## 1 inertial Component

maintains a fraction of the current velocity vector, by continuing along the same direction as during previous step

## 2 Cognitive component:

- The difference between the current position and the particle's best position

  → local memory effect

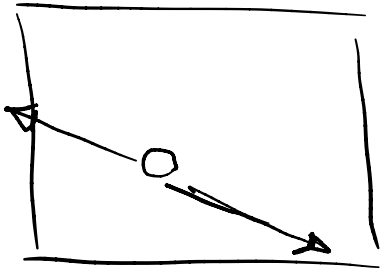
## 3 Social Component:

Proportional to the difference of the current agent position and the current best position of the entire swarm

The final expression for the global velocity update:

$$\underline{x}^{(i)} \leftarrow \underline{x}^{(i)} + \underline{v}^{(i)}$$

$$\underline{v}^{(i)} \leftarrow \underline{\omega}\,\underline{v}^{(i)} + \underline{c_1 r_1}\left(x_{best}^{(i)} - x^{(i)}\right) + \underline{c_2 r_2}\left(x_{best} - x^{(i)}\right)$$



$w, c_1, c_2$ : parameters → deterministic scalar components

↳ allow user to weight the various components of the velocity

$r_1, r_2$ : random numbers drawn from $\mathcal{U}(0,1)$

↳ parabolistic components allowing exploration of other parts of the space ⇒ not local minima

two concepts in balance :

- exploitation
- exploration

Exploitation :

communication of the global minima to all particles → general tendency of the particles to follow a "leading" Particle

## Exploration:

- Induced by random coefficients
- allows probing of soln. away from optima

## Accounting for constraints:

General Optimisation problem

Cost/objective function $f(x_1, x_2, \ldots x_n)$ → Optimise

equality constraints $g_j(x_1, x_2, \ldots, x_n) = 0 \quad j = 1, \ldots, J$

inequality constraints $h_k(x_1, \ldots, x_n) \leq 0 \quad k = 1, \ldots, K$

$$l_i \leq x_i \leq h_i$$

↳ restrict the soln. to given intervals

- Simplest one → constraint on the domain

* enforce by monitoring the Soln and clipping the components of the soln. Vector back to permissible interval

* due to the inertial part of the velocity component the particle can leave the interval

  ⟹ change sign | Velocity mirroring |

- Also apply limits on the velocity at each iteration

  * avoid excessive speed along soln. space
  * more careful exploitation

Additional constraints :

- equality
- inequality

for inequality constraints:

(i) feasibility check

→ check whether potential new soln $x_i^{(n+1)}$ satisfy all constraints

(yes) → kept

(no) → $x_i^{(n)} = (x_i)^{(n+1)}$

→ works for simple constraints

→ low-dimentional problems

(ii) adding the constraint to the cost functional

→ Penalty term

$f$ ---- $+ \varepsilon \| h \|$

→ low-dimentional problems

For equality constraints:

More difficult to implement

(i) non-linear Newton-Raphson

(ii) Approximate Jacobian based on current particle position

## Stochastic Methods:

- Randomisation Strategically to explore the design space
- Global minima
- Use pseudo-random number generators to ensure repeatability

# Noisy Descent:

Adding Stochasiticity to gradient descent

- large nonlinear optimisation problems
- Saddle points → gradient very close to zero

$$x^{(k+1)} = x^{(k)} + \alpha g^{(k)} + \varepsilon^{(k)}$$

$\varepsilon^{(k)}$: zero-mean Gaussian noise with standard deviation "$\sigma$"

- Amount of noise is usually reduced overtime

  eg. a decreasing sequence: $\sigma^{(k)} = 1/k^3$

→ A common approach for training networks

- traverse past saddle points
- evaluate noisy gradients using randomly chosen subsets of training data

## Simulated Annealing:

- Borrows inspiration from metallurgy
- Temperature is used to control the degree of stochasticity during randomised search
- High Temperature in the beginning
  ⇒ explore the space
  ⇒ lower temperature to converge to an optimum

- works well on functions with many local minima

→ At every iteration, a transition from $x$ to $x'$ is sampled from transition distribution $T$ with probability:

$$\Delta y = f(x') - f(x)$$

Metropolis Criterion:

$$\begin{cases} 1 & \text{if } \Delta y \leq 0 \\ \min\left(e^{-\Delta y / t}, 1\right) & \text{if } \Delta y > 0 \end{cases}$$

$t$: temperature

- Temperature parameter $t$ controls the acceptance probability

- An annealing schedule is used to slowly bring down the Temperature as the algorithm progress

- If brought down too quickly you miss areas in space ⇒ less exploration

- logarithmic annealing

$$t^{(k)} = t^{(1)} \ln(2) / \ln(k+1)$$

for the "$k^{th}$" iteration is guaranteed to reach global optimum

⇒ very slow

- Exponential annealing schedule:

PS4

$$t^{(k+1)} = \gamma\, t^{(k)}$$

for some $\gamma \in (0,1)$

- fast annealing

$$t^k = \frac{t^{(1)}}{k}$$

→ This implementation can lead to different results based on transition distributions and annealing schedules

"

Corana et al 1987

↳ for step size to change during search

$$x' = x + r\, v_i e_i$$

r: max step size

# Linear Programming:

* Arise in many applications
  - operations research
  - research source allocation
  - transportation problems
  - task scheduling
  - economics

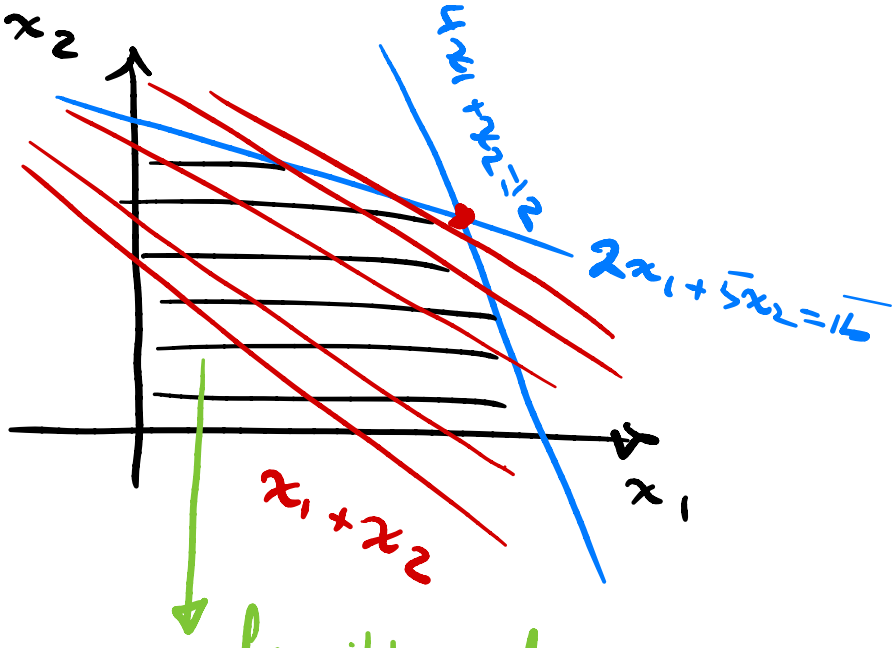
* Optimization of a linear cost objective subject to linear inequality constraints

* We have

$$\text{minimize} \quad c^T x$$

$$\text{subject to} \quad Ax \leq b$$

$$x \geq 0$$




Feasible domain → Convex Polytope

$a_j x = b_j$ → jth component of the b vector

↓

jth row of A

Methods to solve:

- Simplex method:

Optimal soln found on the boundary of the convex polytope

$(Ax=b)$ → move towards min cost funct.

$c^T x$

- <u>Interior Methods:</u>

Starts at non-optimal soln. → moving towards The min!

⇒ primal affine scaling algorithm

# Primal affine Scaling Algorithm:

Slightly modify the Opt. problem

$$\begin{cases} \text{minimize} & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{cases}$$

- guess $\Rightarrow$ $Ax = b$, $x \geq 0$ $\leftarrow$ satisfy

→ direction of descent $\Rightarrow -c$

perpendicular to the red lines of the figure

→ by moving in $c$ we might violate

(i) Constraint $Ax = b$ → projection

(ii) positivity constraint $x \geq 0$

→ decrease step size

→ Assume current Soln. $x$

* we seek a new (improved) soln. in the gradient direction $(-c)$

$$x_{new} = x - \xi c$$

⟹ preserve positivity

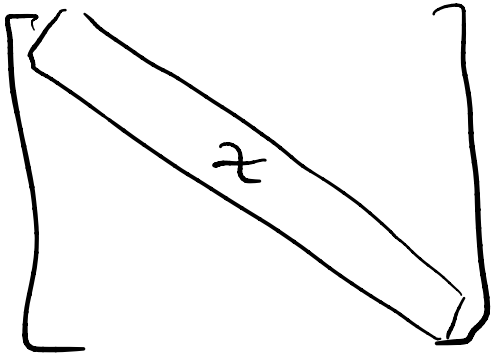
we have to modify $\xi$ (step size)

max possible step size:

$$\xi_{max} = \min_j \frac{x_j}{c_j}$$

* We can scale the optimisation problem using $x$ (current)

→ Diagonal matrix $D$ with $x$ along the diagonal

$D =$ 

transform:

minimize $\bar{c}^T y$ $\quad \bar{c} = Dc$

s.t $\bar{A} y = b$ $\quad \bar{A} = AD$

$y \geq 0$

The original feasible point = Vectors of ones

$x = D.e$

$\Rightarrow$ $y_{new} = e - \bar{S}\bar{C}$

$\bar{S}$ → step sized that are nearly One !

independant of the closness ← at the boundary

$\Rightarrow$ **Next step**

modify the constraints $\bar{A}y = b$

$y_{new}$ has to satisfy:

$$\bar{A}\underbrace{(e - \bar{S}\bar{C})}_{y_{new}} = b$$

knowing that e satisfeis the constraint $\Rightarrow$ $\boxed{\bar{A}e = b}$

So: $\bar{A}\bar{c} = 0$

$\bar{c}$ has to be in the nullspace of $\bar{A}$

$\Rightarrow$ need to project out (eliminate) components of $\bar{c}$ that do not fall into the nullspace $\bar{A}$

$$\Pi = \bar{I} - \bar{A}^T(\bar{A}\bar{A}^T)\bar{A}$$

multiply any vector by $\Pi$

it will eliminate all components that fall out of the null-space of $\bar{A}$.

- Our final update for

$y_{new}$

$$y_{new} = e - S\bar{\Pi}\bar{C}$$

↳ feasible design point
⇒ satisfies both constraint

→ scale back → $x_{new}$

## from inequality to equality constraints

* Algorithm derived for equality : $Ax=b$
* How to transform inequality to equality

$$Ax \leq b \rightsquigarrow Ax = b$$

→ through introduction of auxiliary variables (slack variables)

e.g. single constraint:

$ax \leq b$ → Scalar

$x \geq 0$ → given row vector

$\Rightarrow$ conversion:

$ax + z = b$, $x \geq 0$, $z \geq 0$

$z$ → slack variable

So the transformation in matrix form:

$$A \in \mathbb{R}^{n \times m} \longrightarrow [A \quad I_n] \in \mathbb{R}^{n \times (m+n)}$$

$$C \in \mathbb{R}^{m} \longrightarrow [C \quad 0_n] \in \mathbb{R}^{(m+n)}$$

$$x \in \mathbb{R}^{m} \longrightarrow [x \quad z] \in \mathbb{R}^{(m+n)}$$

# finding a feasible Soln:

→ first take a strictly positive starting vector $x_0 > 0$

→ define stack vector $f$:

residual of ← the equality-constraint system

$$f = b - Ax_0 \qquad (Ax^k = b)$$

$f = 0 \Rightarrow x_0$ is feasible $\Rightarrow$ proper starting vector → affine scaling Alg.

$f \neq 0 \Rightarrow$ process to determine a feasible Soln. from $x_0$

⇒ 1. extend the state vector $x \in \mathbb{R}^n$ by a scalar variable $\rho$

$$x \leftarrow [x \quad \rho]$$

2. Add scalar vector $f$ as an addition column on the right of the constraint matrix $A$

$$A \longrightarrow [A \mid f]$$

3. formulate the following linear programming problem

$$\begin{cases} \text{minimize} & \rho \\ \text{subject to} & Ax + \rho f = b \\ & x \geq 0 \\ & \rho \geq 0 \end{cases}$$

Affine scaling

in a more compact form.

minimize $\bar{c}^T \bar{x}$ $\quad \bar{c} \begin{bmatrix} 0_m & 1 \end{bmatrix}$

subject to $\bar{A}\bar{x} = b$ $\quad \bar{x} = \begin{bmatrix} x & \rho \end{bmatrix}$

$\bar{x} \geq 0$ $\quad \bar{A} = \begin{bmatrix} A & \beta \end{bmatrix}$

$\rho = 0 \longrightarrow$ recover a soln $x$ that satisfies the original constraint expressed by $A$ & $b$

$\rho^* = 0 \longrightarrow$ found a starting point for optimisation

$\rho^* \neq 0 \longrightarrow$ Problem is not feasible

$\longrightarrow$ app. starting soln not found

The problem has turned into another affine scaling problem!

$\Rightarrow$ feasible soon.

The design of $\bar{A}$ as the original $A$ augmented by $f$ allows us to form a feasible starting point as $\bar{x}_0 = [x_0 \quad 1]$

$$\bar{A}\bar{x}_0 = [A \quad f]\begin{bmatrix} x_0 \\ 1 \end{bmatrix} = Ax_0 + f$$

$$= Ax_0 + (b - Ax_0) = b$$

$\rightarrow \bar{x}_0$ satisfies the constraints!