Non–linear optimization using MapReduce with SystemML

Deepti Pachauri
(Mentor–Prithviraj Sen, Manager–Rajasekar Krishnamurthy)

Intern Talk

Overview

- Motivation
- ARIMA
  - ARIMA(1,0,1)
  - ARIMA(p,0,1)
- Optimization
  - Nelder-Mead
  - Distributed Nelder-Mead
  - CG
  - BFGS
- Solver
  - Jacobi
  - GMRES
- Experimental Results

Motivation

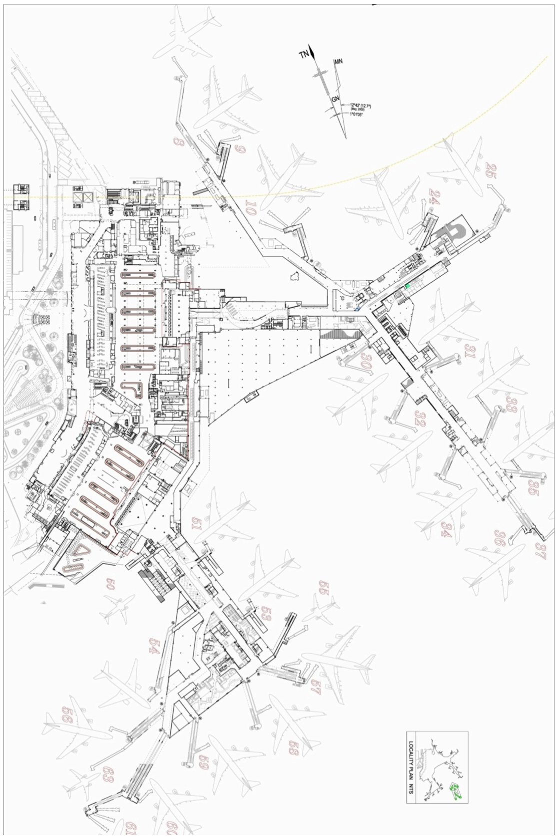
Non–linear optimization is required in many real life problems.

Motivation

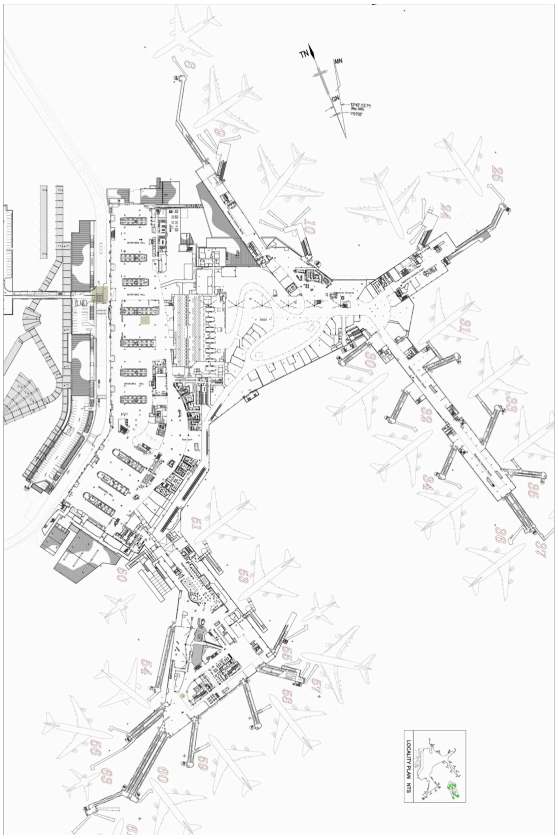
Non–linear optimization is required in many real life problems.

$\boxed{\textbf{Time Series Analysis}}$

Passengers on airport

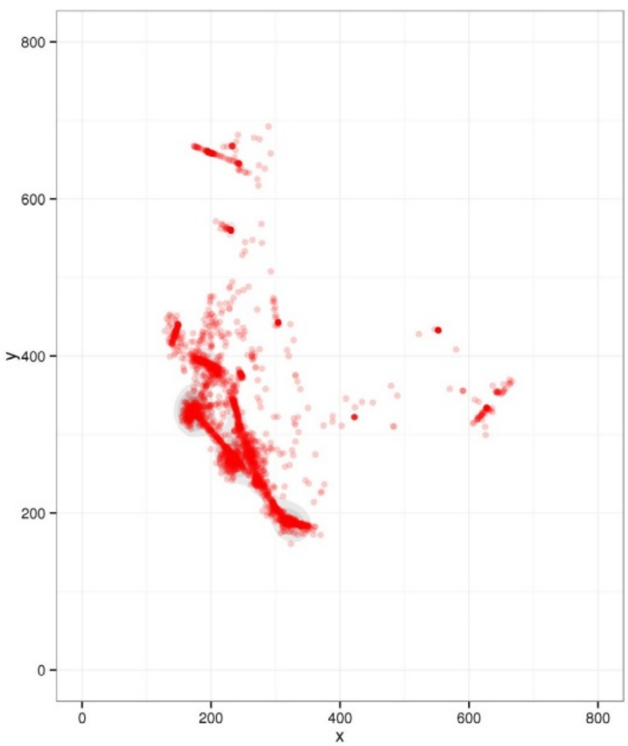


Arrivals

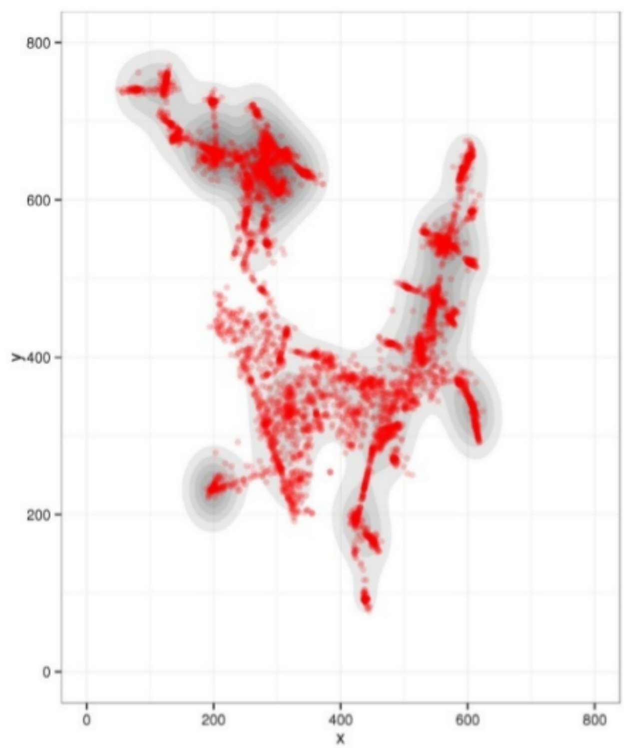


Departures

Passengers on airport

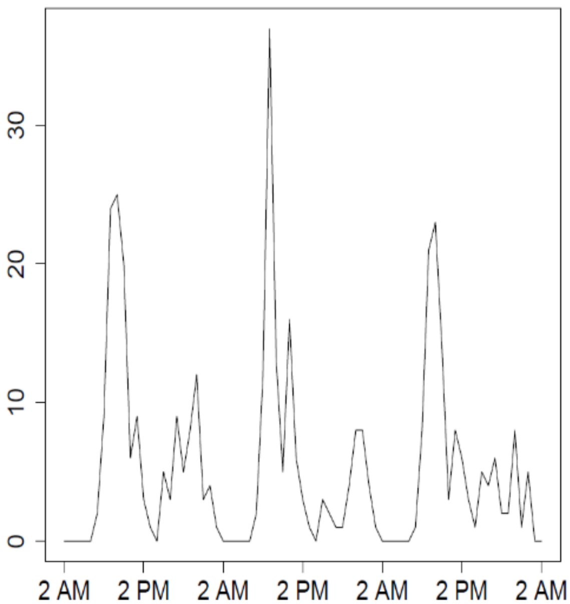


Arrivals

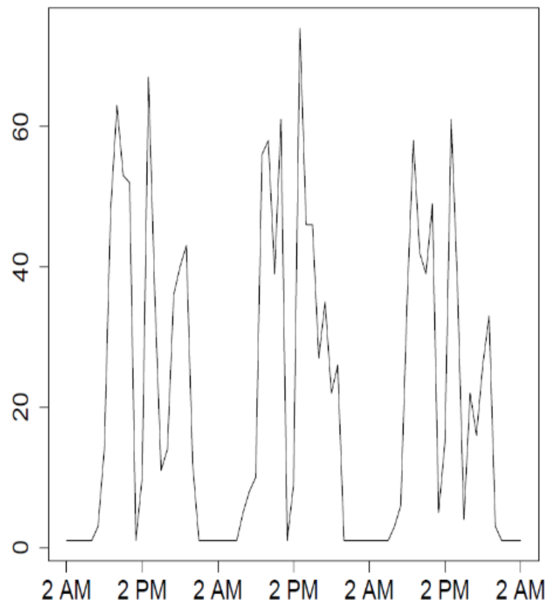


Departures

Passengers on airport



Arrivals



Departures

Passengers on airport

Given: Historical data of passenger movements across the airport.
Goal: Predict passenger counts at locations of interest, eg - immigration check points, security lines etc.
Why: Proper resource allocation and that makes life easy!!!!!!

ARIMA: AutoRegressive Integrated Moving Average

Give a time series data X_t

An ARIMA(p, d, q) process is expressed as

$$(1 - \sum_{i=1}^{p} \phi_i L^i)(1 - L)^d X_t = (1 + \sum_{i=1}^{q} \theta_i L^i)\epsilon_t$$

where

L is the **lag operator** given by $L^i X_t = X_{t-i}$

ϕ_i are the autoregressive parameters

θ_i are the moving average parameters

ϵ_t are i.i.d. error terms

Goal: Estimate ϕ_i and θ_i for given (p, d, q)

Formulation

ARIMA$(\mathbf{1}, \mathbf{0}, \mathbf{1})$

$$\hat{X}_2 = \phi_1 X_1 + \theta_1 (X_1 - \hat{X}_1)$$
$$\hat{X}_3 = \phi_1 X_2 + \theta_1 (X_2 - \hat{X}_2)$$
$$\hat{X}_4 = \phi_1 X_3 + \theta_1 (X_3 - \hat{X}_3)$$ .....and so on

Formulation

ARIMA$(\mathbf{1}, \mathbf{0}, \mathbf{1})$

$$\hat{X}_2 = \phi_1 X_1 + \theta_1(X_1 - \hat{X}_1)$$
$$\hat{X}_3 = \phi_1 X_2 + \theta_1(X_2 - \hat{X}_2)$$
$$\hat{X}_4 = \phi_1 X_3 + \theta_1(X_3 - \hat{X}_3)\text{.....and so on}$$

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \theta_1 & 1 & 0 & \cdots & 0 \\ 0 & \theta_1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & \cdots & \theta_1 & 1 \end{pmatrix} \begin{pmatrix} \hat{X}_1 \\ \hat{X}_2 \\ \hat{X}_3 \\ \vdots \end{pmatrix} = (\theta_1 + \phi_1) \begin{pmatrix} 1 \\ X_1 \\ X_2 \\ \vdots \end{pmatrix}$$

Formulation

ARIMA$(1, 0, 1)$

$$\hat{X}_2 = \phi_1 X_1 + \theta_1 (X_1 - \hat{X}_1)$$
$$\hat{X}_3 = \phi_1 X_2 + \theta_1 (X_2 - \hat{X}_2)$$
$$\hat{X}_4 = \phi_1 X_3 + \theta_1 (X_3 - \hat{X}_3) \text{.....and so on}$$

For given θ_1 and ϕ_1, we need to solve

$$\boxed{A(\theta_1, \phi_1)\hat{X}_t = b(\theta_1, \phi_1)}$$

Formulation

ARIMA$(\mathbf{1},\mathbf{0},\mathbf{1})$

$$\hat{X}_2 = \phi_1 X_1 + \theta_1 (X_1 - \hat{X}_1)$$
$$\hat{X}_3 = \phi_1 X_2 + \theta_1 (X_2 - \hat{X}_2)$$
$$\hat{X}_4 = \phi_1 X_3 + \theta_1 (X_3 - \hat{X}_3)\text{.....and so on}$$

For given θ_1 and ϕ_1, we need to solve

$$\boxed{A(\theta_1, \phi_1)\hat{X}_t = b(\theta_1, \phi_1)}$$

How do we pick correct θ_1 and ϕ_1 for given X_t?

$$\min_{\theta_1, \phi_1} ||X_t - \hat{X}_t(\theta_1, \phi_1)||_2$$

Formulation

ARIMA$(\mathbf{p}, \mathbf{0}, \mathbf{1})$

$$\hat{X}_p = \phi_1 X_{p-1} + \phi_2 X_{p-2} + +\phi_3 X_{p-3} + \cdots + \theta_1 (X_{p-1} - \hat{X}_{p-1})$$

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \theta_1 & 1 & 0 & \cdots & 0 \\ 0 & \theta_1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & \cdots & \theta_1 & 1 \end{pmatrix} \begin{pmatrix} \hat{X}_1 \\ \hat{X}_2 \\ \hat{X}_3 \\ \vdots \end{pmatrix} = (\theta_1 + \phi_1) \begin{pmatrix} 1 \\ X_1 \\ X_2 \\ \vdots \end{pmatrix} + \phi_2 \begin{pmatrix} 1 \\ 1 \\ X_1 \\ \vdots \end{pmatrix} + \cdots$$

Formulation

Summary

$$\begin{aligned} &\min_{\theta,\phi} f(\theta,\phi) \\ \text{subject to} \quad & A(\theta,\phi)\hat{X}_t = b(\theta,\phi) \end{aligned}$$

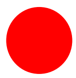
where $f(\theta,\phi) = ||X_t - \hat{X}_t(\theta,\phi)||_2$

- Optimization methods to solve the problem: Nelder-Mead, CG, BFGS, L-BFGS.

Optimization 1: Nelder-Mead

Downhill simplex method

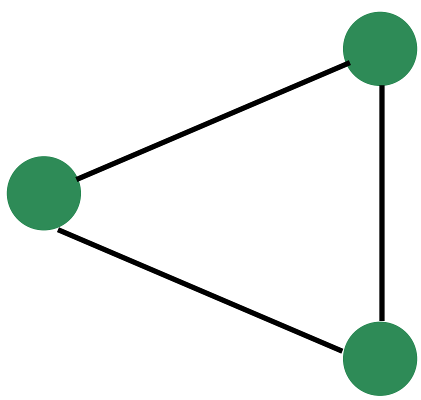
$$\boxed{\textit{Initial point} \in \mathbb{R}^n}$$



Optimization 1: Nelder-Mead

Downhill simplex method

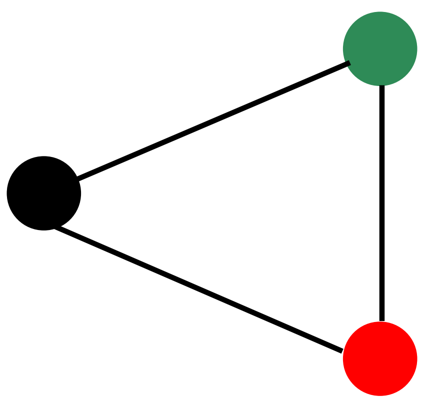
$$\boxed{\textit{Simplex } n+1 \text{ points}}$$



Optimization 1: Nelder-Mead

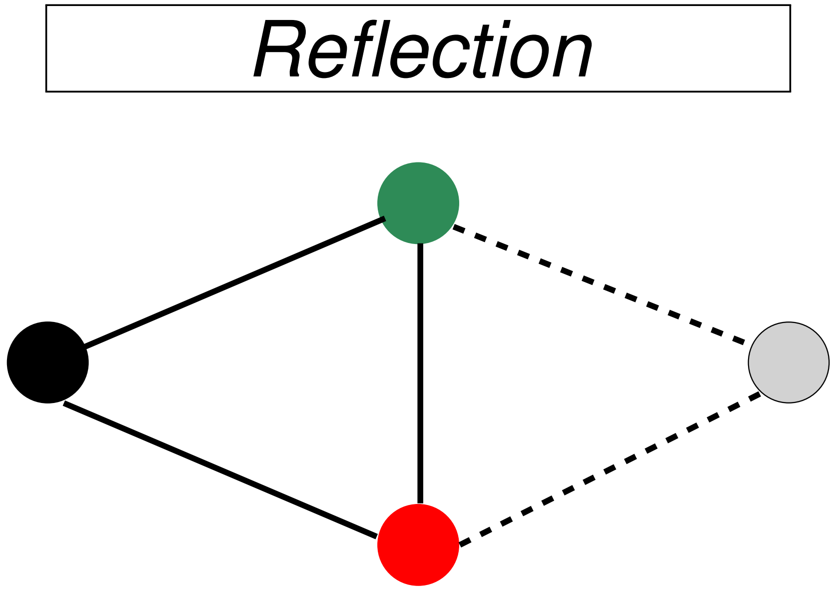
Downhill simplex method

$$\boxed{\textit{Simplex}\ n+1 \text{ points}}$$



Optimization 1: Nelder-Mead

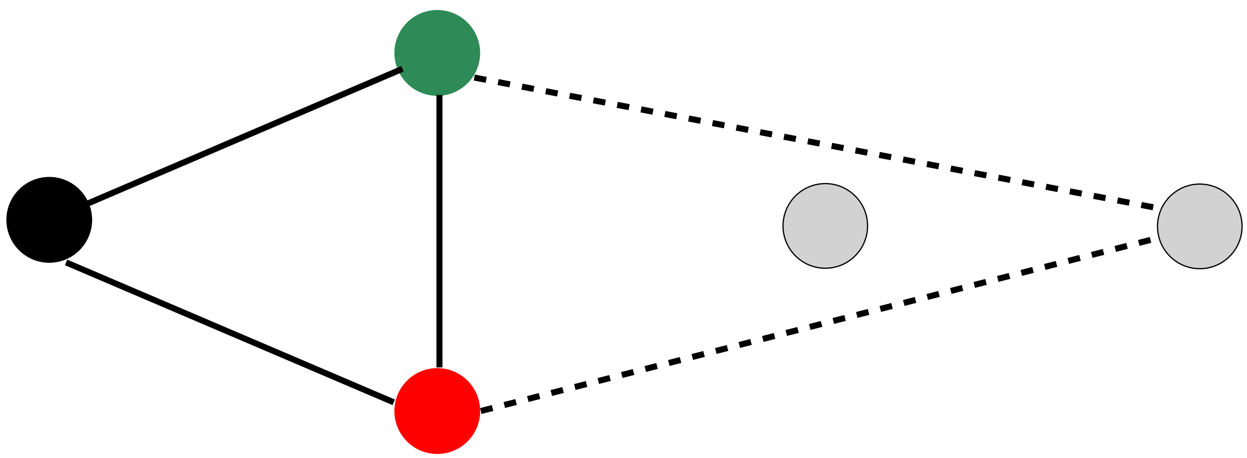
Downhill simplex method



Optimization 1: Nelder-Mead

Downhill simplex method

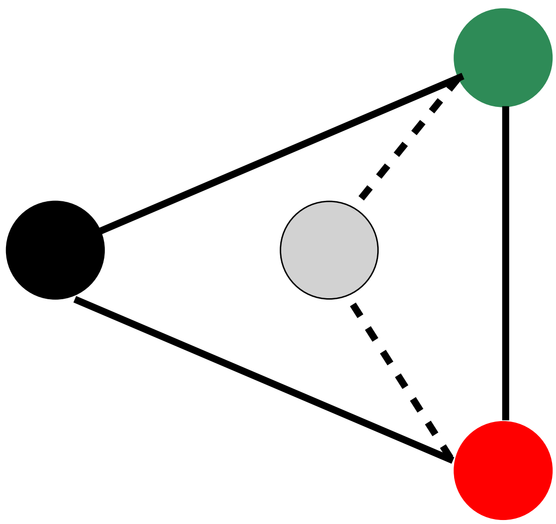
Expand



Optimization 1: Nelder-Mead

Downhill simplex method

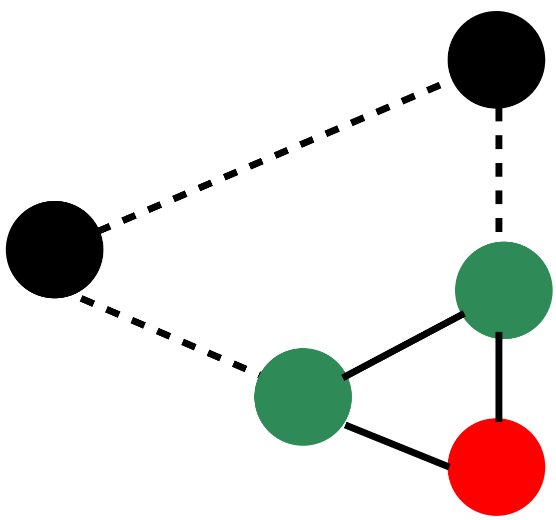
Contraction



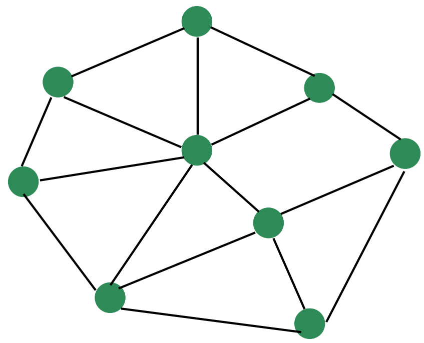
Optimization 1: Nelder-Mead

Downhill simplex method

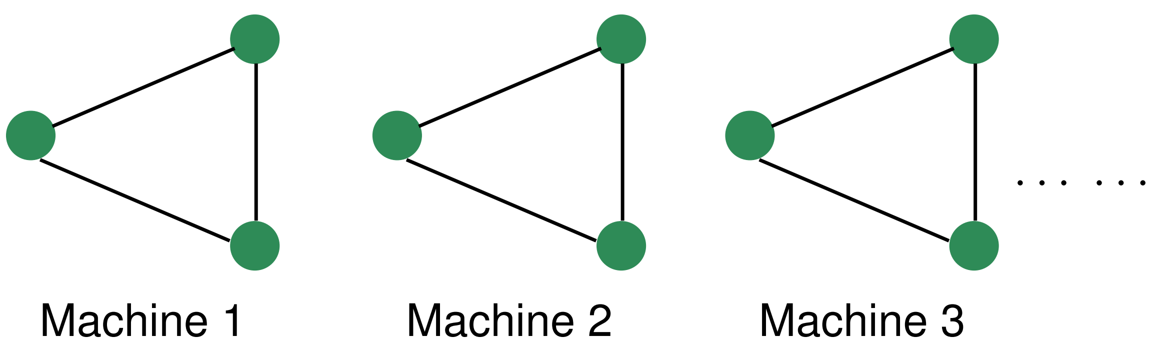
Shrink



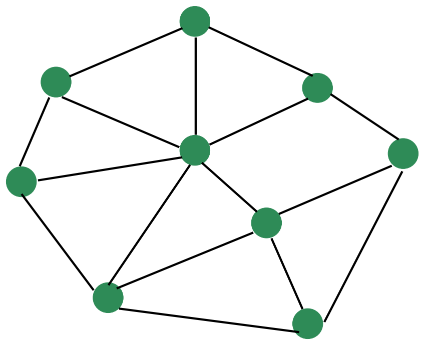
Optimization 1: Distributed Nelder-Mead



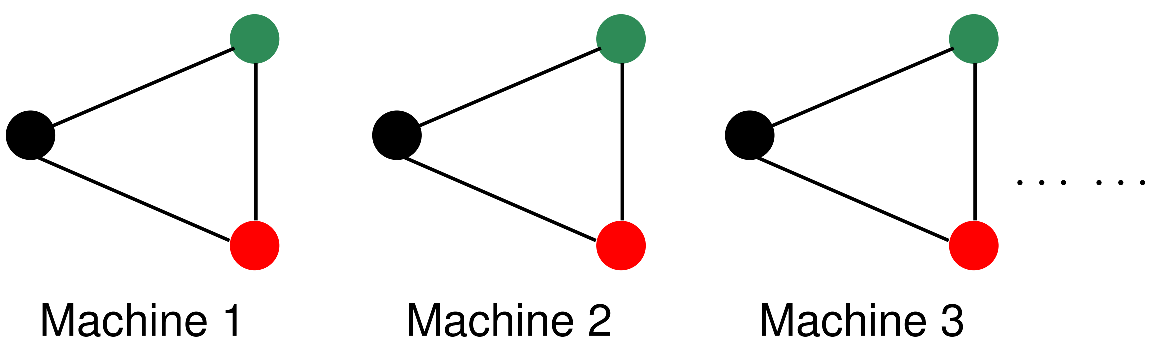
SystemML: Distribute simplex on multiple machines.



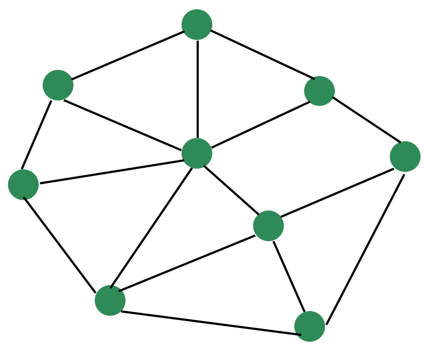
Optimization 1: Distributed Nelder-Mead



SystemML: Distribute simplex on multiple machines.



Optimization 1: Distributed Nelder-Mead



SystemML: Distribute simplex on multiple machines.

- Collect “local best” at each machine.
- Identify “global best”
- Communicate “global best” back to each machine
- Proceed with Nelder-Mead steps i.e.,*reflection,expansion,contraction*
- If no update recorded anywhere, *shrink* simplex on each machine

Optimization 2: Conjugate Gradient

First-order optimization method

Generate a sequence of points $(\theta^0, \phi^0), (\theta^1, \phi^1), (\theta^2, \phi^2), \cdots$ such that

$$f(\theta^0, \phi^0) \geq f(\theta^1, \phi^1) \geq f(\theta^2, \phi^2), \cdots$$

One possible way to obtain such a sequence is – move along a descent direction of the function $\Delta(\theta, \phi)$

$$(\theta^{k+1}, \phi^{k+1}) := (\theta^k, \phi^k) + t_k \Delta(\theta^k, \phi^k)$$

Optimization 2: Conjugate Gradient

Algorithm

given initial point (θ, ϕ).
repeat

- Determine descent direction $\Delta(\theta, \phi)$
- Choose a step size $t > 0$.
- *Update* $(\theta, \phi) := (\theta, \phi) + t\Delta(\theta, \phi)$.

until stopping criterion is satisfied.

Optimization 2: Conjugate Gradient

Algorithm

given initial point (θ, ϕ).
repeat

- Set $\Delta(\theta, \phi) = -\nabla f$.
  Use finite difference method to compute ∇f.
- Choose a step size $t > 0$.
  Backtracking line serach
  $t := 1, \alpha \in (0, 0.5), \beta \in (0, 1)$
  while
  $f((\theta, \phi) + t\Delta(\theta, \phi)) > f(\theta, \phi) + \alpha t \nabla f^\top \Delta(\theta, \phi), \quad t := \beta t$
- *Update* $(\theta, \phi) := (\theta, \phi) + t\Delta(\theta, \phi)$.

until $||\nabla f||_2 \leq \eta$

Optimization 3

Newton method

Use second order Taylor expansion to obtain the descent direction

$$\Delta(\theta, \phi) = -[Hf]^{-1}\nabla f$$

Hf is the Hessian matrix.

Optimization 3

Newton method

Use second order Taylor expansion to obtain the descent direction

$$\Delta(\theta, \phi) = -[Hf]^{-1}\nabla f$$

Hf is the Hessian matrix. **Prohibitively Expensive**

Optimization 3:**B**royden–**F**letcher–**G**oldfarb–**S**hanno

Quasi-Newton method

- Approximate Hessian matrix at each iteration by constructing a rank-two update matrix using ∇f.
- Efficiently use Sherman–Morrison formula to obtain inverse of the aprroximate Hessian.

Note: approximate Hessian matrix is denoted by B.

Optimization 3:**B**royden–**F**letcher–**G**oldfarb–**S**hanno

Algorithm

given initial point (θ_0, ϕ_0) and $B_0 = I$.
repeat

- Determine descent direction $\Delta(\theta_k, \phi_k) = -B_k^{-1}\nabla f$
- Choose a step size $t_k > 0$
  Update $(\theta_{k+1}, \phi_{k+1}) = (\theta_k, \phi_k) + t_k\Delta(\theta_k, \phi_k)$
- Set $s_k = t_k\Delta(\theta_k, \phi_k)$
- Calculate $y_k = \nabla f(\theta_{k+1}, \phi_{k+1}) - \nabla f(\theta_k, \phi_k)$
- $B_{k+1}^{-1} = B_k^{-1} + \frac{(s_k^\top y_k + y_k^\top B_k^{-1} y_k)(s_k s_k^\top)}{(s_k^\top y_k)^2} - \frac{B_k^{-1} y_k s_k^\top + s_k y_k^\top B_k^{-1}}{s_k^\top y_k}$

until stopping criterion is satisfied

Optimization: How do we choose??

Nelder-Mead

Pros: Simple, no derivative required, good for non-convex problems.
Cons: Local search method – can easily get stuck in local minimum, too many function evaluations required, very slow.

Conjugate Gradient

Pros: Simple.
Cons: Too slow near minimum, ill-defined for non-differentiable functions.

BFGS

Pros: Works in most cases, fast.
Cons: ill-defined for non-differentiable functions, computationally expensive (try L-BFGS), do not necessarily converge.

Revisit – Formulation

Summary

$$\min_{\theta,\phi} f(\theta, \phi)$$

$$\text{subject to} \quad A(\theta, \phi)\hat{X}_t = b$$

where $f(\theta, \phi) = ||X_t - \hat{X}_t(\theta, \phi)||_2$

Note: $A \in \mathbb{R}^{T \times T}$ and T is large for large time-series.

Solve non–symmetric sparse system of linear equations efficiently?

Solver 1: Jacobi method

Key idea: $A = D + R$

$$\begin{aligned} (D+R)\hat{X} &= b \\ D\hat{X} &= b - R\hat{X} \end{aligned}$$

Here D is a diagonal matrix with $D_{ii} = A_{ii}$. R constitute off–diagonal entries of A.

Iterate until convergence: $\hat{X}^{k+1} = D^{-1}b - D^{-1}R\hat{X}^k$

Solver 1: Jacobi method

Key idea: $A = D + R$

$$\begin{aligned} (D+R)\hat{X} &= b \\ D\hat{X} &= b - R\hat{X} \end{aligned}$$

Here D is a diagonal matrix with $D_{ii} = A_{ii}$. R constitute off–diagonal entries of A.

Iterate until convergence: $\hat{X}^{k+1} = D^{-1}b - D^{-1}R\hat{X}^k$

For problem at hand,

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \theta_1 & 1 & 0 & \cdots & 0 \\ 0 & \theta_1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & \cdots & \theta_1 & 1 \end{pmatrix} \begin{pmatrix} \hat{X}_1 \\ \hat{X}_2 \\ \hat{X}_3 \\ \vdots \end{pmatrix} = (\theta_1 + \phi_1) \begin{pmatrix} 1 \\ X_1 \\ X_2 \\ \vdots \end{pmatrix}$$

Solver 1: Jacobi method

Key idea: $A = D + R$

$$\begin{aligned}(D+R)\hat{X} &= b \\ D\hat{X} &= b - R\hat{X}\end{aligned}$$

Here D is a diagonal matrix with $D_{ii} = A_{ii}$. R constitute off–diagonal entries of A.

Iterate until convergence: $\hat{X}^{k+1} = D^{-1}b - D^{-1}R\hat{X}^k$

For problem at hand,

$$D = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}; \quad R = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ \theta_1 & 0 & 0 & \cdots & 0 \\ 0 & \theta_1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & 0 & \cdots & \theta_1 & 0 \end{pmatrix}$$

Solver 1: Jacobi method

Key idea: $A = D + R$

$$\begin{aligned}(D+R)\hat{X} &= b \\ D\hat{X} &= b - R\hat{X}\end{aligned}$$

Here D is a diagonal matrix with $D_{ii} = A_{ii}$. R constitute off–diagonal entries of A.

Iterate until convergence: $\hat{X}^{k+1} = D^{-1}b - D^{-1}R\hat{X}^{k}$

This method is guaranteed to converge given **diagonal dominance** of A

Solver 2: **G**eneralized **m**inimal **r**esidual **m**ethod

Key idea: *Krylov subspace*

k-th *Krylov subspace* of $A\hat{X} = b$
$$\mathcal{K}_k = span(b, Ab, A^2b, \cdots, A^{k-1}b)$$

- GMRES approximate the exact solution of $A\hat{X} = b$ by $\hat{X}^k \in \mathcal{K}_k$
- $\hat{X}^k \in \mathcal{K}_k$ is the vector that minimizes residual

$$r_k = b - A\hat{X}^k$$

Solver 2: **G**eneralized **m**inimal **r**esidual **m**ethod

Algorithm 1 Arnoldi iteration for Q_k

Require: A, b
  Compute $q_1 = \frac{b}{||b||_2}$
  for $i = 1$ to k **do**
    $v = Aq_i$
    for $j = 1$ to i **do**
      $S(j,i) = v^\top q_j$
      $v = v - S(j,i) * q_j$
    end for
    $q_{i+1} = \frac{v}{||v||_2}$
    $S(i+1,i) = ||v||_2$
  end for
Ensure: Orthonormal basis Q_k and upper *Hessenberg* matrix S_k

Solver 2: **G**eneralized **m**inimal **r**esidual **m**ethod

Solution for $A\hat{X} = b$

$$
\begin{aligned}
e_1 &= (1, 0, 0, \cdots, 0) \\
e_1 &= ||b||_2 e_1 \\
y_k &= S_k(1:k, 1:k) \backslash e_1 && \text{We used exact solver here} \\
\hat{X}^k &= Q_k y_k && \text{Best solution in } \mathcal{K}_k
\end{aligned}
$$

Our experimental results used **GMRES**

Preliminary results

We present preliminary results with

- Optimization routine – BFGS
- Solver – GMRES (K_{10})
- Line search – backtracking $(\alpha = 0.0001, \beta = 0.9)$

Preliminary results: ARIMA $(1, 0, 1)$

Time step–1 hour

X1011

	SystemML	R
ϕ_1	-0.7233	NaN
θ_1	0.6932	NaN

X1157

	SystemML	R
ϕ_1	0.8662	0.8662 $\pm$ 0.0189
θ_1	-0.3316	-0.3318 $\pm$ 0.0417

Preliminary results: ARIMA $(1, 0, 1)$

Time step–1 hour

X1158

	SystemML	R
ϕ_1	0.9053	0.9976 ± 0.0017
θ_1	-0.5076	-0.014 ± 0.0246

X1178

	SystemML	R
ϕ_1	0.9479	1.0001 ± 0.0006
θ_1	-0.8860	-0.9944 ± 0.0026

Preliminary results: ARIMA$(5, 0, 1)$

Time step–1 hour

X1157

	SystemML	R
ϕ_1	0.8920	0.892 $\pm$ 0.0759
ϕ_2	-0.3264	-0.3265 $\pm$ 0.0562
ϕ_3	0.4233	0.4233 $\pm$ 0.0313
ϕ_4	-0.3277	-0.3278 $\pm$ 0.0399
ϕ_5	0.2202	0.2203 $\pm$ 0.0242
θ_1	-0.2133	-0.2133 $\pm$ 0.0756

Preliminary results: More

X1011 Time step–15mins

ARIMA$(1, 0, 1)$

	SystemML	R
ϕ_1	0.2318	0.2315 $\pm$ 0.0701
θ_1	-0.0285	-0.028 $\pm$ 0.0726

ARIMA$(3, 0, 1)$

	SystemML	R
ϕ_1	-0.1549	-0.1381 $\pm$ 0.2211
ϕ_2	0.0691	0.0657 $\pm$ 0.0472
ϕ_3	0.0391	0.0394 $\pm$ 0.0122
θ_1	0.3591	0.3423 $\pm$ 0.2212

Summary

- Implemented Serial Nelder-Mead in DML.
- Implemented Distributed Nelder-Mead in DML.
- Implemented CG in DML.
- Implemented BFGS in DML.
- Implemented L-BFGS in DML.
- Implemented GMRES in DML.
- Made useful observations to improve SystemML.

Thank You!

Preliminary results: X1157

Time step–1 hour

ARIMA$(2, 0, 1)$

	SystemML	R
ϕ_1	0.0442	0.0184 ± 0.0459
ϕ_2	0.4410	0.4636 ± 0.0389
θ_1	0.7626	0.787 ± 0.0367

ARIMA$(3, 0, 1)$

	SystemML	R
ϕ_1	0.2588	0.2585 ± 0.0479
ϕ_2	0.1320	0.1323 ± 0.0398
ϕ_3	0.3413	0.3413 ± 0.0228
θ_1	0.4140	0.4143 ± 0.0481