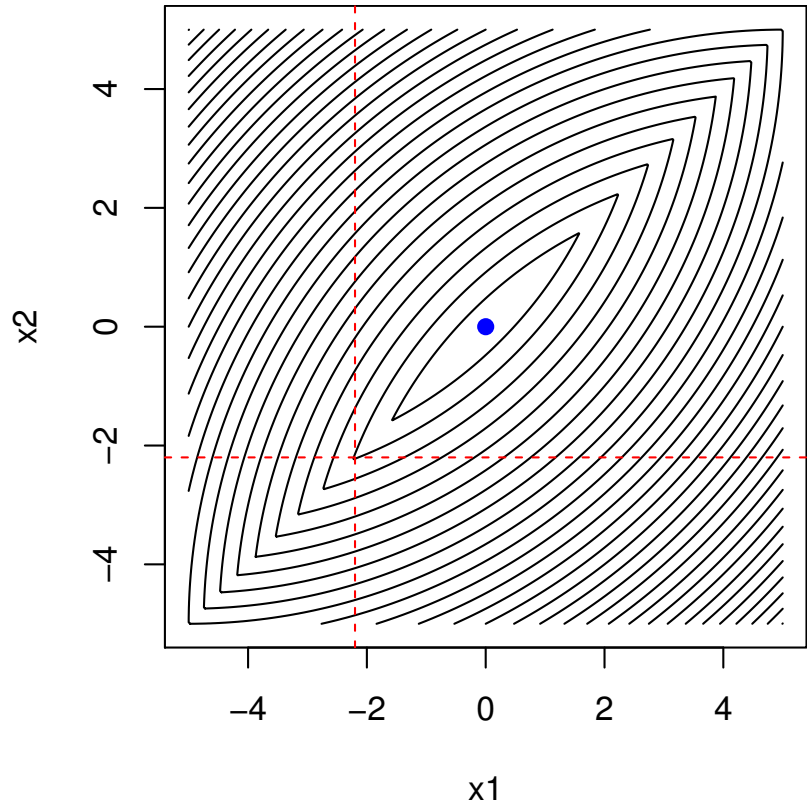
Our focus today is a very simple technique that can be surprisingly efficient and scalable: coordinate descent, i.e., coordinate-wise minimization

Q: Given convex, differentiable $f : \mathbb{R}^n \to \mathbb{R}$, if we are at a point x such that $f(x)$ is minimized along each coordinate axis, *have we found a global minimizer?*

I.e., does $f(x + \delta \cdot e_i) \geq f(x)$ for all $\delta, i \implies f(x) = \min_z f(z)$?

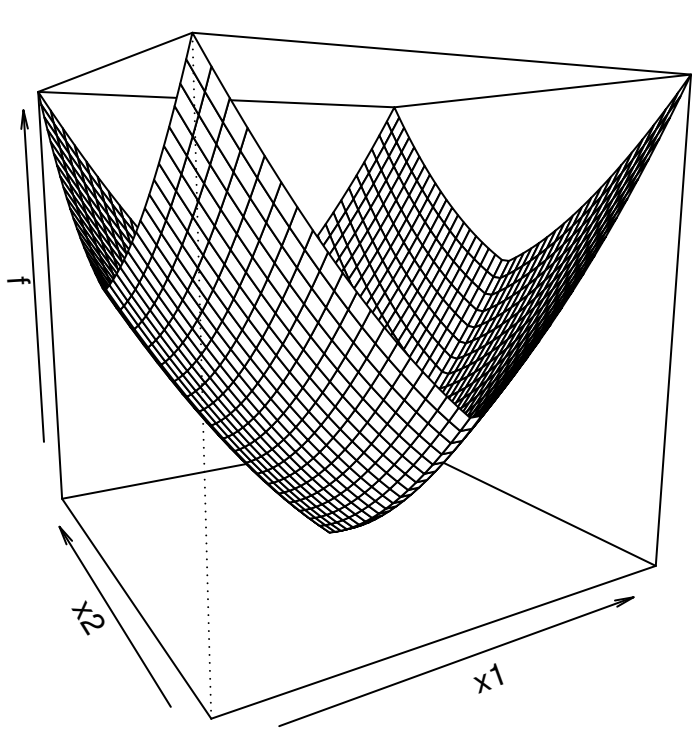
(Here $e_i = (0, \ldots, 1, \ldots 0) \in \mathbb{R}^n$, the ith standard basis vector)

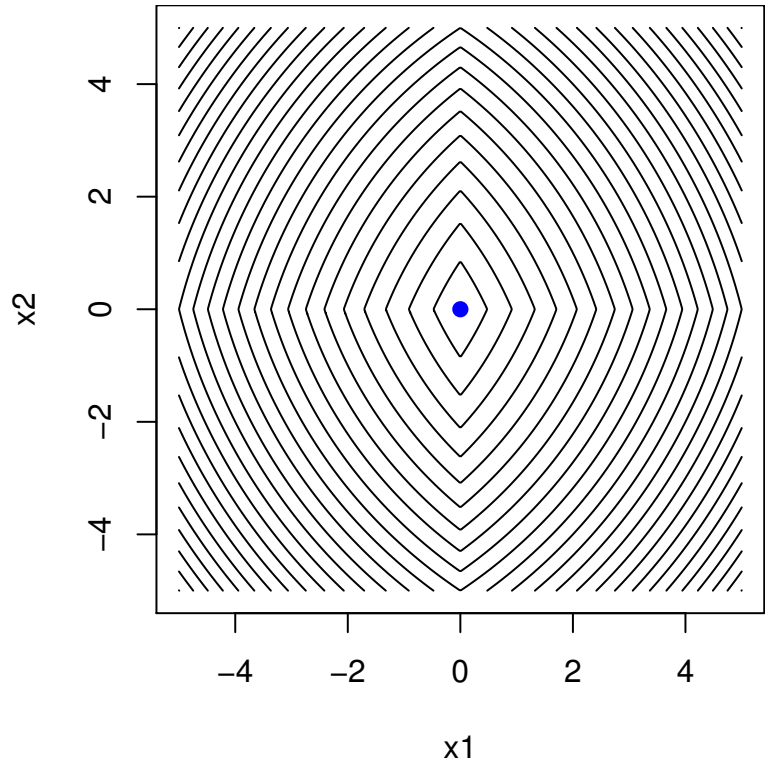




A: No! Look at the above counterexample

Q: Same question again, but now $f(x) = g(x) + \sum_{i=1}^{n} h_i(x_i)$, with g convex, differentiable and each h_i convex ... ? (Non-smooth part here called separable)





A: Yes! Proof: for any y,

$$f(y) - f(x) \geq \nabla g(x)^T (y - x) + \sum_{i=1}^{n} [h_i(y_i) - h_i(x_i)]$$

$$= \sum_{i=1}^{n} \underbrace{[\nabla_i g(x)(y_i - x_i) + h_i(y_i) - h_i(x_i)]}_{\geq 0} \geq 0$$

Coordinate descent

This suggests that for $f(x) = g(x) + \sum_{i=1}^n h_i(x_i)$ (with g convex, differentiable and each h_i convex) we can use coordinate descent to find a minimizer: start with some initial guess $x^{(0)}$, and repeat

$$
\begin{aligned}
x_1^{(k)} &\in \underset{x_1}{\operatorname{argmin}}\ f\big(x_1, x_2^{(k-1)}, x_3^{(k-1)}, \ldots x_n^{(k-1)}\big) \\
x_2^{(k)} &\in \underset{x_2}{\operatorname{argmin}}\ f\big(x_1^{(k)}, x_2, x_3^{(k-1)}, \ldots x_n^{(k-1)}\big) \\
x_3^{(k)} &\in \underset{x_2}{\operatorname{argmin}}\ f\big(x_1^{(k)}, x_2^{(k)}, x_3, \ldots x_n^{(k-1)}\big) \\
&\ldots \\
x_n^{(k)} &\in \underset{x_2}{\operatorname{argmin}}\ f\big(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \ldots x_n\big)
\end{aligned}
$$

for $k = 1, 2, 3, \ldots$ (note: after we solve for $x_i^{(k)}$, we use its new value from then on!)

Seminal work of Tseng (2001) proves that for such f (provided f is continuous on compact set $\{x : f(x) \leq f(x^{(0)})\}$ and f attains its minimum), any limit point of $x^{(k)}$, $k = 1, 2, 3, \ldots$ is a minimizer of f[1]

Notes:

- Order of cycle through coordinates is arbitrary, can use any permutation of $\{1, 2, \ldots n\}$
- Can everywhere replace individual coordinates with blocks of coordinates
- "One-at-a-time" update scheme is critical, and "all-at-once" scheme does not necessarily converge

[1] Using real analysis, we know that $x^{(k)}$ has subsequence converging to $x^\star$ (Bolzano-Weierstrass), and $f(x^{(k)})$ converges to $f^\star$ (monotone convergence)

Linear regression

Consider linear regression

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{2}\|y - X\beta\|_2^2$$

where $y \in \mathbb{R}^n$, and $X \in \mathbb{R}^{n \times p}$ with columns $X_1, \ldots X_p$

Minimizing over β_i, with all β_j, $j \neq i$ fixed:

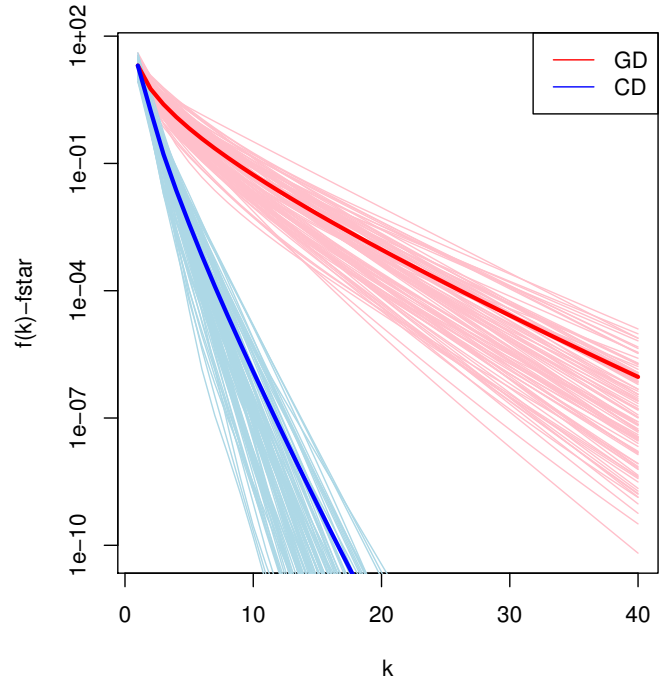
$$0 = \nabla_i f(\beta) = X_i^T(X\beta - y) = X_i^T(X_i\beta_i + X_{-i}\beta_{-i} - y)$$

i.e., we take

$$\beta_i = \frac{X_i^T(y - X_{-i}\beta_{-i})}{X_i^T X_i}$$

Coordinate descent repeats this update for $i = 1, 2, \ldots, p, 1, 2, \ldots$

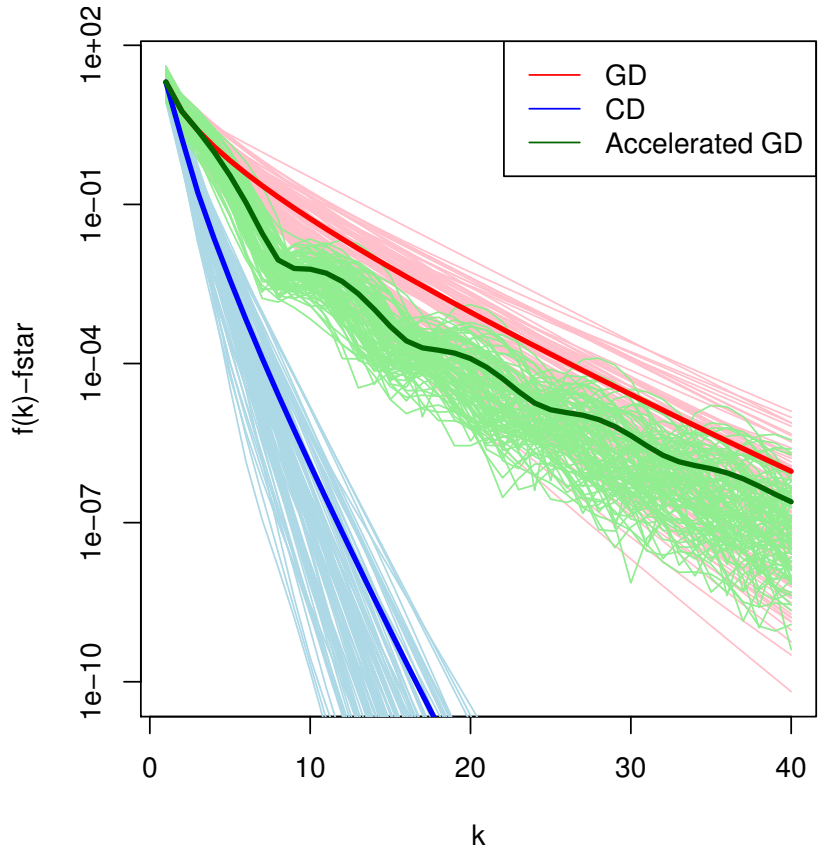
Coordinate descent vs gradient descent for linear regression: 100 instances ($n = 100$, $p = 20$)



Is it fair to compare 1 cycle of coordinate descent to 1 iteration of gradient descent? Yes, if we're clever:

$$\beta_i \leftarrow \frac{X_i^T(y - X_{-i}\beta_{-i})}{X_i^T X_i} = \frac{X_i^T r}{\|X_i\|_2^2} + \beta_i$$

where $r = y - X\beta$. Therefore each coordinate update takes $O(n)$ operations — $O(n)$ to update r, and $O(n)$ to compute $X_i^T r$ — and one cycle requires $O(np)$ operations, just like gradient descent



Same example, but now with accelerated gradient descent for comparison

Is this contradicting the optimality of accelerated gradient descent? I.e., is coordinate descent a first-order method?

No. It uses much more than first-order information

Lasso regression

Now consider the lasso problem

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1$$

Note that the non-smooth part is separable: $\|\beta\|_1 = \sum_{i=1}^p |\beta_i|$

Minimizing over β_i, with β_j, $j \neq i$ fixed:

$$0 = X_i^T X_i \beta_i + X_i^T (X_{-i}\beta_{-i} - y) + \lambda s_i$$

where $s_i \in \partial|\beta_i|$. Solution is simply given by soft-thresholding

$$\beta_i = S_{\lambda/\|X_i\|_2^2}\left(\frac{X_i^T(y - X_{-i}\beta_{-i})}{X_i^T X_i}\right)$$

Repeat this for $i = 1, 2, \ldots p, 1, 2, \ldots$

Box-constrained regression

Consider box-constrainted linear regression

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{2}\|y - X\beta\|_2^2 \quad \text{subject to} \quad \|\beta\|_\infty \leq s$$

Note this fits our framework, as $1\{\|\beta\|_\infty \leq s\} = \sum_{i=1}^n 1\{|\beta_i| \leq s\}$

Minimizing over β_i with all β_j, $j \neq i$ fixed: same basic steps give

$$\beta_i = T_s\left(\frac{X_i^T(y - X_{-i}\beta_{-i})}{X_i^T X_i}\right)$$

where T_s is the truncating operator:

$$T_s(u) = \begin{cases} s & \text{if } u > s \\ u & \text{if } -s \leq u \leq s \\ -s & \text{if } u < -s \end{cases}$$

Introduction

- Gradient descent is a way to minimize an objective function $J(\theta)$
  - $\theta \in \mathbb{R}^d$: model parameters
  - η: learning rate
  - $\nabla_\theta J(\theta)$: gradient of the objective function with regard to the parameters
- Updates parameters **in opposite direction** of gradient.
- Update equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$

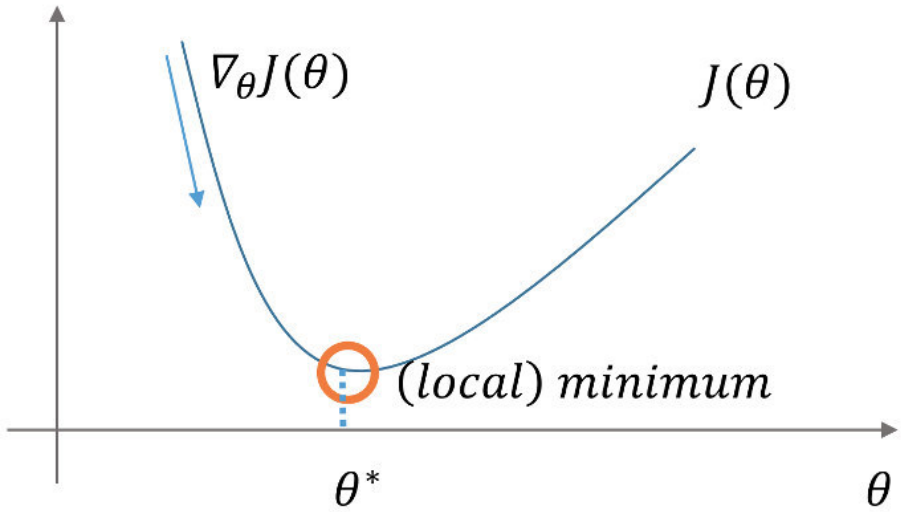


Figure: Optimization with gradient descent

Gradient descent variants

1. Batch gradient descent
2. Stochastic gradient descent
3. Mini-batch gradient descent

Difference: Amount of data used per update

Batch gradient descent

- Computes gradient with the **entire** dataset.
- Update equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$

```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(
    loss_function, data, params)
  params = params - learning_rate * params_grad
```

Listing 1: Code for batch gradient descent update

- Pros:
  - Guaranteed to converge to **global** minimum for **convex** error surfaces and to a **local** minimum for **non-convex** surfaces.
- Cons:
  - **Very slow**.
  - Intractable for datasets that **do not fit in memory**.
  - **No online learning**.

Stochastic gradient descent

- Computes update for **each** example $x^{(i)} y^{(i)}$.
- Update equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(
      loss_function, example, params)
    params = params - learning_rate * params_grad
```

Listing 2: Code for stochastic gradient descent update

- Pros
  - **Much faster** than batch gradient descent.
  - Allows **online learning**.
- Cons
  - **High variance** updates.

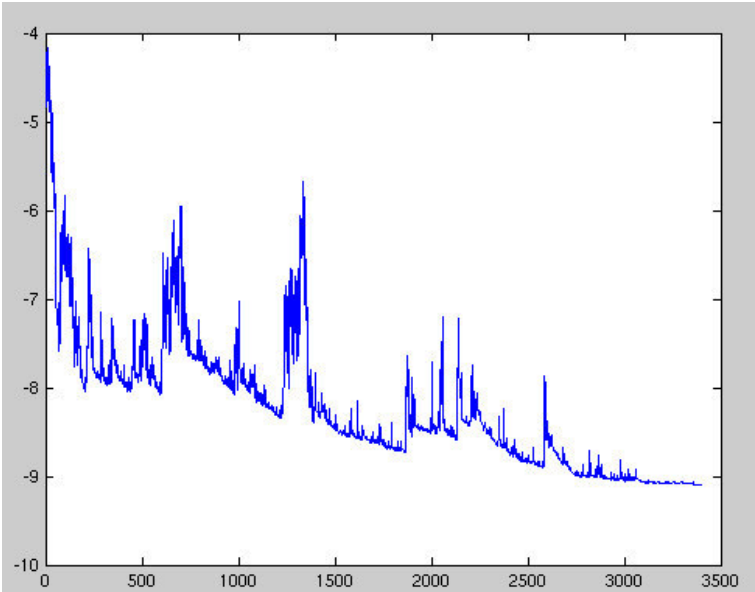


Figure: SGD fluctuation (Source: Wikipedia)

Batch gradient descent vs. SGD fluctuation

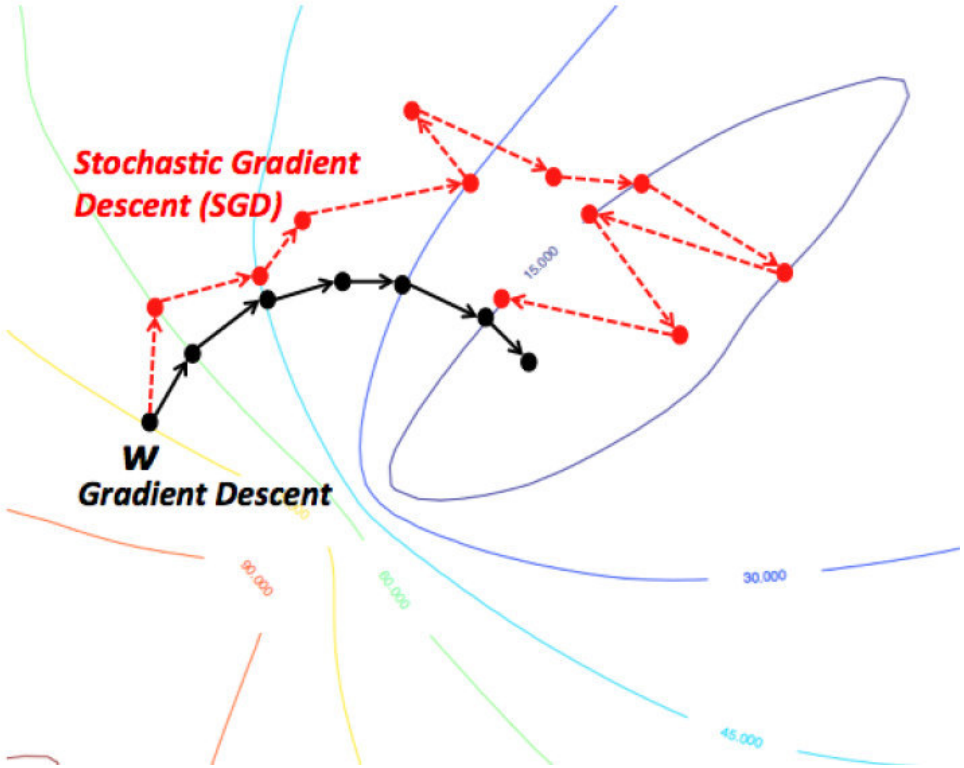


Figure: Batch gradient descent vs. SGD fluctuation (Source: wikidocs.net)

- SGD shows same convergence behaviour as batch gradient descent if learning rate is **slowly decreased (annealed)** over time.

Mini-batch gradient descent

- Performs update for every **mini-batch** of n examples.
- Update equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(
      loss_function, batch, params)
    params = params - learning_rate * params_grad
```

Listing 3: Code for mini-batch gradient descent update

- Pros
  - **Reduces variance** of updates.
  - Can exploit **matrix multiplication** primitives.
- Cons
  - **Mini-batch size** is a hyperparameter. Common sizes are 50-256.
- Typically the algorithm of choice.
- Usually referred to as SGD even when mini-batches are used.

Method	Accuracy	Update Speed	Memory Usage	Online Learning
Batch gradient descent	Good	Slow	High	No
Stochastic gradient descent	Good (with annealing)	High	Low	Yes
Mini-batch gradient descent	Good	Medium	Medium	Yes

Table: Comparison of trade-offs of gradient descent variants

Challenges

- Choosing a **learning rate**.
- Defining an **annealing schedule**.
- Updating features to **different extent**.
- **Avoiding suboptimal minima**.

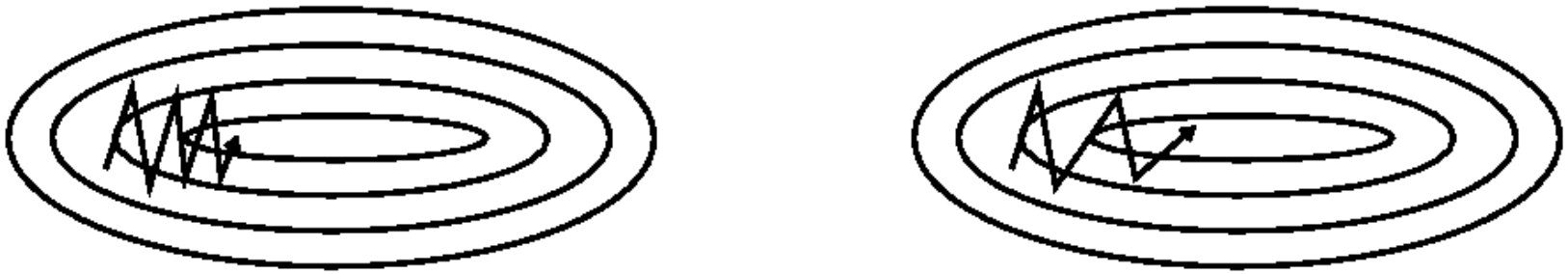
Gradient descent optimization algorithms

1. Momentum
2. Nesterov accelerated gradient
3. Adagrad
4. Adadelta
5. RMSprop
6. Adam
7. Adam extensions

Momentum

- SGD has trouble navigating **ravines**.
- Momentum [Qian, 1999] helps SGD **accelerate**.
- Adds a fraction γ of the update vector of the past step v_{t-1} to current update vector v_t. Momentum term γ is usually set to 0.9.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{1}$$



(a) SGD without momentum

(b) SGD with momentum

Figure: Source: Genevieve B. Orr

- **Reduces updates** for dimensions whose gradients **change directions**.
- **Increases updates** for dimensions whose gradients **point in the same directions**.

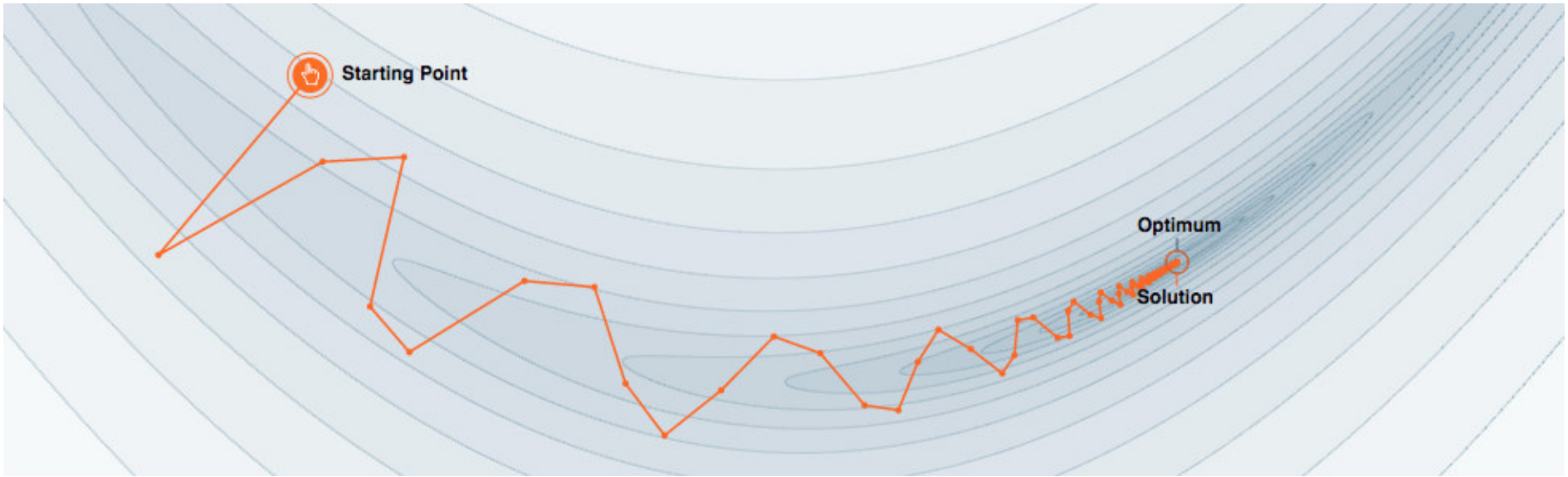


Figure: Optimization with momentum (Source: distill.pub)

Nesterov accelerated gradient

- Momentum **blindly accelerates** down slopes: First computes gradient, then makes a big jump.
- Nesterov accelerated gradient (NAG) [Nesterov, 1983] first makes a big jump in the direction of the previous accumulated gradient $\theta - \gamma v_{t-1}$. Then measures where it ends up and makes a correction, resulting in the complete update vector.

$$\begin{aligned} v_t &= \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned} \tag{2}$$

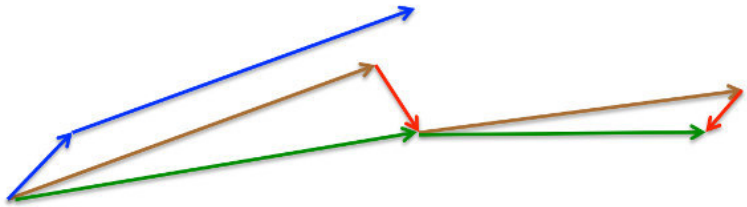


Figure: Nesterov update (Source: G. Hinton's lecture 6c)

Adagrad

- Previous methods: **Same learning rate** η for all parameters θ.
- Adagrad [Duchi et al., 2011] **adapts** the learning rate to the parameters (**large** updates for **infrequent** parameters, **small** updates for **frequent** parameters).
- SGD update: $\theta_{t+1} = \theta_t - \eta \cdot g_t$
  - $g_t = \nabla_{\theta_t} J(\theta_t)$
- Adagrad divides the learning rate by the **square root of the sum of squares of historic gradients**.
- Adagrad update:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{3}$$

  - $G_t \in \mathbb{R}^{d \times d}$: diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t
  - ϵ: smoothing term to avoid division by zero
  - $\odot$: element-wise multiplication

- Pros
  - Well-suited for dealing with **sparse data**.
  - Significantly **improves robustness** of SGD.
  - Lesser need to manually tune learning rate.
- Cons
  - **Accumulates squared gradients** in denominator. Causes the learning rate to **shrink** and become **infinitesimally small**.

Adadelta

- Adadelta [Zeiler, 2012] restricts the window of accumulated past gradients to a **fixed size**. SGD update:

$$\begin{aligned} \Delta\theta_t &= -\eta \cdot g_t \\ \theta_{t+1} &= \theta_t + \Delta\theta_t \end{aligned} \tag{4}$$

- Defines **running average** of squared gradients $E[g^2]_t$ at time t:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{5}$$

  - γ: fraction similarly to momentum term, around 0.9
- Adagrad update:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{6}$$

- Preliminary Adadelta update:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{7}$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{8}$$

- Denominator is just root mean squared (RMS) error of gradient:

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t \tag{9}$$

- Note: **Hypothetical units do not match**.
- Define **running average of squared parameter updates** and RMS:

$$\begin{aligned} E[\Delta\theta^2]_t &= \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2 \\ RMS[\Delta\theta]_t &= \sqrt{E[\Delta\theta^2]_t + \epsilon} \end{aligned} \tag{10}$$

- Approximate with $RMS[\Delta\theta]_{t-1}$, replace η for **final Adadelta update**:

$$\begin{aligned} \Delta\theta_t &= -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \\ \theta_{t+1} &= \theta_t + \Delta\theta_t \end{aligned} \tag{11}$$

RMSprop

- Developed independently from Adadelta around the same time by Geoff Hinton.
- Also divides learning rate by a **running average of squared gradients**.
- RMSprop update:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{12}$$

  - γ: decay parameter; typically set to 0.9
  - η: learning rate; a good default value is 0.001

Adam

- Adaptive Moment Estimation (Adam) [Kingma and Ba, 2015] also stores **running average of past squared gradients** v_t like Adadelta and RMSprop.
- Like Momentum, stores **running average of past gradients** m_t.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{13}$$

  - m_t: first moment (mean) of gradients
  - v_t: second moment (uncentered variance) of gradients
  - β_1, β_2: decay rates

- m_t and v_t are initialized as 0-vectors. For this reason, they are biased towards 0.
- Compute bias-corrected first and second moment estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{14}$$

- Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{15}$$

Update equations

Method	Update equation
	$g_t = \nabla_{\theta_t} J(\theta_t)$
SGD	$\Delta\theta_t = -\eta \cdot g_t$
	$\theta_t = \theta_t + \Delta\theta_t$
Momentum	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta g_t$
NAG	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta \nabla_\theta J(\theta - \gamma v_{t-1})$
Adagrad	$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$
Adadelta	$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$
RMSprop	$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
Adam	$\Delta\theta_t = -\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

Table: Update equations for the gradient descent optimization algorithms.

Which optimizer to choose?

- Adaptive learning rate methods (Adagrad, Adadelta, RMSprop, Adam) are **particularly useful for sparse features**.
- Adagrad, Adadelta, RMSprop, and Adam work well in similar circumstances.
- [Kingma and Ba, 2015] show that bias-correction helps Adam **slightly outperform RMSprop**.

Additional strategies for optimizing SGD

1. Shuffling and Curriculum Learning [Bengio et al., 2009]
   - Shuffle training data after every epoch to **break biases**
   - Order training examples to **solve progressively harder problems**; infrequently used in practice
2. Batch normalization [Ioffe and Szegedy, 2015]
   - **Re-normalizes every mini-batch** to zero mean, unit variance
   - Must-use for computer vision
3. Early stopping
   - "*Early stopping (is) beautiful free lunch*" (Geoff Hinton)
4. Gradient noise [Neelakantan et al., 2015]
   - Add Gaussian noise to gradient
   - Makes model **more robust to poor initializations**

Tuned SGD vs. Adam

- Many recent papers use **SGD with learning rate annealing**.
- SGD with tuned learning rate and momentum is **competitive with Adam** [Zhang et al., 2017b].
- Adam **converges faster**, but **underperforms SGD** on some tasks, e.g. Machine Translation [Wu et al., 2016].
- Adam with **2 restarts and SGD-style annealing** converges faster and outperforms SGD [Denkowski and Neubig, 2017].
- **Increasing the batch size** may have the same effect as decaying the learning rate [Smith et al., 2017].

Learning to optimize

- Rather than manually defining an update rule, **learn it**.
- Update rules outperform existing optimizers and transfer across tasks.

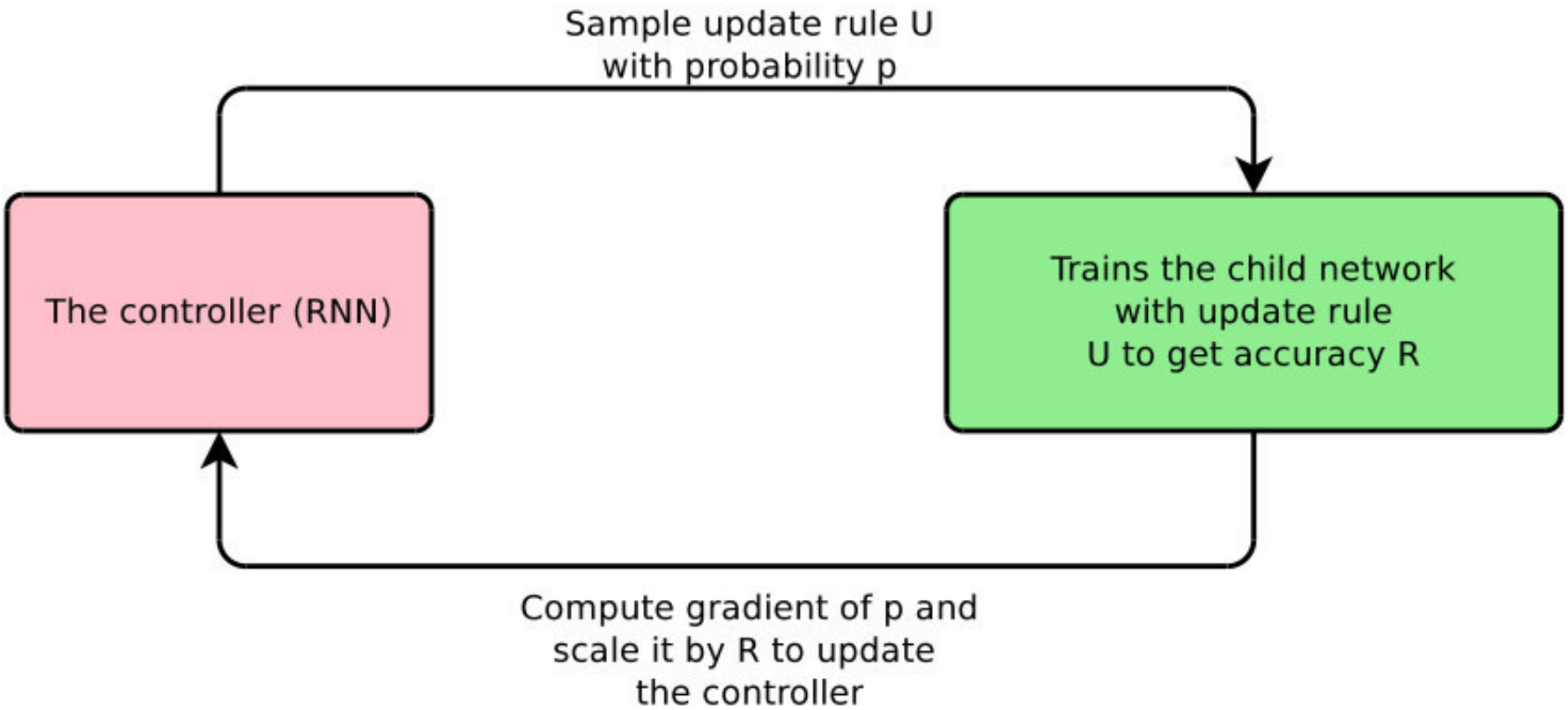


Figure: Neural Optimizer Search [Bello et al., 2017]

Discovered update rules

- **PowerSign**:

$$\alpha^{f(t)*\text{sign}(g)*\text{sign}(m)} * g \tag{16}$$

  - α: often e or 2
  - f: either 1 or a decay function of the training step t
  - m: moving average of gradients
  - Scales update by $\alpha^{f(t)}$ or $1/\alpha^{f(t)}$ depending on whether the direction of the gradient and its running average agree.

- **AddSign**:

$$(\alpha + f(t) * \text{sign}(g) * \text{sign}(m)) * g \tag{17}$$

  - α: often 1 or 2
  - Scales update by $\alpha + f(t)$ or $\alpha - f(t)$.

Understanding generalization in Deep Learning

- **Optimization** is closely tied to **generalization**.
- The number of possible local minima **grows exponentially with the number of parameters** [Kawaguchi, 2016].
- Different local minima **generalize to different extents**.
- Recent insights in understanding generalization:
  - Neural networks can **completely memorize random inputs** [Zhang et al., 2017a].
  - Sharp minima found by batch gradient descent have **high generalization error** [Keskar et al., 2017].
  - Local minima that generalize well can be **made arbitrarily sharp** [Dinh et al., 2017].
- Several submissions at ICLR 2018 on understanding generalization.