Optimisation algorithms in Statistics I, lecture 3

Frank Miller, Department of Statistics; Stockholm University

October 23, 2020

Stockholm University

Course schedule

- Topic 1: **Gradient based algorithms**
  Lectures: October 2; Time 10-12, 13-15 (online, Zoom)
- Topic 2: **Stochastic gradient based algorithms**
  Lecture: October 13; Time: 9-12 (online, Zoom)
- Topic 3: **Gradient free algorithms**
  Lecture: October 23; Time 9-12 (online, Zoom)
- Topic 4: **Optimisation with restrictions**
  Lecture: November 6, Time 9-12 (online, Zoom)

Course homepage: http://gauss.stat.su.se/phd/oasi/

Includes reading material, lecture notes, assignments

Stockholm University

Today’s schedule

- Stochastic gradient based methods
  - Adaptive step sizes for stochastic steepest ascent
- Gradient free methods
  - Nelder-Mead algorithm
  - Simulated annealing
  - Particle swarm optimisation

Stochastic steepest ascent method – momentum method

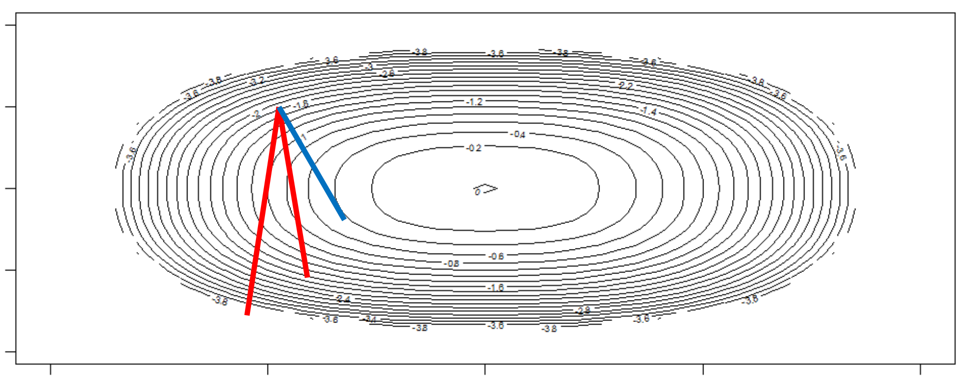
- Stochastic steepest ascent $\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \alpha \boldsymbol{g}'_i\left(\boldsymbol{x}^{(t)}\right)$ can be combined with momentum method (see Goodfellow, Bengio, Courville, 2016, Chapter 8.3.2)
- Iteration:
  - Choose $i \in \{1, \dots, n\}$ randomly
  - $\boldsymbol{v}^{(t+1)} = \beta \boldsymbol{v}^{(t)} + \boldsymbol{g}'_i\left(\boldsymbol{x}^{(t)}\right)$
  - $\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \alpha \boldsymbol{v}^{(t+1)}$
- Advantages:
  - Momentum advantages (handling ill-conditioning, accelerating)
  - Information from previous gradients contribute (variance of stochastic gradient reduced)

Stochastic steepest ascent method – momentum method

- Both hyperparameters α, β may depend on iteration number
- Iteration:
  - Choose $i \in \{1, \dots, n\}$ randomly
  - $\boldsymbol{v}^{(t+1)} = \beta^{(t)} \boldsymbol{v}^{(t)} + \boldsymbol{g}'_i\left(\boldsymbol{x}^{(t)}\right)$
  - $\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \alpha^{(t)} \boldsymbol{v}^{(t+1)}$
- Changing hyperparameters:
  - $\beta^{(t)}$ usually increased with t, common values 0.5 to 0.99
  - $\alpha^{(t)}$ is decreased with t
  - Decreasing $\alpha^{(t)}$ more important than changing $\beta^{(t)}$

Stochastic steepest ascent method – adaptive step sizes

- Stochastic steepest ascent:
  - Choose $i \in \{1, \dots, n\}$ randomly
  - $\boldsymbol{x}^{(\boldsymbol{t}+\mathbf{1})} = \boldsymbol{x}^{(\boldsymbol{t})} + \alpha^{(t)} \boldsymbol{g}'_i\left(\boldsymbol{x}^{(\boldsymbol{t})}\right)$
- $\alpha^{(t)}$ is now adapted automatically based on previous iterations and separately for each dimension
- If previous gradients in a dimension were large, we want to reduce step size more



Stockholm University

Stochastic steepest ascent method – adaptive step sizes, AdaGrad

- AdaGrad:
  $\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \mathrm{diag}(\boldsymbol{\alpha}^{(t)})\boldsymbol{g}'_i\left(\boldsymbol{x}^{(t)}\right)$ with vector $\boldsymbol{\alpha}^{(t)}$
- $\alpha_j^{(t)} = \alpha / \sqrt{\epsilon + \sum_{k=1}^{t} (g'_{j,k})^2}$
  - $g'_{j,k}$ is j[th] partial derivative of gradient in iteration k
  - ϵ is small constant (e.g. 1e-8)
- A default value $\alpha = 0.01$ is a popular choice

Stockholm University

Stochastic steepest ascent method – adaptive step sizes, AdaDelta

- AdaGrad:
  - Choose $i \in \{1, \ldots, n\}$ randomly
  - $\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \mathrm{diag}(\boldsymbol{\alpha}^{(t)})\boldsymbol{g}'_i\left(\boldsymbol{x}^{(t)}\right)$
  - $\alpha_j^{(t)} = \alpha / \sqrt{\epsilon + \sum_{k=1}^{t} (g'_{j,k})^2}$
- Disadvantage: $\alpha_j^{(t)}$ can only decrease
- AdaDelta:
  - $\alpha_j^{(t)} = \alpha / \sqrt{\epsilon + h_j^{(t)}}$
  - $h_j^{(t)} = \gamma h_j^{(t-1)} + (1-\gamma)(g'_{j,t})^2$
    (exponential smoothing of earlier partial derivatives; popular choice of γ is around 0.9)

Stockholm University

Stochastic steepest ascent method – adaptive step sizes, Adam

- AdaDelta:
  - Random $i \in \{1, \ldots, n\}$; $\boldsymbol{x}^{(\boldsymbol{t}+\mathbf{1})} = \boldsymbol{x}^{(\boldsymbol{t})} + \mathrm{diag}(\boldsymbol{\alpha}^{(t)})\boldsymbol{g}'_i\left(\boldsymbol{x}^{(\boldsymbol{t})}\right)$
  - $\alpha_j^{(t)} = \alpha / \sqrt{\epsilon + h_j^{(t)}}$; $h_j^{(t)} = \gamma h_j^{(t-1)} + (1-\gamma)(g'_{j,t})^2$

- Adam (”Adaptive moment estimation”)
  - Random $i \in \{1, \ldots, n\}$; $\boldsymbol{x}^{(\boldsymbol{t}+\mathbf{1})} = \boldsymbol{x}^{(\boldsymbol{t})} + \mathrm{diag}(\boldsymbol{\alpha}^{(t)})\widehat{\boldsymbol{m}}_t$;
  - $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}'_{i(t)}\left(\boldsymbol{x}^{(\boldsymbol{t})}\right)$ $\quad \widehat{\boldsymbol{m}}_t = \boldsymbol{m}_t/(1-\beta_1)$
  - $\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)((g'_{j,t})^2)_{j=1,\ldots,p}$ $\quad \widehat{\boldsymbol{v}}_t = \boldsymbol{v}_t/(1-\beta_2)$
  - $\alpha_j^{(t)} = \alpha / \sqrt{\epsilon + \hat{v}_{j,t}}$
- Default values $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$

Stockholm University

Stochastic steepest ascent method – adaptive step sizes, summary

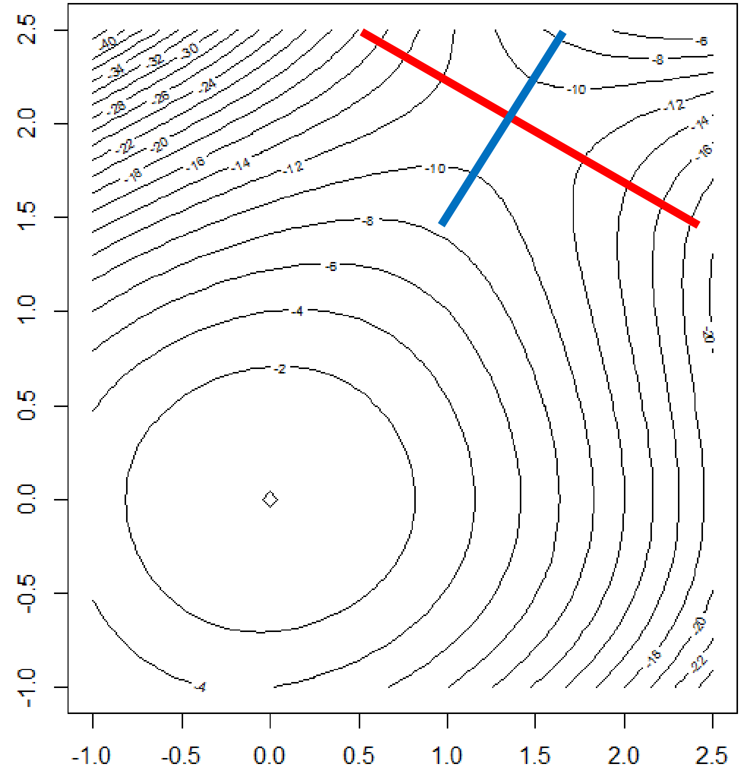
- Momentum method can be added to AdaGrad and AdaDelta
- AdaGrad works well for concave functions
- AdaDelta handles non-concave functions better
- In Adam, momentum method already included

Steepest ascent – comparisons of methods

- Animated comparisons:
  - https://imgur.com/a/Hqolp

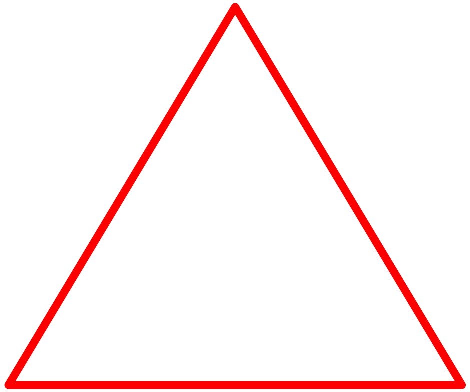
Saddle point and eigen-vectors of the Hessian

- $\boldsymbol{g}\begin{pmatrix} x \\ y \end{pmatrix} = -3x^2 - 4y^2 + xy^3$
- Saddle point at (4/3,2)
- $\boldsymbol{g}'\begin{pmatrix} 4/3 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$
- $\boldsymbol{g}''\begin{pmatrix} 4/3 \\ 2 \end{pmatrix} = \begin{pmatrix} -6 & 12 \\ 12 & 8 \end{pmatrix}$
- Eigenvalues 14.89, -12,89; eigenvectors $\begin{pmatrix} 0.498 \\ 0.867 \end{pmatrix}$, $\begin{pmatrix} -0.867 \\ 0.498 \end{pmatrix}$



Stockholm University

Gradient free optimisation – Nelder-Mead method



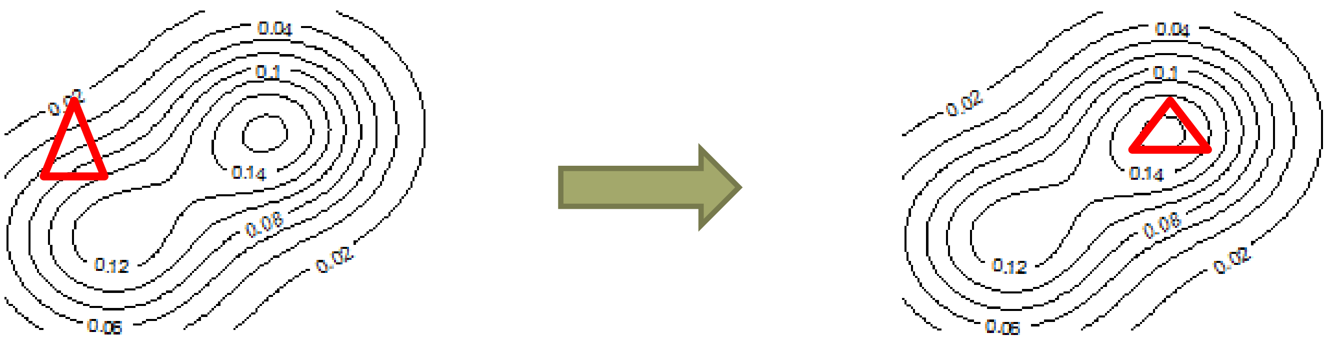
Gradient free optimisation – Nelder-Mead method

- $\boldsymbol{x}$ $p-$dimensional vector, $g: \mathbb{R}^p \rightarrow \mathbb{R}$ function
- We search $\boldsymbol{x}^*$ with $g(\boldsymbol{x}^*) = \max g(\boldsymbol{x})$

- Nelder-Mead method is heuristic method for *p*-dimensional optimisation problem (default in R-function **`optim`**)
- Positive:
  + No computation of derivatives necessary
- Negative:
  – Does not necessarily converge (however no general guarantee for other methods either…)
  – Might be slow
- Works often well, especially if *p* not too large

Gradient free optimisation – Nelder-Mead method

- Idea: Work with simplex of $p+1$ points;
  i.e. for two-dimensional optimisation: work with triangle
- Aim that triangle includes maximum
- Choose arbitrary starting triangle
- Change vertices to ”move the triangle upwards”



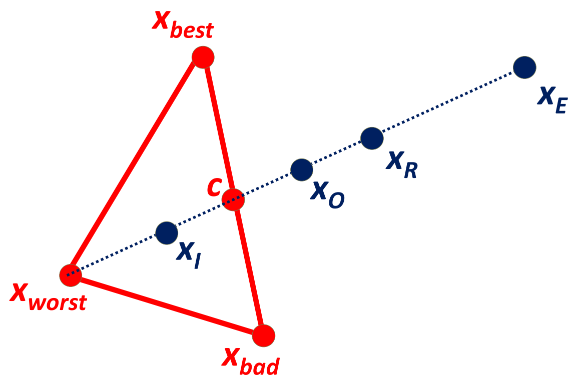
- Two animations:
  - https://www.youtube.com/watch?v=HUqLxHfxWqU
  - https://www.youtube.com/watch?v=KEGSLQ6TlBM

Gradient free optimisation – Nelder-Mead method

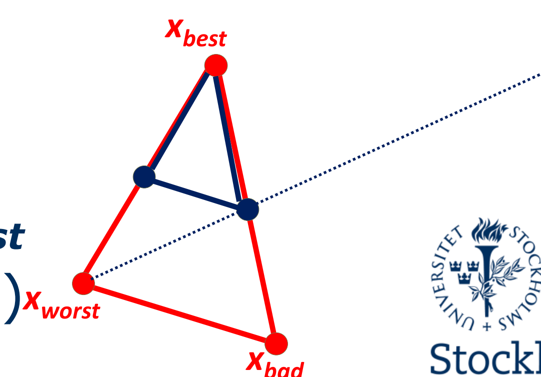
- Identify worst vertex $\boldsymbol{x}_{\boldsymbol{worst}}$ ($g(\boldsymbol{x}_{\boldsymbol{worst}})$ minimal among all vertices) and compute average $\boldsymbol{c}$ of remaining vertices
- Let $\boldsymbol{x}_{\boldsymbol{best}}$ be best and $\boldsymbol{x}_{\boldsymbol{bad}}$ be second worst vertex
- Rules for
  - Reflection
  - Expansion
  - Outer contraction
  - Inner contraction
  - Shrinkage

Gradient free optimisation – Nelder-Mead method

- Replace $\boldsymbol{x}_{\boldsymbol{worst}}$ with one of $\boldsymbol{x}_{\boldsymbol{I}}$, $\boldsymbol{x}_{\boldsymbol{O}}$, $\boldsymbol{x}_{\boldsymbol{R}}$, $\boldsymbol{x}_{\boldsymbol{E}}$ (rule depends on values for $g(\boldsymbol{x}_{\boldsymbol{worst}})$, $g(\boldsymbol{x}_{\boldsymbol{bad}})$, $g(\boldsymbol{x}_{\boldsymbol{best}})$, $g(\boldsymbol{x}_{\boldsymbol{I}})$, $g(\boldsymbol{x}_{\boldsymbol{O}})$, $g(\boldsymbol{x}_{\boldsymbol{R}})$, $g(\boldsymbol{x}_{\boldsymbol{E}})$; see Givens and Hoeting, page 47-48) and create new simplex/triangle



- Or in specific cases: Shrink (keep $\boldsymbol{x}_{\boldsymbol{best}}$ and move all other vertices towards it)



Stockholm University

Gradient free optimisation – Nelder-Mead method

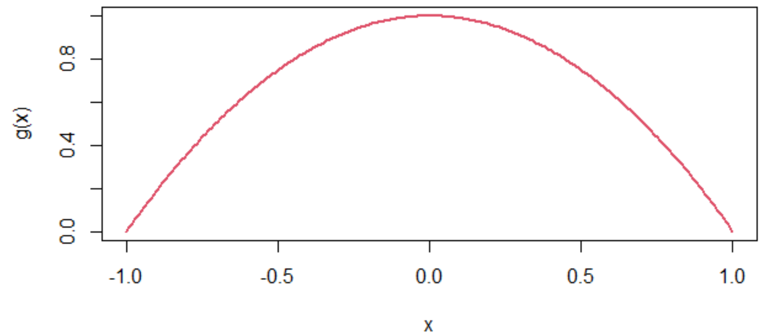
- Nelder-Mead algorithm is quite old, but still popular
- Research is ongoing e.g. about convergence results and variants of Nelder-Mead

- Note that Nelder-Mead can be used for dimension p=1 as well

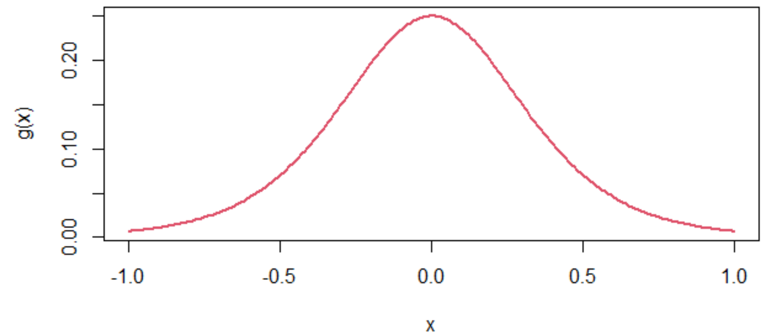
Stockholm University

Convexity / Concavity and log likelihood

- Function g concave, if $g((\mathbf{x}+\mathbf{y})/2) \geq (g(\mathbf{x})+g(\mathbf{y}))/2$ for all $\mathbf{x},\mathbf{y}$



concave



non-concave

- If g is concave, a local maximum is a global maximum

- Log likelihood for exponential families is concave
- Log likelihoods can be non-concave (e.g. Problem 1.1)
- Deep learning optimisation problems are often non-concave / non-convex and have multiple local extrema

Gradient free optimisation – Simulated annealing

AlphaOpt (2017). Introduction To Optimization: Gradient Free Algorithms (2/2) – Simulated Annealing, Nelder-Mead (0:15-1:35)

Gradient free optimisation – Simulated annealing

- Start value $x^{(0)}$; Stage j=0,1,2,… has m_j iterations; set j=0
- Given iteration $x^{(t)}$, generate $x^{(t+1)}$ as follows:

1. Sample a candidate x^* from a proposal distribution $p(\cdot|x^{(t)})$
2. Compute $h\left(x^{(t)}, x^*\right) = \exp\left(\frac{g(x^*)-g\left(x^{(t)}\right)}{\tau_j}\right)$

   $g\left(x^{(t)}\right) - g(x^*)$
   for minimisation
3. Define next iteration $x^{(t+1)}$ according to

$$x^{(t+1)} = \begin{cases} x^*, \text{with probability} \min\{h\left(x^{(t)}, x^*\right), 1\} \\ x^{(t)}, \text{otherwise} \end{cases}$$

4. Set `t <- t+1` and repeat 1.-3. m_j times
5. Update $\tau_j = \alpha(\tau_{j-1})$ and $m_j = \beta(m_{j-1})$; set `j <- j+1`; go to 1

τ_j is temperature; function α should slowly decrease it

Function β should be increasing

Stockholm University

Gradient free optimisation – Simulated annealing

- Initially, also “bad” proposals are accepted
- With decreasing temperature, accept only improvements
- This helps to explore first and avoids convergence to a local maximum too early
- Algorithm which has therefore chances to find the global optimum in presence of multiple local optima

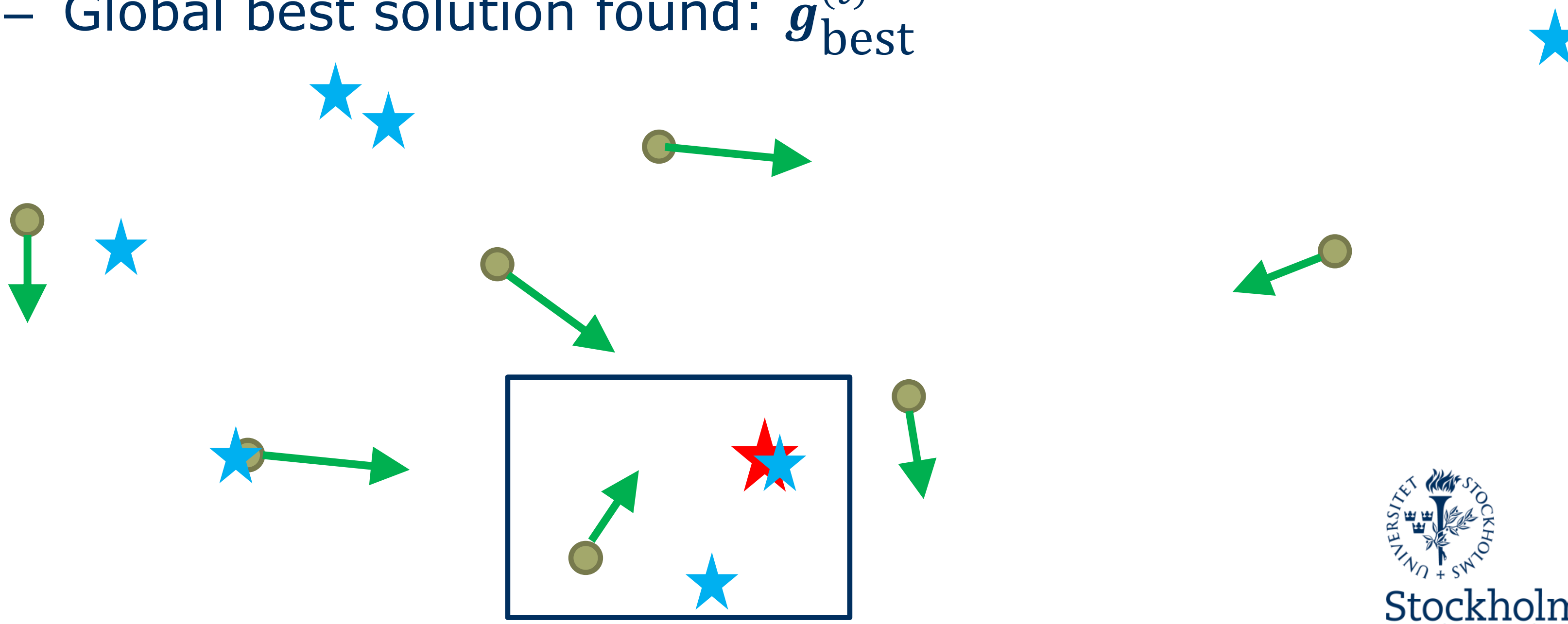
- **`method="SANN"`** of R function **`optim`** is “a variant of simulated annealing” (documentation of **`optim`**)
  - Initial temperature seems to be important choice (can be changed e.g. by **`control=list(temp=0.01)`**; default 10 might be bad)

Stockholm University

Gradient free optimisation – Particle swarm optimisation

Gradient free optimisation – Particle swarm optimisation

- Swarm of N particles
  - Position of particle i at iteration t+1: $\boldsymbol{x}_i^{(t+1)}$
  - Velocity of particle i at iteration t+1: $\boldsymbol{v}_i^{(t+1)}$
- Best positions found so far:
  - Best location found by particle i: $\boldsymbol{p}_{\text{best}, i}^{(t)}$
  - Global best solution found: $\boldsymbol{g}_{\text{best}}^{(t)}$



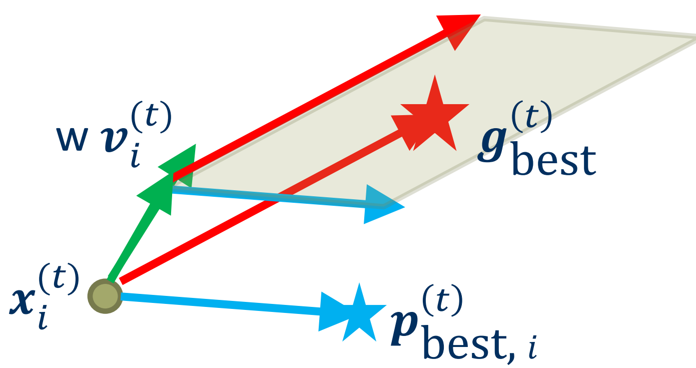
Stockholm University

Gradient free optimisation – Particle swarm optimisation

- Movement of particle i at iteration t+1:
  - $\boldsymbol{x}_i^{(t+1)} = \boldsymbol{x}_i^{(t)} + \boldsymbol{v}_i^{(t+1)}$
  - $\boldsymbol{v}_i^{(t+1)} = \mathrm{w}\boldsymbol{v}_i^{(t)} + c_1 R_1^{(t+1)} (\boldsymbol{p}_{\text{best, } i}^{(t)} - \boldsymbol{x}_i^{(t)}) + c_2 R_2^{(t+1)} (\boldsymbol{g}_{\text{best}}^{(t)} - \boldsymbol{x}_i^{(t)})$

cognitive component social component

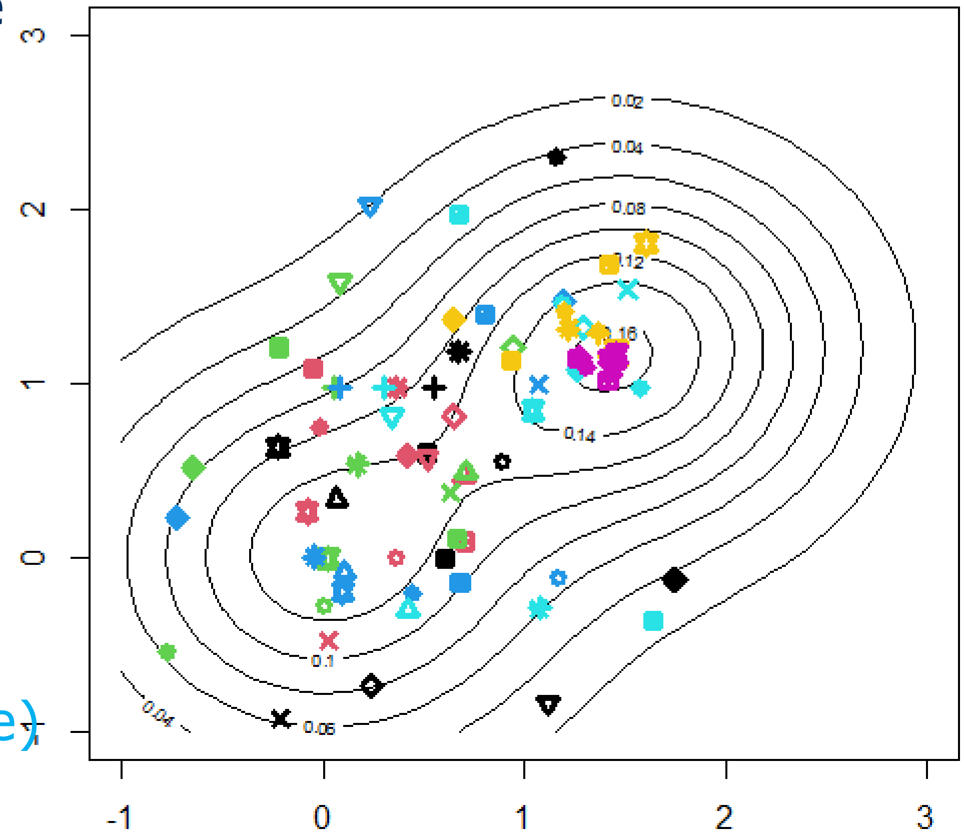
- $R_1^{(t+1)}$ and $R_2^{(t+1)}$ are uniformly distributed, **`runif()`**



Stockholm University

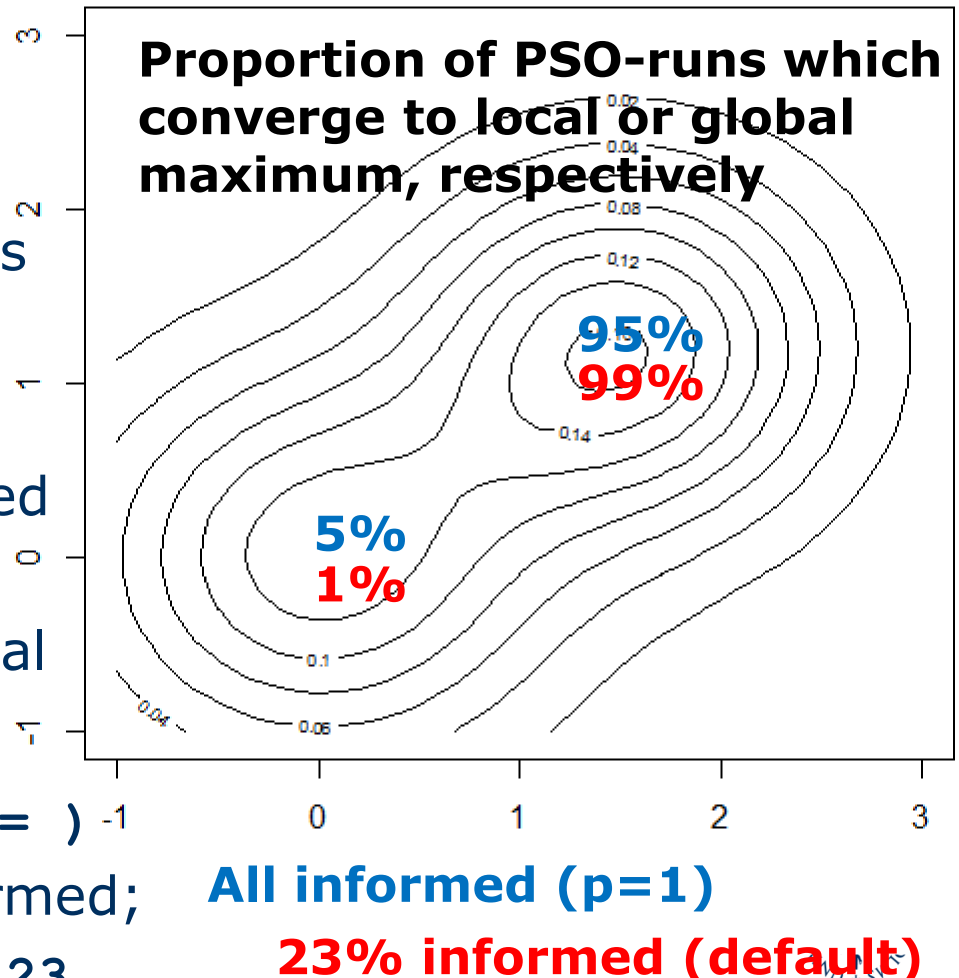
Gradient free optimisation – Particle swarm optimisation

- Bimodal normal mixture example from Lecture 1
- PSO with s=12 particles using **`psoptim`** (in **`R`**-package **`pso`**)
  - Iteration 1 (black)
  - Iteration 2 (red)
  - Iteration 3 (green)
  - Iteration 4 (blue)
  - Iteration 5 (light blue)
  - Iteration 20 (yellow)
  - Iteration 40 (pink)



Gradient free optimisation – Particle swarm optimisation

- Bimodal normal mixture example from Lecture 1
- In some runs, the local maximum is identified as global maximum
- Risk to remain at a local maximum can be reduced if not all particles are informed about the global maximum
- Option **`control=list(p= )`** controls proportion informed; default **`1-(11/12)^3=0.23`**.



Stockholm University

Gradient free optimisation – Particle swarm optimisation

- Example call:
- `pso <- psoptim(par=rep(NA,2),` Dimension of problem

```
pso <- psoptim(par=rep(NA,2),
               fn=g,
               lower=-1, upper=3,
               control=list(
                 fnscale=-1,
                 maxit=1000,
                 p=0.23,
                 s=12
               )).
```

Dimension of problem

Function to optimise

Search space

For maximisation

Iteration number; *default can be too large in many situations*

Proportion informed

Swarm size; *default can be too low in some situations*

Running time roughly linear in each of these two parameters

- Some further options: **c.p=** c_1 (cognitive comp.), **c.g=** c_2 (social comp.), **w=** w (exploitation const.), **trace=1** (output of tracing info)

Stockholm University