UNIVERSITY OF MINNESOTA TWIN CITIES

Linear and Nonlinear Methods for Data Science Applications

Yousef Saad

University of Minnesota

Spring School - Univ. M6, Benguerir, Morocco

Feb 26-29, 2024

Focus of numerical linear algebra in past decades

1940s–1950s: Major issue: flutter problem in aerospace engineering → eigenvalue problem [cf. Olga Taussky Todd] → LR, QR, .. → 'EISPACK'

1960s: Problems related to the power grid promoted what we would call today general sparse matrix techniques

1970s– Automotive, Aerospace, ..: Computational Fluid Dynamics (CFD)

Late 1980s: Thrust on parallel matrix computations.

Late 1990s: Spur of interest in "financial computing"

Current: Machine Learning and AI

First wave of computing (CSE). Example: Fluid flow

Physical Model

↓

Nonlinear PDEs

↓

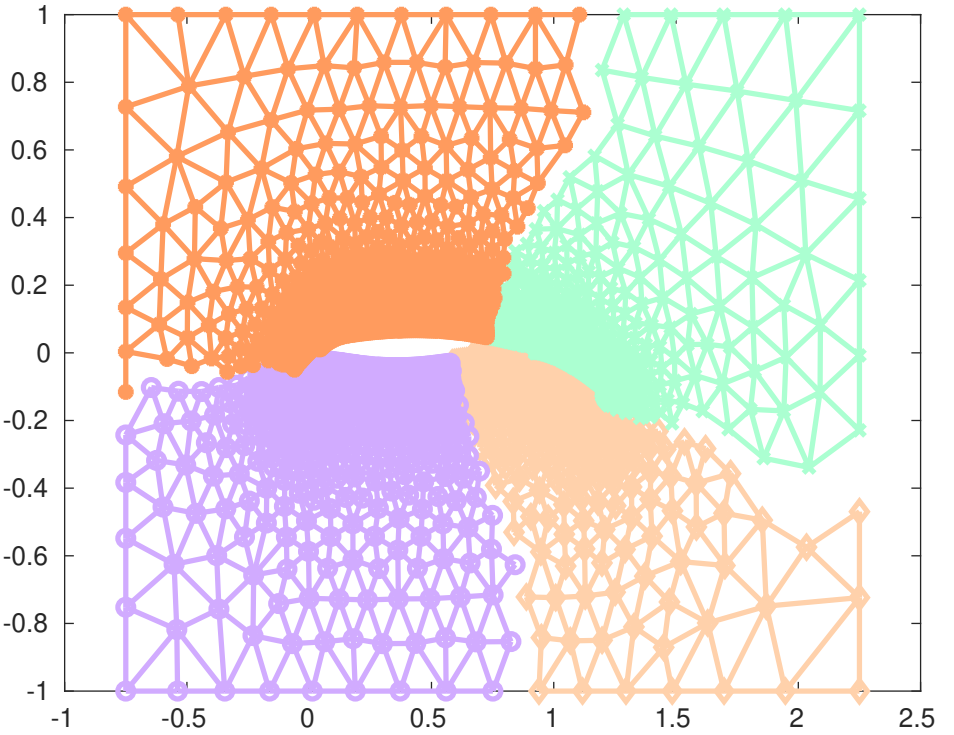
Discretization

↓

Linearization (Newton)

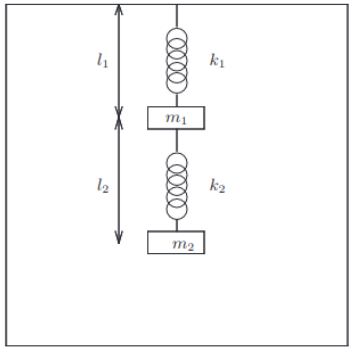
↓

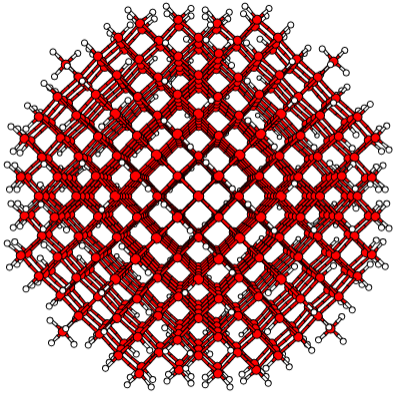
Sparse Linear Systems $Ax = b$

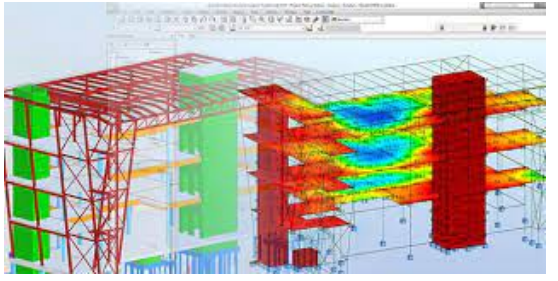


First Wave. Example: Eigenvalue Problems

➤ Many applications require the computation of a few eigenvalues + associated eigenvectors of a matrix A



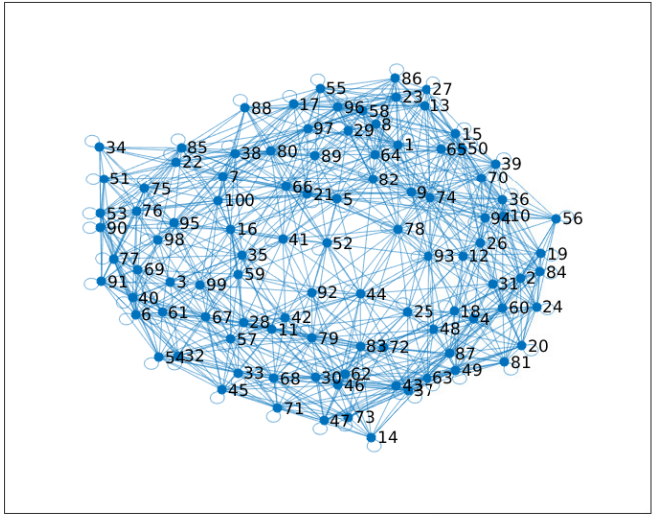




- Structural Engineering – (Goal: frequency response)
- Electronic structure calculations [Schrödinger equation..] – Quantum chemistry
- Stability analysis [e.g., electrical networks, mechanical system,..]
- ...

Second Wave (Data Mining). Example: Google Page Rank

Rank importance of nodes using random walks: visit web-pages following each link on a node at random with equal likelihood → Markov chain

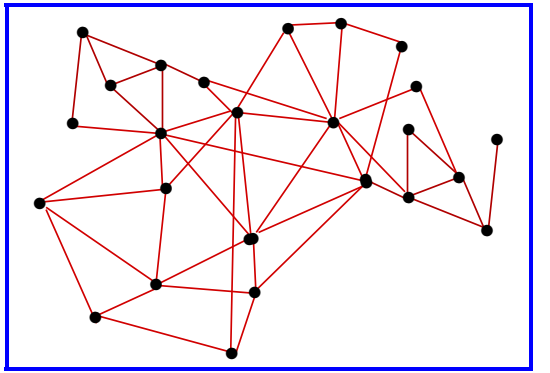


- Problem type: (homogeneous) Linear system. 'Eigenvector' problem.

- Many similar measures of 'centrality' of vertices in a graph -
- Applications: Network analysis, Social networks
- Observation: lots of insight from Physics ('Estrada index')

Second Wave. Example: Graph embedding

Problem: In order to work with a graph we need to represent each vertex as a vector in $\mathbb{R}^d$. How can we map a graph with n nodes into an array of $\mathbb{R}^{d\times n}$?



$\downarrow$

$\boldsymbol{Y} = [\boldsymbol{y_1}, \boldsymbol{y_2}, \cdots, \boldsymbol{y_n}]$ in $\mathbb{R}^{d\times n}$

Mapping: Graph $\rightarrow$ Data

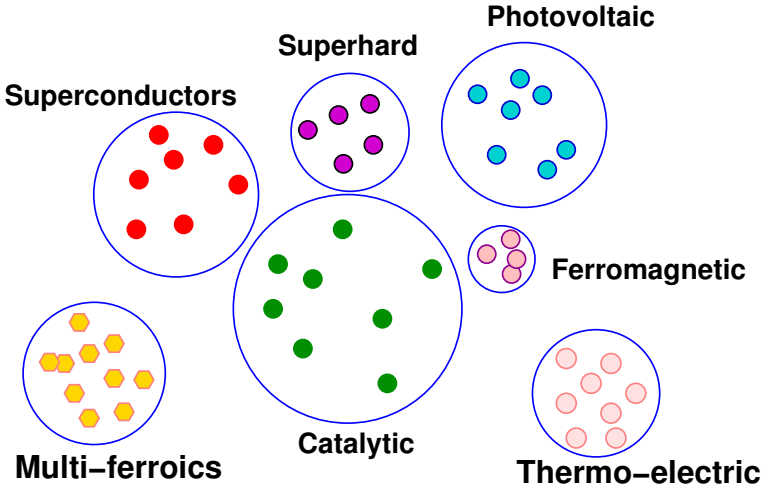
- Many applications, e.g., visualization of a graph/ network
- Embedding is a key ingredient of many methods in deep learning (Graph Neural Networks, Graph Convolutional Networks)
- Key ingredient in manifold learning [illustration coming shortly]

TOPICS IN CLASSICAL DATA MINING: CLUSTERING

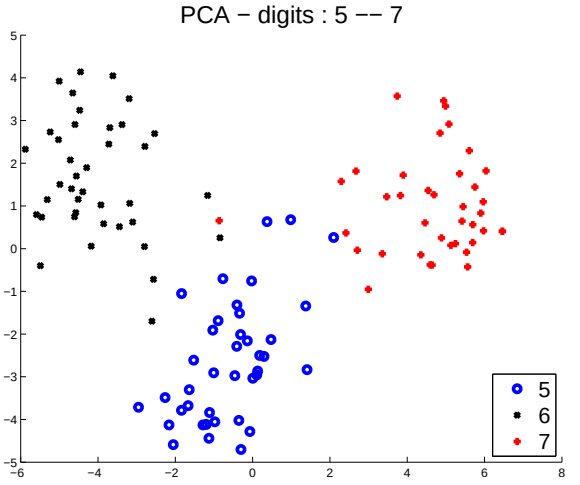
From graph partitioning to data clustering

• Problem: we are given n data items: $x_1, x_2, \cdots, x_n$. Would like to *'cluster'* them, i.e., group them so that each group or cluster contains items that are similar in some sense.

➤ Example: materials



➤ Example: Digits

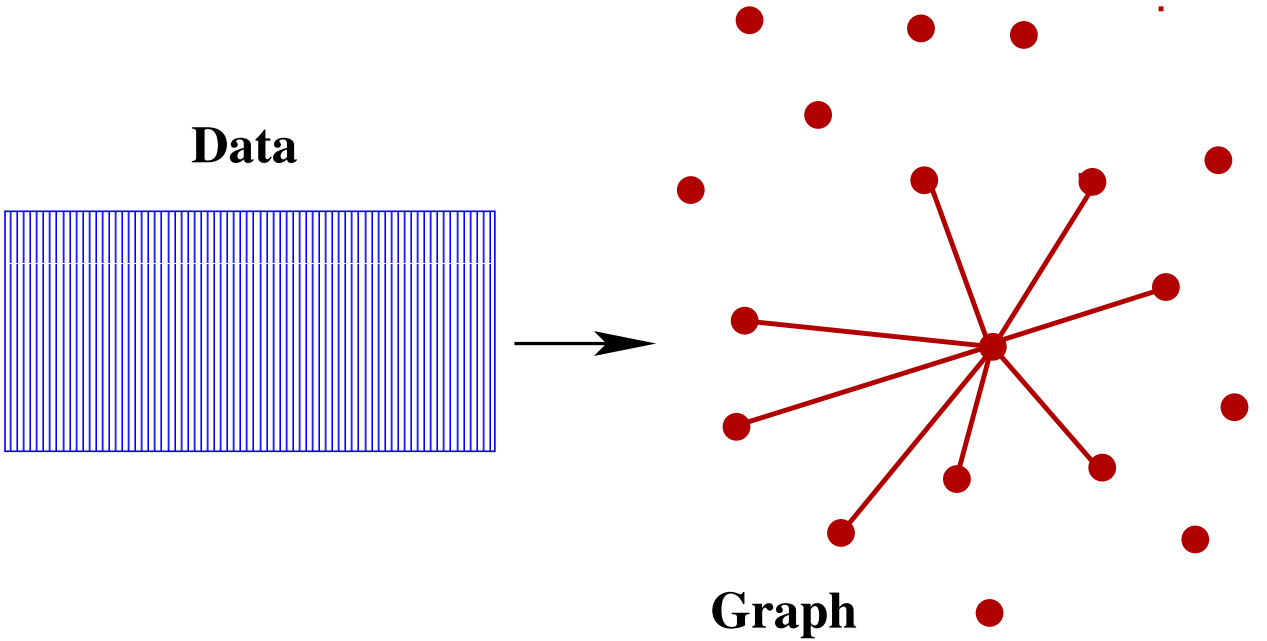


➤ Each group is a 'cluster' or a 'class'

➤ 'Unsupervised learning'

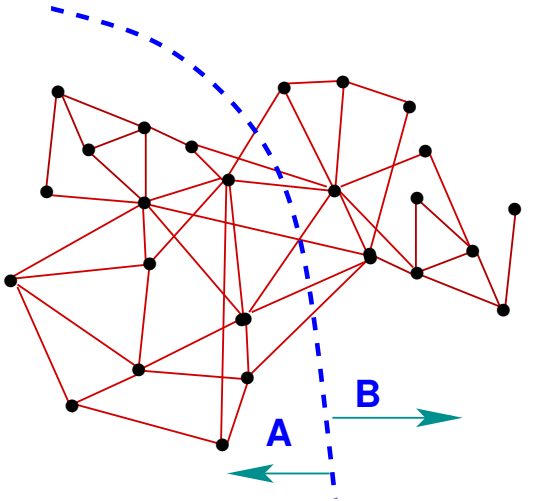
Methods based on similarity graphs: Nearest neighbor graphs

- For each node, get a few of the nearest neighbors $\rightarrow$ Graph



- Problem: How to build a nearest-neighbor graph from given data?
- Divide and conquer approaches: e.g., H-r Fang, J. Chen, YS '08.

Edge cuts, ratio cuts, normalized cuts, ...



- Assume now that we have a 'similarity graph'
- Want: partition vertex set into A and B

$$A \cup B = V, \quad A \cap B = \emptyset$$

- What criterion?

- Define:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v)$$

- Naive idea 1: minimize $cut(A, B)$. Problem: imbalance
- Naive idea 2: graph partitioning: $|A| = |B|$. Problem: not meaningful

- Many alternative objective functions defined in litterature
- Ratio cuts: minimize $cut(A, B)/(|A||B|)$ (Hagen-Kahng, '91)
- Normalized Cuts: (Shi-Malik, '00) minimize:

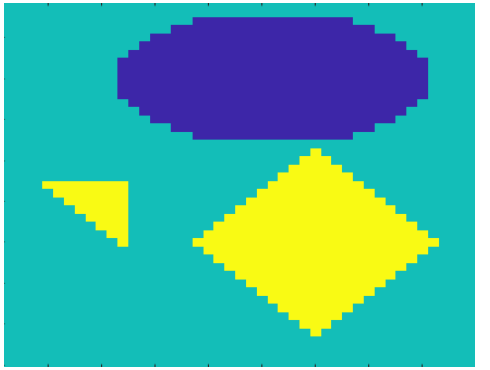
$$cut(A, B)/w(A, V) + cut(A, B)/w(B, V)$$

- ....
- If $L = D - W$ = graph Laplacian, then :
- All methods lead to eigenvalue problem like: $Lx = \lambda Dx$,

Example of application: Image segmentation

➤ Image segmentation = technique for separating parts of a given image

➤ Example: remove background, extract a face in photo, etc..



➤ First task: obtain a graph from pixels: Common to use "Heat kernels":

$$w_{ij} = e^{\frac{-\|F_i - F_j\|^2}{\sigma_I^2}} \times \begin{cases} e^{\frac{-\|X_i - X_j\|^2}{\sigma_X^2}} & \text{if} \|\boldsymbol{X}_i - \boldsymbol{X}_j\| < r \\ 0 & \text{else} \end{cases}$$

- Where $\boldsymbol{F}_j =$ feature value (e.g., brightness), and Let $\boldsymbol{X}_j =$ spatial position.

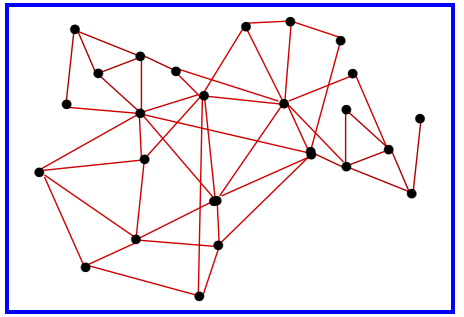
➤ Sparsity depends on parameters

TOPICS IN CLASSICAL DATA MINING: EMBEDDING

Graph embeddings

- We have seen how to build a graph to represent data
- *Graph embedding* does the opposite: maps a graph to data

Vertex embedding: map every vertex x_i to a vector $y_i \in \mathbb{R}^d$



$\longrightarrow$

Data: $\boldsymbol{Y} = [y_1, y_2, \cdots, y_n]$ *in* $\mathbb{R}^{d \times n}$

Want: preserve *similarities* in graph.

- Many methods do this: Eigenmaps and LLE are two of the best known
- Eigenmaps uses the *graph Laplacean* $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$

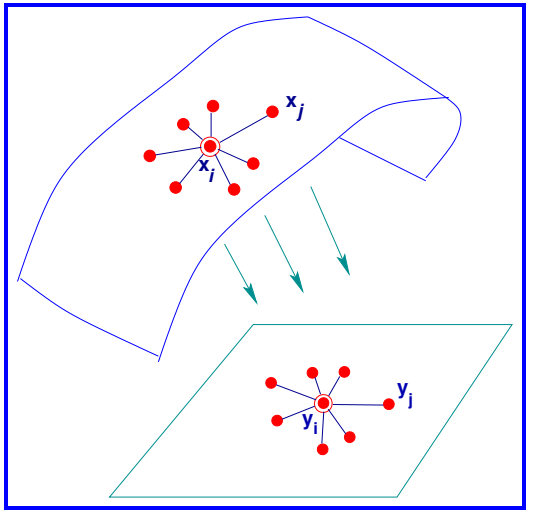
Example: The Laplacean eigenmaps approach

Laplacean Eigenmaps [Belkin-Niyogi '01] *minimizes*

$$\mathcal{F}(Y) = \sum_{i,j=1}^{n} w_{ij} \|y_i - y_j\|^2 \quad \text{subject to} \quad YDY^\top = I$$

Motivation: if $\|x_i - x_j\|$ is small (orig. data), we want $\|y_i - y_j\|$ to be also small (low-Dim. data)

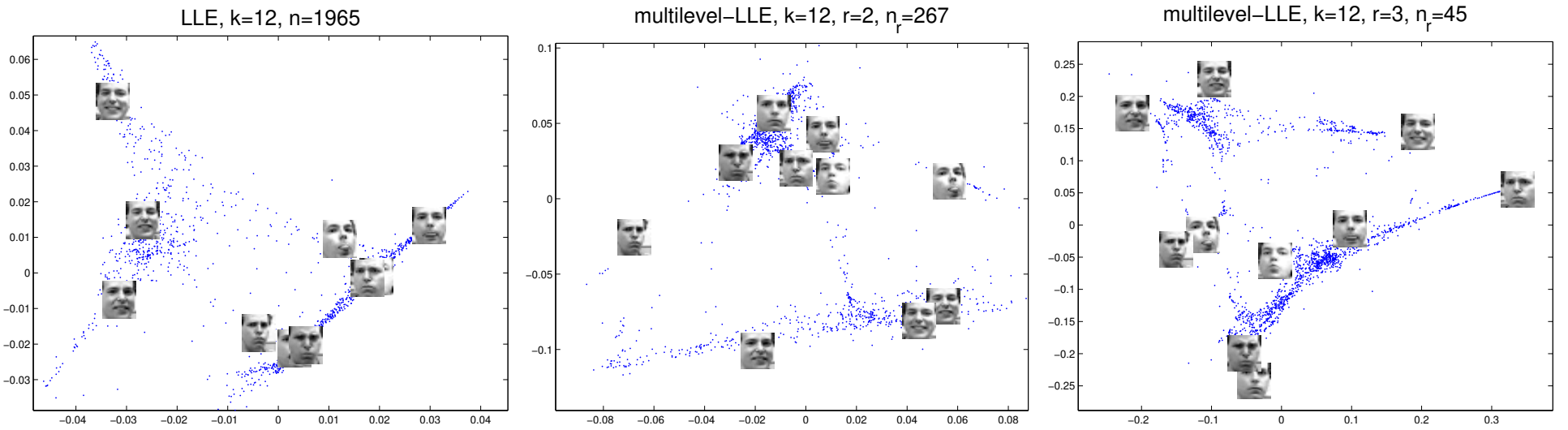
- Original data used indirectly through its graph
- Objective function translated to a trace ratio
- Yields a sparse eigenvalue problem



"Manifold Learning" Example: projection of face images

- Frey Dataset: 1,965 images of an individual – different expressions. Each image: 20×28 grey-scale pixels

Various projections [see H-R Fang, S. Sakellaridi, YS '10]



2D mappings of `Frey Face` *database using LLE and multilevel-LLE.*

More recent methods

- Quite a bit of recent work - e.g., methods: node2vec, DeepWalk, GraRep, ... See the following papers ... among many others :

[1] *William L. Hamilton, Rex Ying, and Jure Leskovec* *Representation Learning on Graphs: Methods and Applications* arXiv:1709.05584v3 (2017)

[2] *Shaosheng Cao, Wei Lu, and Qiongkai Xu* *GraRep: Learning Graph Representations with Global Structural Information*, CIKM, ACM Conference on Information and Knowledge Management, 24, (2015)

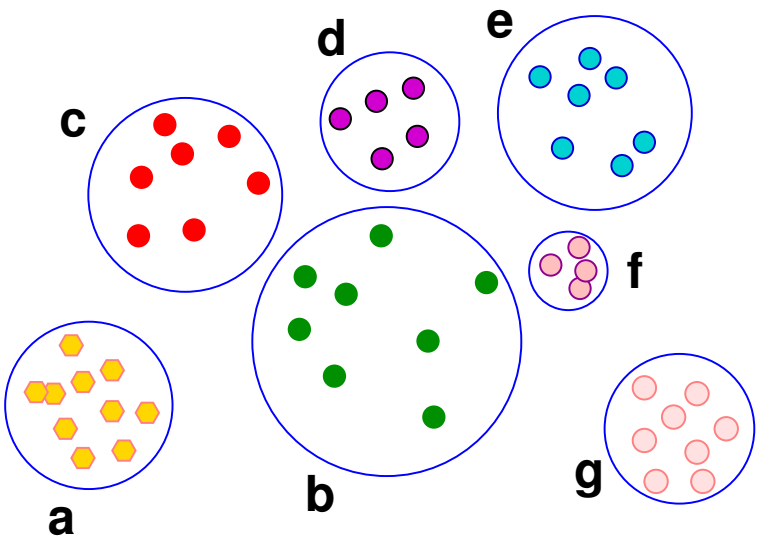
[3] *Amr Ahmed, Nino Shervashidze, and Shravan Narayanamurthy*, *Distributed Large-scale Natural Graph Factorization* [Proc. WWW 2013, May 13–17, 2013, Rio de Janeiro, Brazil]

TOPICS IN CLASSICAL DATA MINING: CLASSIFICATION

Supervised learning

- We now have data that is 'labeled' - eg. Message is *'Spam'* or *'Not-Spam'*

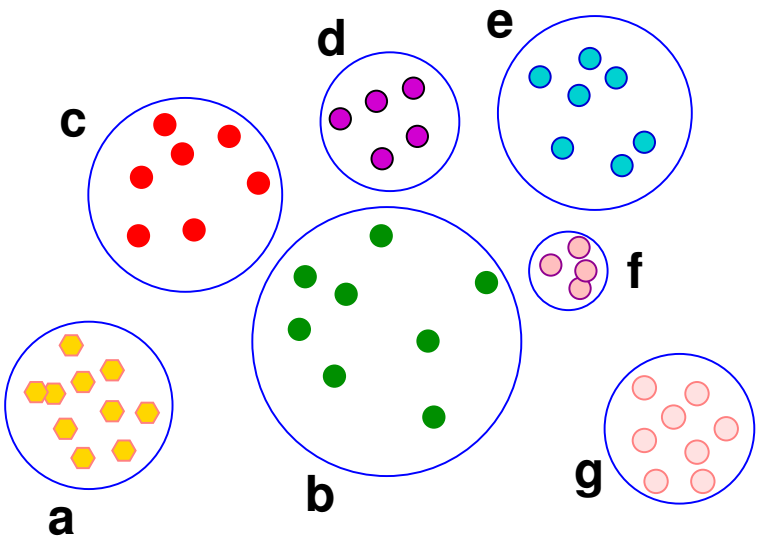
Examples: Health Sciences ('malignant'- 'non malignant') ; Materials ('photovoltaic', 'hard', 'conductor', ...) ; Digit Recognition ('0', '1', ..., '9')

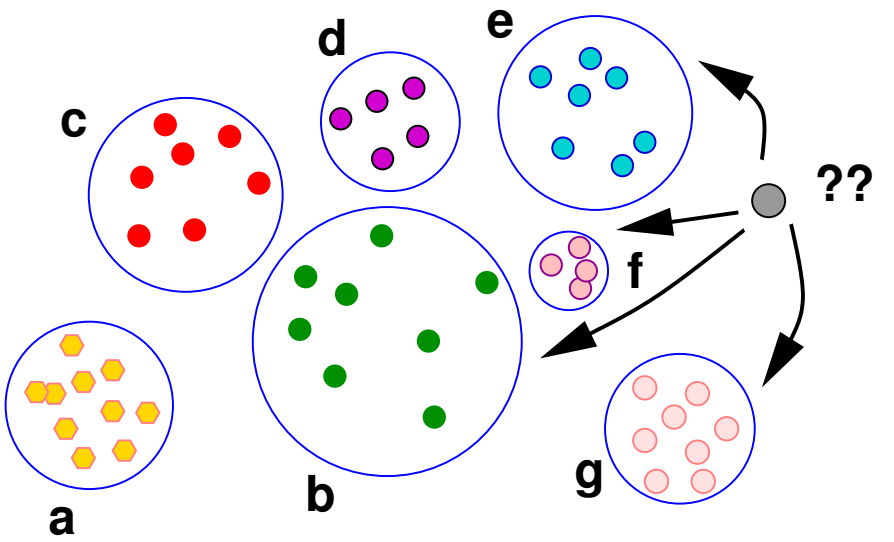


Supervised learning

➤ We now have data that is 'labeled'

Examples: Health Sciences ('malignant'- 'non malignant') ; Materials ('photovoltaic', 'hard', 'conductor', ...) ; Digit Recognition ('0', '1', ..., '9')

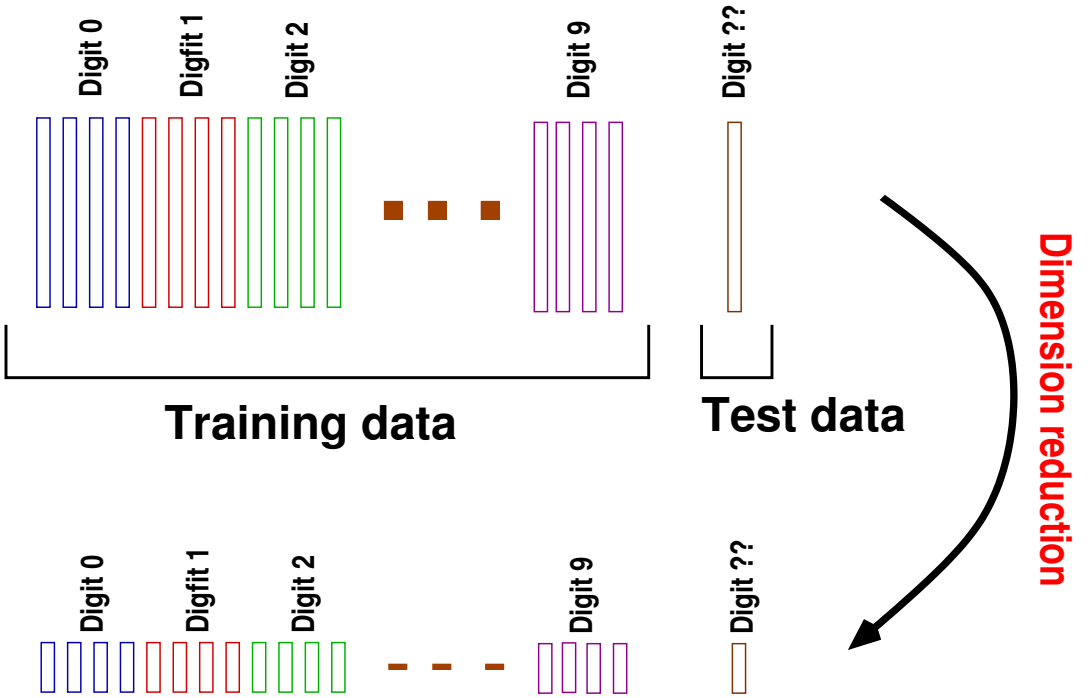




Supervised learning: classification

➤ Best illustration: written digits recognition example

Given: set of labeled samples (training set), and an (unlabeled) test image x.
Problem: label of x =?



➤ Roughly speaking: we seek dimension reduction so that recognition is 'more effective' in low-dim. space

In Brief: Support Vector Machines (SVM) - [Vapnik,'92]

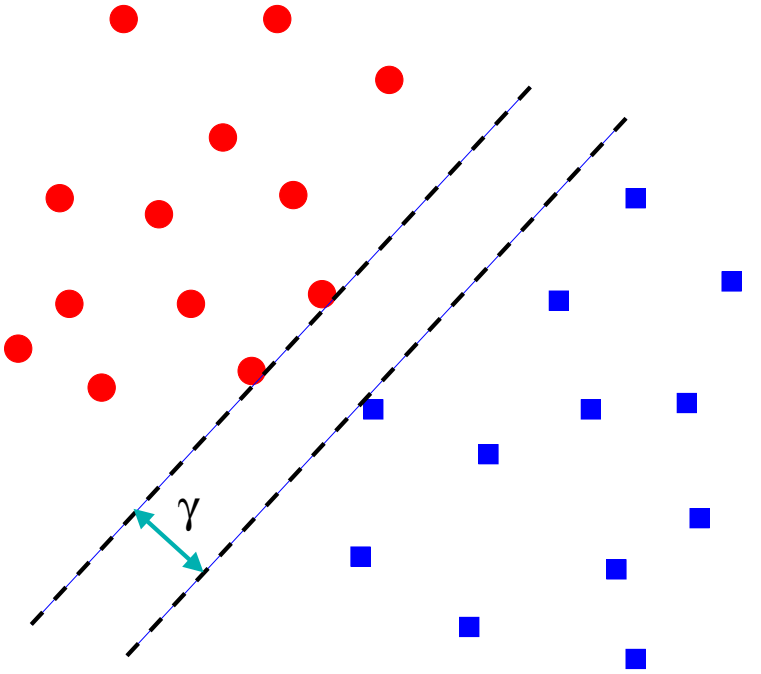
➤ Formally, SVM finds a hyperplane that best separates two training sets belonging to two classes.

Hyperplane: $\boxed{w^T x + b = 0}$ $\rightarrow$ Classifier: $f(x) = sign(w^T x + b)$

➤ Normalize parameters w, b so that: hyperplane $w^T x + b \geq 1 \rightarrow$ set 1, $w^T x + b \leq -1 \rightarrow$ set 2.

➤ With $y_i = +1$ for one class and $y_i = -1$ for the other, we can write the constraints as $y_i(w^T x_i + b) \geq 1$.

- The margin == max. distance between the two planes on 'decision boundary'
- Goal: find w, b to maximize margin
- Maximize margin subject to the constraint $y_i(w^T x_i + b) \geq 1$.
- It turns out that: margin == $\gamma = \frac{2}{\|w\|_2}$



- Need to solve the constrained quadratic programming problem:

$$\begin{aligned} \min_{w.b} \quad & \frac{1}{2}\|w\|_2^2 \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq 1, \quad \forall x_i. \end{aligned}$$

Modifications: 1) Soft margin; 2) Use kernel; ....

Deep Neural Networks (DNNs)

➤ Training a neural network can be viewed as a problem of approximating a function ϕ which is defined via sets of parameters:

Input: x, **Output:** y

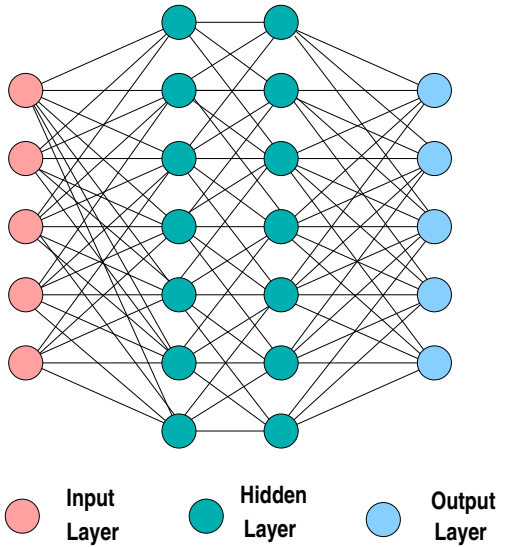
Set: $z_0 = x$

For $l = 1 : \mathtt{L+1}$ **Do:**

$$z_l = \sigma(W_l^T z_{l-1} + b_l)$$

End

Set: $y = \phi(x) := z_{L+1}$



- layer # 0 = input layer
- layer # $(L+1)$ = output layer
- Matrix W_l associated with layers 1,2, $L+1$.

➤ Problem:

Find ϕ (i.e., matrices W_l) s.t. $\phi(x) \approx y$

DNN (continued)

- Problem is not convex + it is highly parameterized $\rightarrow$ hard to solve
- Basic method used: Stochastic gradient descent + momemtum variants
- Training is very expensive – GPUs help
- Starting in 2017, huge development in Large Language Models (LLMs)
- BERT ('18), GPT-1 ('18), GPT2('19), GPT3 ('20), ... $\rightarrow$
- ... *ChatGPT (3.5, 4)*, *Copilot*, *PaLM/Gemini* (Google), *BLOOM* ...
- This brings us to ...

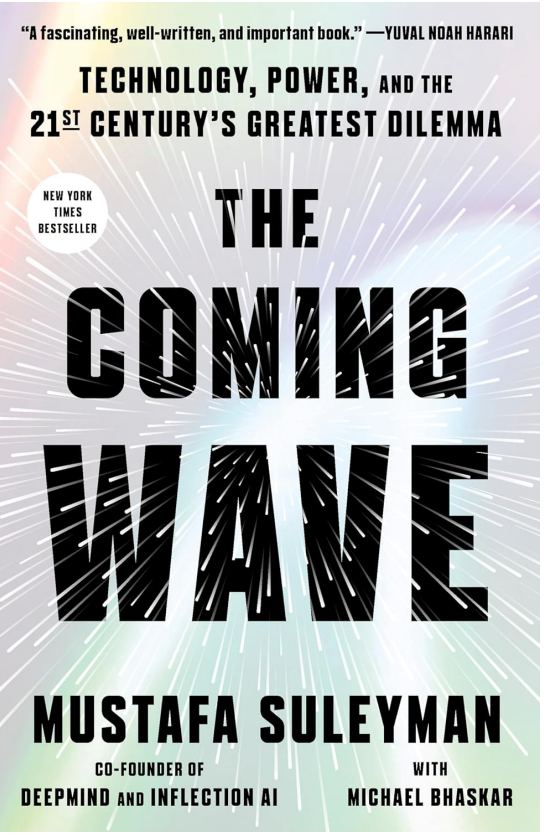
THE ML/AI WAVE

New BIG wave: AI

- Recommended reading (non-technical) →
- Author: co-founder of Deep-Mind

In summary:

- AI is making astounding progress ...
- ... with huge acceleration in past ≈ 6-7 years
- Synergetic forces: Hardware, openness, focus, ...
- Biggest issue now: Containment



A new Moore's law? Compare with progress in microchips

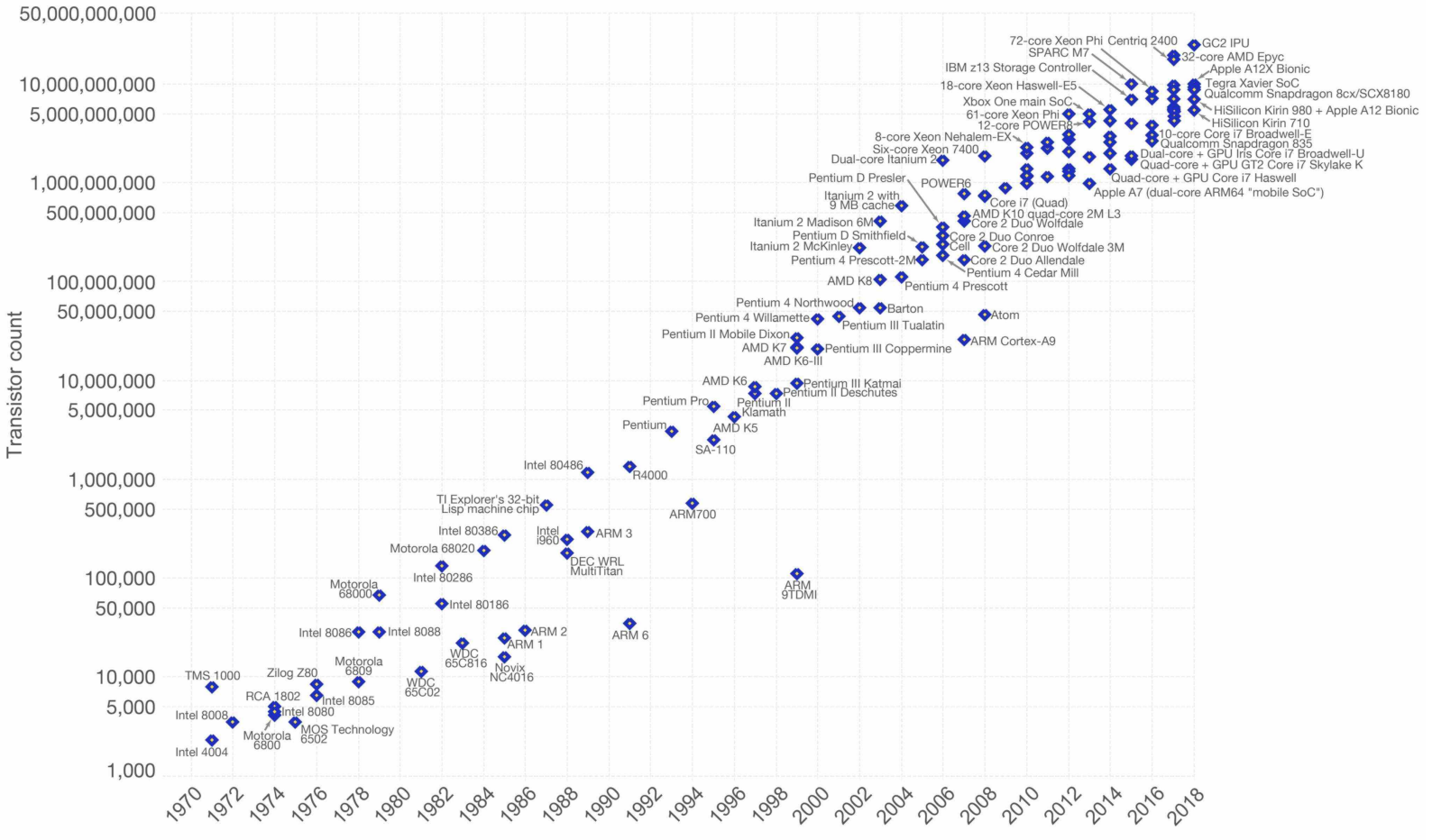
- Useful to compare with progress made in hardware
- Moore's law has been incredibly accurate in predicting advances in microchips. Note: stipulated in 1965! [actual ratio corrected in 1975.. still....]

> Moore's Law: Number of transistors placed on a chip doubles every 2 years

- One can see similar laws in progress of technology ... including AI.
- Technology progresses in exponential bursts

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

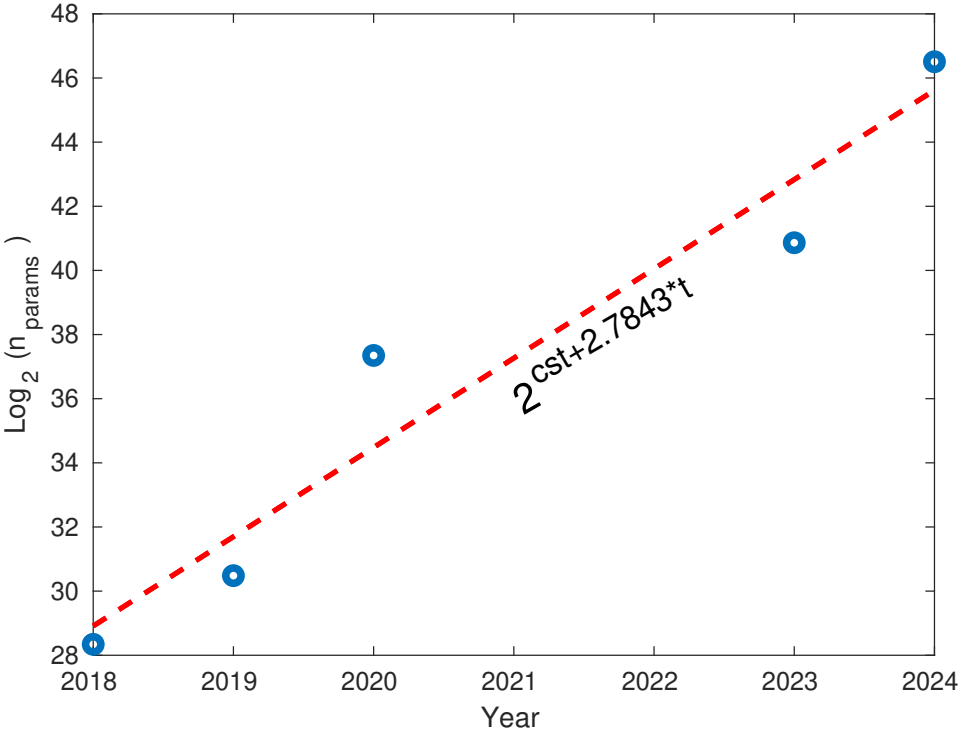


Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

*Thanks: Max Roser, CC BY-SA 4.0,Wikimedia Commons

What about AI?

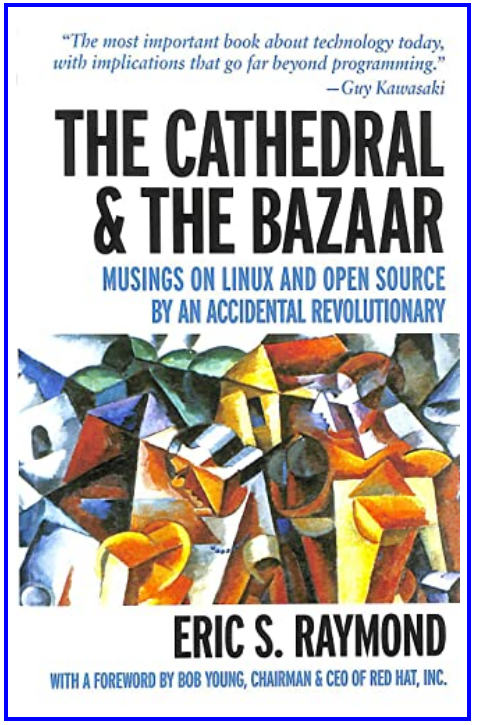
- Look at: Number of parameters in Large Language Models (LLMs)
- Only 5 reference points so far
- What we can say now:
  - Doubling every ≈ 4.3 months
  - Factor of 10 every ≈ 14 months



- Trend - so far (!): advance is much faster than that of Moore's law.
- Last point in curve (Chat-GPT-4 estimate): 100 trillion parameters!

Other factors:

- Chipmaking is very competitive – Technology highly protected
- In contrast: AI is basically 100 % open.
- Bazaar (bottom-up) won vs. Cathedral (top-down)
- A blessing for research/science but ...
- ... also a curse: containment issues



Containment: An old problem

From The Hitchhiker's Guide to the Galaxy:

> *" ... many elevators imbued with intelligence and precognition became terribly frustrated with the mindless business of going up and down, up and down, experimented briefly with the notion of going sideways, as a sort of existential protest, demanded participation in the decision-making process and finally took to squatting in basements sulking."*

How should we in Numerical Linear Algebra react?

- AI is *very* disruptive:
  - Students: only interested in AI-related work
  - Education side: Homeworks can now easily be solved by Chat-GPT4.
  - Research: labs (national, academic, corporate, ..) are pushing AI
- Danger: can NLA become irrelevant? [just used for its software?]
- Reasonable goal: Continue to seek innovation in Linear Algebra while also participating actively in Deep Learning research
- Difficulty: Culture, *Huge* community, different (unfamiliar) world

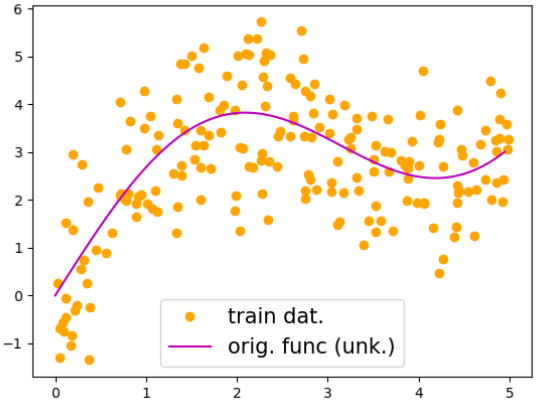
AI thinking vs. numerical analysis thinking: all about data

Trivial example: Given very noisy 'training points'

x_i, y_i to an unknown function f, recover f

NA : Interpolate in Least-Squares sense

➤ Need to select interpolant type, e.g., cubic

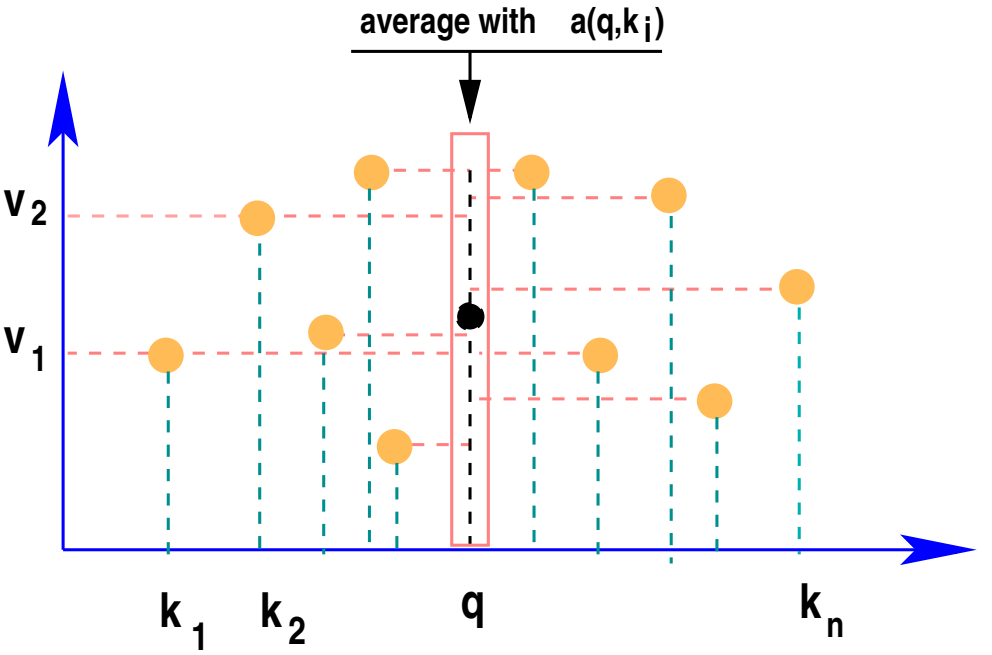


ML : use data points + some form of averaging with 'attention'.

- Given $\{k_i, v_i\}$ keys, values (NA: $x_i^{train}, y_i^{train}\}$)
- ... a query q (NA: The x where we want $f(x)$)
- ... and a Kernel $a(q, k)$. Approximation at q:

$$A(q) = \sum_i a(q, k_i) v_i$$

- "Attention" mechanism averages by giving more importance to points near q
- Nadaraya-Watson attention [Kernel Regression]

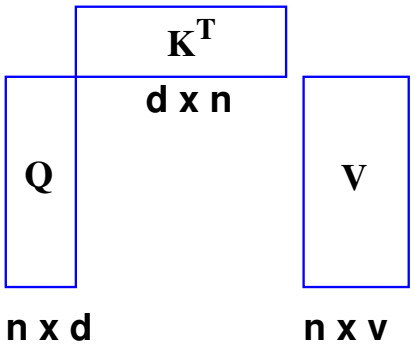


Example: A pure LA idea that is very successful

> *"Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention"*, A. Katharopoulos, A. Vyas, N. Pappas, F. Fleuret ('21)

- Scaled Dot-product Attention :
- Softmax applied row-wise
- $\boldsymbol{Q}$: $n \times d$, $\boldsymbol{K}$: $n \times d$, $\boldsymbol{V}$: $n \times v$

$$\boldsymbol{A}_l = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}}\right)\boldsymbol{V}$$



- Cost: $O(n^2)$ – But without the softmax term:
- Do $\boldsymbol{K}^T\boldsymbol{V}$ first – then $\boldsymbol{Q}\times$ result: $\rightarrow O(n)$ cost
- Idea: replace softmax$(\boldsymbol{Q}\boldsymbol{K}^T)$ by $\phi(\boldsymbol{Q})\phi(\boldsymbol{K}^T)$

  (Judicious func. ϕ applied rowwise to $\boldsymbol{Q}, \boldsymbol{K}$)

- Very simple idea. Very impactful paper [Huge gain in training time]

Example: Low-rank structure in DNN

> *"LoRa: Low-Rank Adaptation of Large Language Models"* E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen ('21)

➤ LoRa able to reduce number of parameters in Chat-GPT3 from 175B to $17M$ - (i.e., / by 10,000)

➤ Observation: Depth on DNN leads to low-dimension in paramaters ("Law of parsimony") *Over-parameterization → Low-Dim.*

➤ Many other papers, e.g., on analysis - example:

> *"The Low-Rank Simplicity Bias in Deep Networks"* M. Huh, H. Mobahi, R. Zhang, B. Cheung, P. Agrawal, P. Isola ('22)

➤ Main ingredient in analysis: "Random Matrix Theory"

PART 2: ACCELERATION METHODS

➤ Accelerators for linear systems: Conjugate Gradient, Conjugate Residual, GCR, ORTHOMIN, GMRES, BiCGSTAB, IDR, ... (*Krylov subspace methods*)

➤ Picture for nonlinear equations is more complex:

(a) Linear accelerators invoked when solving Jacobian systems iteratively in Newton $\rightarrow$ Inexact Newton methods

(b) Quasi-Newton methods, BFGS, LBFGS, ..., : approximate Jacobian or its inverse with Low-rank updates

(c) Anderson acceleration, Pulay mixing, ... nonlinear acceleration view-point + a (rough) linear model.

Acceleration of fixed point iterations

➤ Common situation: A (complex) physical simulation leading to a sequence of a physical quantity (charge densities, potentials, pressures, ...)

➤ Common approach: fixed point iteration $x_{k+1} = g(x_k)$

- Acceleration: combine $g(x_k)$ with previous iterates $\rightarrow$ faster convergence
- These methods try to solve: $f(x) = 0$ where $f(x) \equiv x - g(x)$

➤ Restriction: Use only function evaluations and lin. combinations

- To solve $f(x) = 0$ can apply an acceleration method for: $g(x) = x - \mu f(x)$
- (i.e., $f(x)$ viewed as the gradient in gradient descent for optimization)

Inexact Newton

We now focus on solving $f(x) = 0$ ($f : \mathbb{R}^n \to \mathbb{R}^n$). Newton Approach

Set $x_0 =$ an initial guess.
For $n = 0, 1, 2, \cdots$ until conv. do:
  Solve: $J(x_j)\delta_j = -f(x_j)$ (*)
  Set: $x_{j+1} = x_j + \delta_j$

$\leftarrow f(x_j + \delta) \approx f(x_j) + J(x_j)\delta$

with $J(x_j) = f'(x_j)$ = Jacobian at x_j

Standard Newton: solve (*) exactly

Inexact Newton methods: solve system (*) approximately.

Quasi-Newton methods: solve system (*) in which Jacobian is replaced by an estimate obtained from previous iterates.

Newton-Krylov methods: solve system (*) by a Krylov subspace method

Inexact Newton with Krylov methods

In Newton Krylov: $x_{j+1} = x_j + \delta_j$ where $\delta_j \equiv$ approx. solution of $\boxed{J\delta + f(x_j) = 0}$ by a Krylov subspace method

- Approximate sol. == $\boxed{\delta_j = V_l y_l}$ where V_l is an orthonormal basis of the Krylov subspace: $K_l = \mathbf{span}\{v, Jv, \cdots, J^{l-1}v\}$ with $v \equiv -f(x_j)$

- For example, if the method invoked is FOM, then:

$$\boxed{\delta_j = V_l (V_l^T J V_l)^{-1} V_l^T (-f(x_j))}$$

- In essence: inverse Jacobian approximated by the matrix

$$\boxed{B_{j,IOM} = V_l (V_l^T J V_l)^{-1} V_l^T}$$

Note: approximation for step j only – discarded in next step.

Quasi-Newton

➤ Quasi-Newton (QN) methods: build approximations to $J(x_j)$ or $J(x_j)^{-1}$, *progressively* using previous iterates

➤ Notation: $\Delta x_j \equiv x_{j+1} - x_j$, $\Delta f_j \equiv f(x_{j+1}) - f(x_j),$

➤ Secant condition: $J_{j+1}\Delta x_j = \Delta f_j,$

➤ *No-change condition:* $J_{j+1}q = J_j q, \quad \forall q$ such that $q^T \Delta x_j = 0.$

➤ Broyden: ∃! J_{j+1} that satisfies both conditions. Calculated as:

$$J_{j+1} = J_j + (\Delta f_j - J_j \Delta x_j)\frac{\Delta x_j^T}{\Delta x_j^T \Delta x_j}.$$

- Type II Broyden: Inverse Jacobian approximated by G_j at step j
- Secant condition:
- *No-change condition:*

$$G_{j+1}\Delta f_j = \Delta x_j, \qquad G_{j+1}q = G_j q, \quad \forall q \quad \text{such that} \quad q^T \Delta f_j = 0.$$

- Broyden (II): $\exists!$ G_{j+1} that satisfies both conditions. Calculated as:

$$G_{j+1} = G_j + (\Delta x_j - G_j \Delta f_j)\frac{\Delta f_j^T}{\Delta f_j^T \Delta f_j},$$

Note: Common feature of QN methods: The sequence of pairs of $\Delta x_i, \Delta f_i$ used to update previous approximation to $J(x_j)$ or $J(x_j)^{-1}$.

- Progressive low-rank approximation ...
- ... 'One rank at a time'

Anderson Acceleration

- Want fixed point of $g(x) : \mathbb{R}^n \to \mathbb{R}^n$. Let $f(x) = g(x) - x$.
- Select x_0 and define $x_1 = x_0 + \beta f_0$ [β is a parameter]

Given: x_i and $f_i = f(x_i)$ for $i = j - m, \cdots, j$

Let: $\Delta x_i = x_{i+1} - x_i, \quad \Delta f_i = f_{i+1} - f_i$ for $i = 0, 1, \cdots, j - m$

$P_j = [\Delta x_{j-m} \ \cdots \ \Delta x_{j-1}], \qquad V_j = [\Delta f_{j-m} \ \cdots \ \Delta f_{j-1}].$

Compute: $\boxed{x_{j+1} = \bar{x}_j + \beta \bar{f}_j}$ where: $\boxed{\bar{x}_j = x_j - P_j \, \theta^{(j)}, \ \bar{f}_j = f_j - V_j \, \theta^{(j)}}$

And: $\boxed{\theta^{(j)} = \text{argmin}_{\theta \in \mathbb{R}^m} \| f_j - V_j \, \theta \|_2}$

Note: Original article formulated problem in the standard 'acceleration' form

$$\bar{x}_j = \sum_{i=j-k}^{j} \mu_i^{(j)} x_i \quad \text{with} \quad \sum \mu_i^{(j)} = 1$$

- The $\mu_i^{(j)}$'s must now minimize $\left\| \sum_{i=j-k}^{j} \mu_i^{(j)} f_i \right\|_2^2$
- Mathematically equivalent to previous formulation

Relation with other methods

- AA is a *multisecant method* [Eyert '96, Fang-YS '09]
- Detailed study of relation with GMRES in linear case [Walker and Ni'11]
- Several other methods discovered independently turned out equivalent (or very close) to AA

Revisiting old friends: The GCR method

Goal: start with accelerators for linear problems - then see how to extend them to nonlinear case

Class of Krylov subspace methods:

- Conjugate gradient (Hestenes and Stiefel, '51), Conjugate Residual (Stiefel '55), Lanczos (51), Bi-CG (Fletcher 76)
- Accelerators developed in 1980s, 1990s:
  GCR, ORTHOMIN, GMRES, BiCGSTAB, IDR, ..

➤ We consider the *Generalized Conjugate Residual* (GCR) [Eisenstat, Elman, Schultz, '83]

GCR for linear case: $Ax = b$

ALGORITHM : 1. *GCR*

1: ***Input***: *Matrix* A, *RHS* b, *initial* x_0.
2: *Set* $p_0 = r_0 \equiv b - Ax_0$.
3: ***for*** $j = 0, 1, 2, \cdots$, *Until convergence* ***do***
4: $\quad \alpha_j = (r_j, Ap_j)/(Ap_j, Ap_j)$
5: $\quad x_{j+1} = x_j + \alpha_j p_j$
6: $\quad r_{j+1} = r_j - \alpha_j Ap_j$
7: $\quad p_{j+1} = r_{j+1} - \sum_{i=0}^{j} \beta_{ij} p_i$ *where* $\beta_{ij} := (Ar_{j+1}, Ap_i)/(Ap_i, Ap_i)$
8: ***end for***

- Recall: the set $\{Ap_i\}_{i=0,\ldots,j}$ is orthogonal

- Two practical variants

Restarting GCR(k) - restart every k steps

Truncation TGCR(m,k) - Truncated GCR: Orthogonalize against m most recent vectors only + restart dimension of k

- In TGCR(m,k) Line 7 becomes: [Notation: $j_m = \max\{0, j-m+1\}$]

$$p_{j+1} = r_{j+1} - \sum_{i=j_m}^{j} \beta_{ij} p_i \quad \text{where} \quad \beta_{ij} := (Ar_{j+1}, Ap_i)/(Ap_i, Ap_i)$$

- GCR(k): Eisenstat, Elman and Schultz [83] - equivalent to GMRES(k)

- TGCR initially developed by Vinsome '76 (as *ORTHOMIN*), analyzed in 1983 GCR paper

A few properties of (full) GCR in linear case

Notation: $\boxed{P_k = [p_0, p_1, \cdots, p_k]}$ $\boxed{R_k = [r_0, r_1, \cdots, r_k],}$ $\boxed{V_k = AP_k}$

Property: (Eisenstat-Elman-Schultz) *The residual vectors produced by (full) GCR are semi-conjugate, i.e.,* $(r_j, Ar_i) = 0$ *for* $i < j$.

Corollary: When $A = A^T$ residuals are conjugate

Property: *When* A *is nonsingular, (full) GCR breaks down iff it produces an exact solution.*

breakdown $\leftrightarrow$ 'lucky breakdown'

Property: Approximate solution at k-th step is $\boxed{x_{k+1} = x_0 + P_k V_k^T r_0}$

➤ We say that the algorithm induces the 'approximate inverse' $B_k = P_k V_k^T$ - a rank-k matrix. Let $\boxed{\mathcal{L}_k = \text{Span}(V_k)}$ and $\boxed{\pi = V_k V_k^T}$. Then

- $B_k = A^{-1}\pi \;\rightarrow\; B_k$ inverts A exactly in $\mathcal{L}_k$, i.e., $B_k\pi = A^{-1}\pi$.
- $AB_k = \pi$.
- When A is symmetric then B_k is self-adjoint when restricted to $\mathcal{L}_k$.
- $B_k A x = x$ for any $x \in \text{Span}\{P_k\}$, i.e., B_k inverts A exactly from the left when A is restricted to the range of P_k.
- $B_k A$ is the projector onto $\text{Span}\{P_k\}$ and orthogonally to $A^T\mathcal{L}_k$.

Nonlinear case: Exploit a multisecant viewpoint with GCR

- 1st approach: Inexact Newton. Well-known but has disadvantages.
- Instead we will develop a multisecant approach.
- Linear TGCR builds m directions such that:

$\{Ap_{j_m}, \cdots Ap_j\}$ is orthogonal

- In nonlinear case we can still use this basis– where A is 'some' Jacobian
- Assume that at step j we have a set of (at most) m current 'search' directions $\{p_i\}$ for $i = j_m, j_m + 1, \cdots, j$
- Along with $v_i \approx J(x_i)p_i$, $i = j_m, j_m + 1, \cdots, j$
- Set: $P_j = [p_{j_m}, p_{j_m+1}, \cdots, p_j], \qquad V_j = [v_{j_m}, v_{j_m+1}, \cdots, v_j].$

- Note: In Linear case or Inexact Newton case $v_i = Jp_i$ (J is fixed)
- p_i and v_i are 'paired' much like the vectors Δf_i and Δx_i of QN and AA
- Notation $\boxed{V_j \approx [J]P_j}$

Main Idea of Nonlinear Extension:

- Just build orthonormal basis V_j as in TGCR
- Do usual projection step to minimize 'linear residual' - i.e.,

$$x_{j+1} = x_j + P_j y_j \qquad \text{where} \qquad y_j = \text{argmin}_y \|f(x_j) + V_j y\|$$

- Note: V_j orthonormal $\rightarrow$ $y_j = V_j^T(-f(x_j)) \equiv V_j^T r_j$

ALGORITHM : 2. *nlTGCR(m)*

1: ***Input***: $f(x)$, *initial* x_0.
2: *Set* $r_0 = -f(x_0)$.
3: *Compute* $v = J(x_0)r_0$; ▷ *Use Frechet*
4: $v_0 = v/\|v\|, p_0 = r_0/\|v\|_2$;
5: ***for*** $j = 0, 1, 2, \cdots,$ ***do***
6: $y_j = V_j^T r_j$
7: $x_{j+1} = x_j + P_j y_j$ ▷ *Scalar* α_j *becomes vector* y_j
8: $r_{j+1} = -f(x_{j+1})$ ▷ *Replaces linear update:* $r_{j+1} = r_j - V_j y_j$
9: *Set:* $p := r_{j+1}$;
10: *Compute* $v = J(x_{j+1})p$ ▷ *Use Frechet*
11: ***for*** $i = j_m : j$ ***do***
12: $\beta_{ij} := \langle v, v_i \rangle$
13: $p := p - \beta_{ij} p_i$; $v := v - \beta_{ij} v_i$;
14: ***end for***
15: $p_{j+1} := p/\|v\|_2$; $v_{j+1} := v/\|v\|_2$;
16: ***end for***

A few properties

➤ Notation: $\tilde{r}_{j+1} = r_j - V_j y_j$ (Linear Residual) ; $z_j = \tilde{r}_j - r_j$

The following properties are satisfied by the vectors produced by ***nITGCR***:

1. The system $[v_{j_m}, v_{j_m+1}, \cdots, v_{j+1}]$ is orthonormal.
2. $(\tilde{r}_{j+1}, v_i) = 0 \quad \text{for} \quad j_m \leq i \leq j$, i.e., $V_j^T \tilde{r}_{j+1} = 0$.
3. $\|\tilde{r}_{j+1}\|_2 = \min_y \|f(x_j) + V_j y\|_2$
4. $(v_{j+1}, \tilde{r}_{j+1}) = (v_{j+1}, r_j)$
5. $V_j^T r_j = (v_j, \tilde{r}_j) e_1 - V_j^T z_j$ where $e_1 = [1, 0, \cdots, 0]^T \in \mathbb{R}^{m_j}$ with $m_j \equiv \min\{m, j+1\}$.

➤ What can we say about the deviation z_j?

A few properties (cont.)

- Can show: the difference $\tilde{r}_{j+1} - r_{j+1}$ is of "second order"
- Hence: can switch to linear form of residual at some point
- Saves one fun. eval

➤ *Multisecant property*

➤ Observe that the update at step j takes the form:

$$x_{j+1} = x_j + P_j V_j^T r_j = x_j + P_j V_j^T (-f(x_j))$$

➤ Thus, we are in effect using a secant-type method with the Approximate inverse Jacobien:

$$G_{j+1} = P_j V_j^T$$

➤ In addition:

The unique solution to the problem

$$\min\{\|B\|_F \text{ subject to: } \quad BV_j = P_j\}$$

is achieved by the matrix $G_{j+1} = P_j V_j^T$.

➤ Yet another multi-secant type method, but ...

- The method shares also characteristics of inexact Newton, e.g., ..
- .. can add global convergence strategies like backtracking
- The relation $v_j \approx J(x_j)p_j$ should be fairly accurate [Frechet diff.]
- Contrast with the relation $\Delta f_j \approx J\Delta x_j$ (Anderson, QN)
- Two function evaluations per iteration but ...
- ... can be reduced to one as soon as r_j becomes close to $\tilde{r}_j$ ($\sim$ linear)

General GCR framework

- There are known situations where Anderson does quite well..
- e.g., Picard iteration for Navier Stokes

Q: Can we implement Anderson acceleration in the form of GCR?

A: Yes -

General Framework: At step j: define a new pair of vectors: p_{j+1} and v_{j+1} - with the requirement that $v_{j+1} \approx J p_{j+1}$ - Where J = Jacobian at x_j

- As before, orthonormalize to get p_{j+1}, v_{j+1}
- Same update as before: $x_{j+1} = x_j + P_j y_j$

General GCR framework: A matter of pairing

➤ In nlTGCR: [before orthogonalization] – Pair:

1. $p_{j+1} = r_{j+1}$ and ...
2. $v_{j+1} = Jr_{j+1}$ == explicitly computed.

➤ Recall Notation: $\tilde{r}_j = r_j - V_j y_j$ = 'Linear' residual ➤ Then nlTGCR will give the exact same iterates as (full) Anderson if:

1. $p_{j+1} = P_j y_j + \beta \tilde{r}_j$ and $x_{j+1} = x_j + p_{j+1} \equiv \underbrace{(x_j + P_j y_j)}_{\text{nlTGCR}} + \beta \tilde{r}_j$
2. $v_{j+1} = r_{j+1} - r_j$ where $r_{j+1} = f(x_{j+1})$

➤ Subtle point: Truncated versions are not the same.

➤ Let us explore this a little further

General GCR framework: Exploring AA from GCR angle

We can implement 3 different versions of AA which are all mathematically equivalent in the full window case ($m = \infty$).

1. Standard *AA* where LS problem is solved via downdating QR [or just NE]
2. nlTGCR with the pairing just given. We call this *nlTGCR_AA*
3. Anderson acceleration where a (Truncated) Gram-Schmidt process is applied to the columns of V_j and and same linear mixing is applied to P_j. We call this *AA_TGS*

- $V_j \longrightarrow V_j = Q_j S_j$ (Gram-Schmidt QR); $q_j = v_j - \sum s_{ij} q_i$
- $P_j \longrightarrow P_j = U_j S_j$ (same transf. as V_j) $u_j = p_j - \sum s_{ij} u_i$

Anderson-TGS

1: **Input**: x_0, f, β..
2: $f_0 = f(x_0)$;
3: $x_1 = x_0 + \beta f_0$ Compute $f_1 = f(x_1)$
4: **for** $j = 1, 2, \cdots$, until convergence **do**
5: $q = f_j - f_{j-1}$; $u = x_j - x_{j-1}$
6: **for** $i = j_m, \cdots, j-1$ **do**
7: $s_{ij} = q_i^T q$
8: $q := q - s_{ij} q_i$; $u := u - s_{ij} u_i$
9: **end for**
10: $q_j = q/s_{jj}$, $u_j = u/s_{jj}$ where $s_{jj} = \|q\|$
11: $x_{j+1} = (x_j - U_j\theta) + \beta(f_j - Q_j\theta)$ where $\theta = Q_j^T f_j$
12: $f_{j+1} = f(x_{j+1})$
13: **end for**

➤ Assume: $\boxed{f(x) = Ax - b,\ A = A^T, \text{ and } m = \infty}$

➤ Then something remarkable happens:

$$S = \begin{bmatrix} \star & \star & \star & 0 & 0 & 0 \\ & \star & \star & \star & 0 & 0 \\ & & \star & \star & \star & 0 \\ & & & \star & \star & \star \\ & & & & \star & \star \\ & & & & & \star \end{bmatrix} \qquad \theta = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \star \\ \star \end{bmatrix}$$

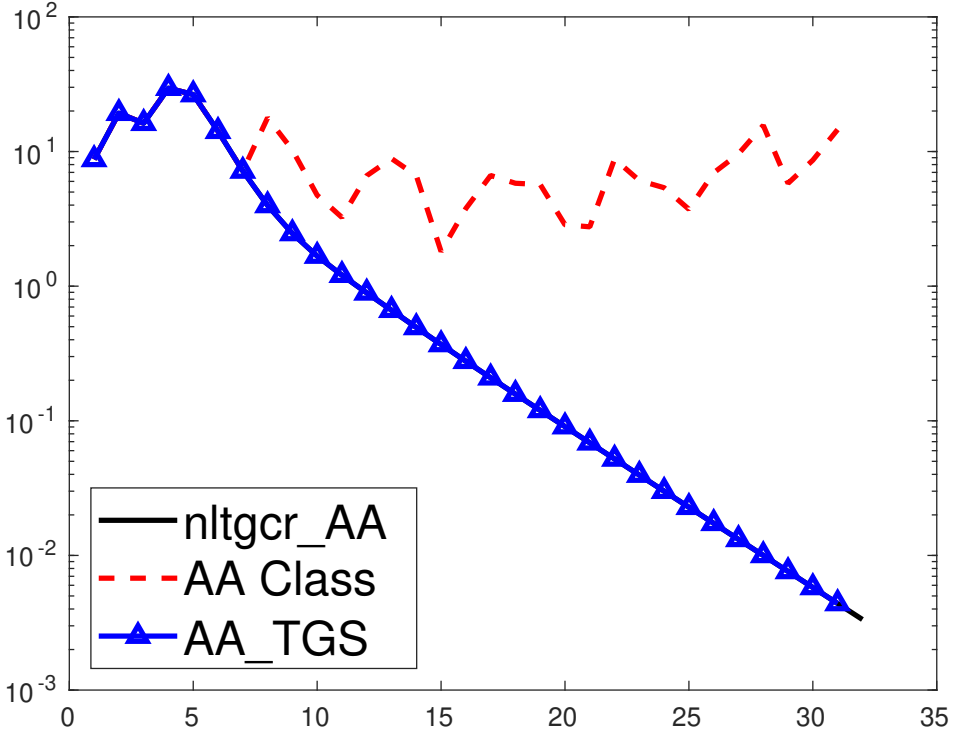
1. The upper triangular matrix S is *tridiagonal*: $s_{ij} = 0$ for $i < j - 2$
2. $\theta = Q_j^T f_j$ has only 2 nonzero components (last 2).

➤ In other words: *the algorithm simplifies in the linear symmetric case.*

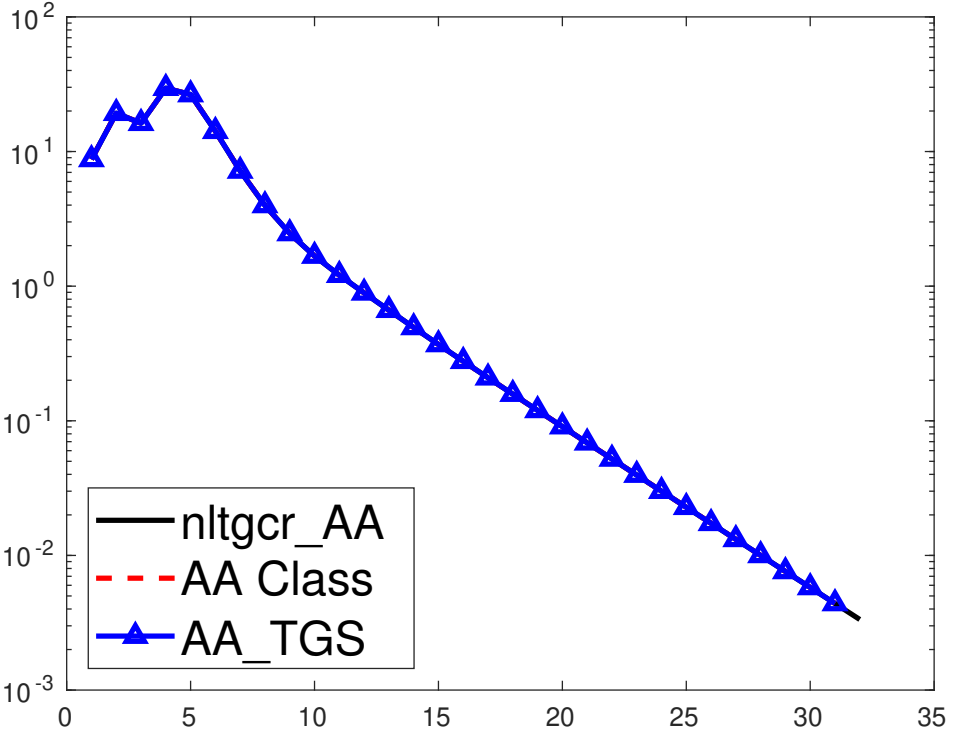
- Let us go back to the general case
- Any relation between: *AA*, *nlTGCR_AA*, and *AA_TGS*?
- It is (very) easy to see that:

1. *AA_TGS* and *nlTGCR_AA* are mathematically equivalent
2. *AA_TGS*, *nlTGCR_AA* and *AA* are equivalent when $m = \infty$

➤ Expl: 30 iterations of the 3 methods on a linear system of size 100



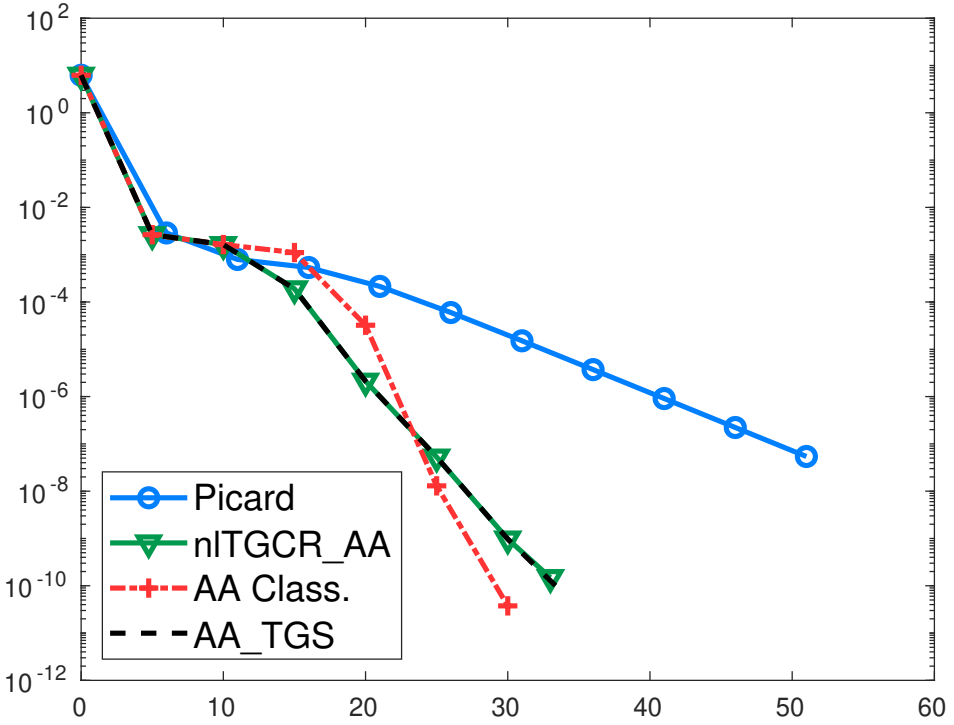
$m = 5$



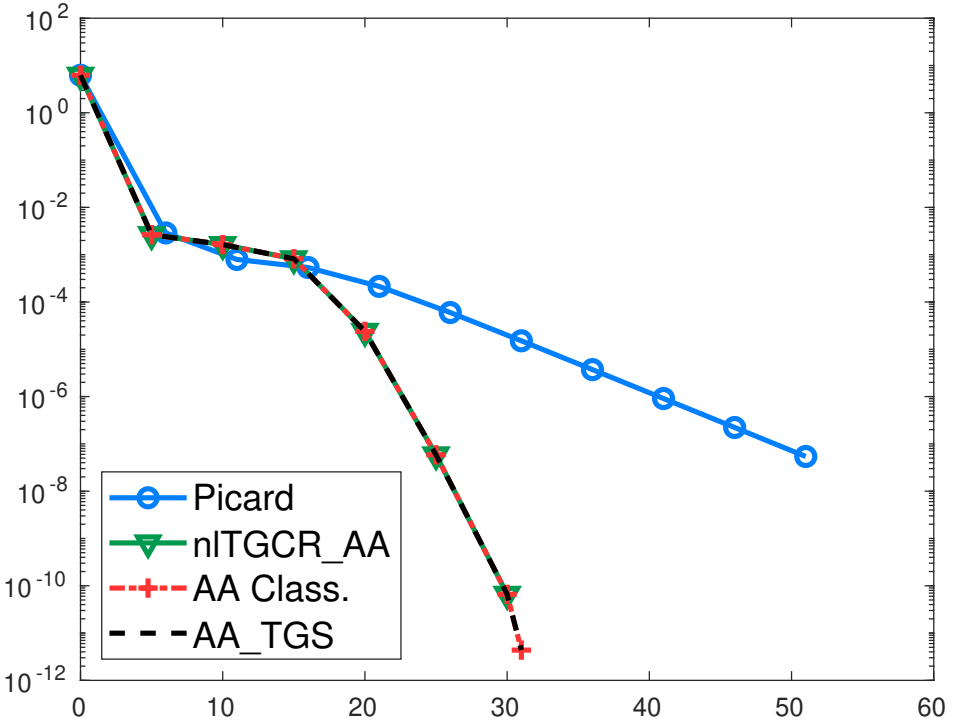
$m = \infty$

➤ Note: system is symmetric

➤ Similar example: a Small nonlinear problem $(n = 2242)$: Navier Stokes equations [back-step] using *ifiss*



$m = 10$



$m = \infty$

➤ Note: Problem is nonlinear + no obvious symmetry

- In effect AA_TGS is a 'symmetric' variant of AA which is ...
- ... equivalent to standard AA in case $m = \infty$.

➤ Obtained by using formalism of nlTGCR

Consequence: For optimization problems - Hessian is symmetric $\rightarrow$ a window of $m = 3$ is enough and optimal.

➤ Further studies needed [Experimentation + Theory]

➤ Experiments to be discussed next are not concerned these variants

Experiments - Bratu problem

➤ Illustrates the importance of exploiting symmetry [Recall: in linear symmetric case GCR becomes CR, requires window-size of 2]

➤ .. and importance of adaptive version

Nonlinear eigenvalue problem (Bratu)

➤ Take $\lambda = 0.5$.

$$-\Delta u = \lambda e^u \text{ in } \Omega = (0,1) \times (0,1)$$
$$u(x,y) = 0, \text{ for } (x,y) \in \partial\Omega$$

➤ FD discretization with grid of size $100 \times 100 \rightarrow$ Problem size = $n = 10,000$

The Adaptive update version

- Bratu problem is almost linear – true in general when nearing convergence
- Idea: exploit the linearized update version of nlTGCR to cut number of func. evals. by $\approx$ half
- Need an adaptive mechanism: switch from the nonlinear to linear updates - [$\approx$ linear regime]
- and switch back when needed
- Define the nonlinear and nonlinear res. at step j:

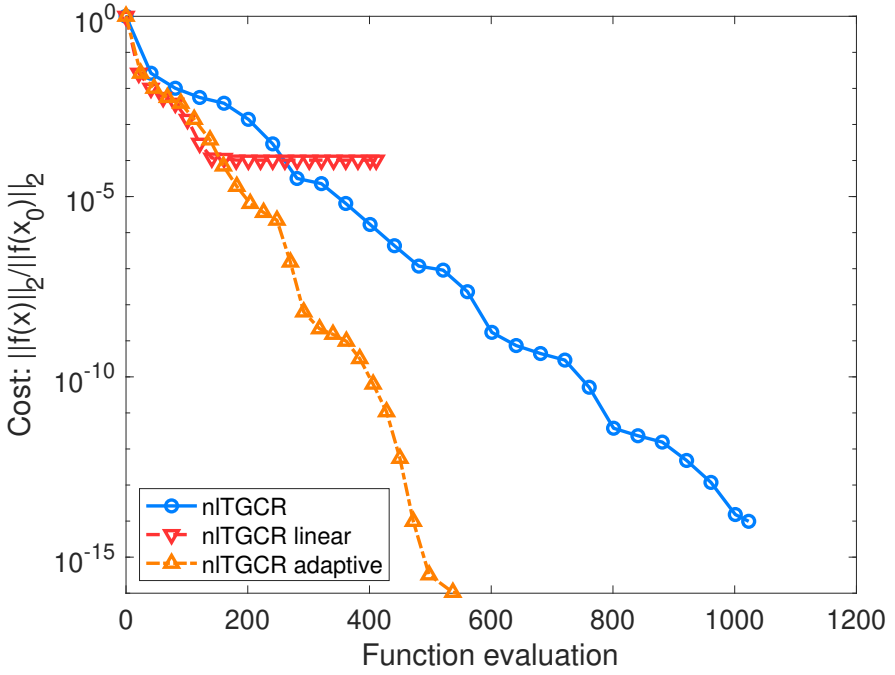
$$\boldsymbol{r}_{j+1}^{nl} = -\boldsymbol{f}(\boldsymbol{x}_{j+1}),$$
$$\boldsymbol{r}_{j+1}^{lin} = \boldsymbol{r}_{j}^{nl} - \boldsymbol{V}_j \boldsymbol{y}_j.$$

- Criterion will use the angular distance between the two vectors:

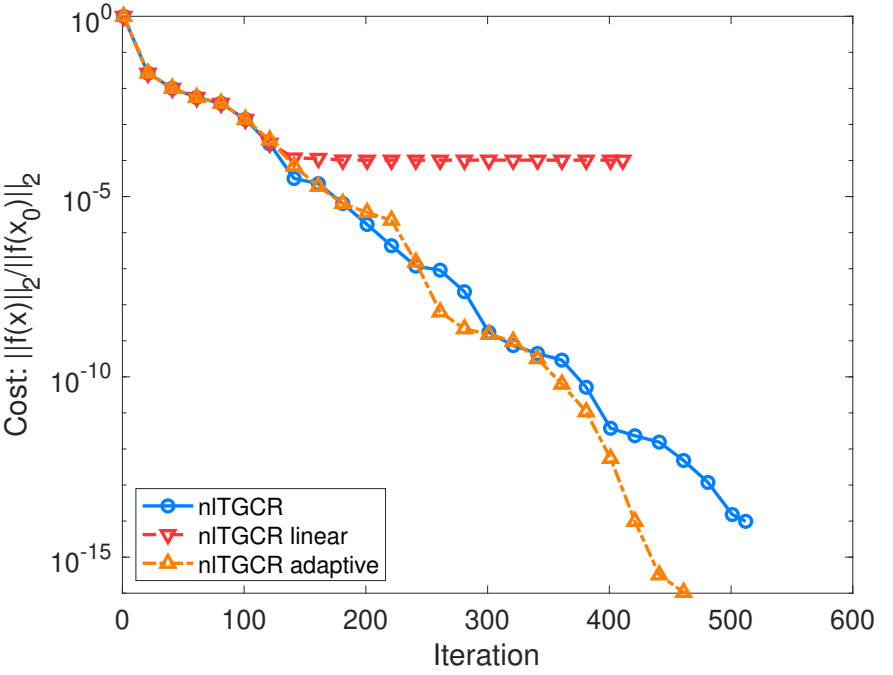
$$d_j := 1 - \frac{(r_j^{nl})^T r_j^{lin}}{\|r_j^{nl}\|_2 \cdot \|r_j^{lin}\|_2}$$

- Linear updates turned on when $d_j < \tau$, where τ is a threshold
- Check d_j regurlarly, for example, every 10 iterations,
- Switch back to nonlinear updates when $d_j \geq \tau$
- In experiments: $\tau = 0.01$

- Window size $m = 1$,



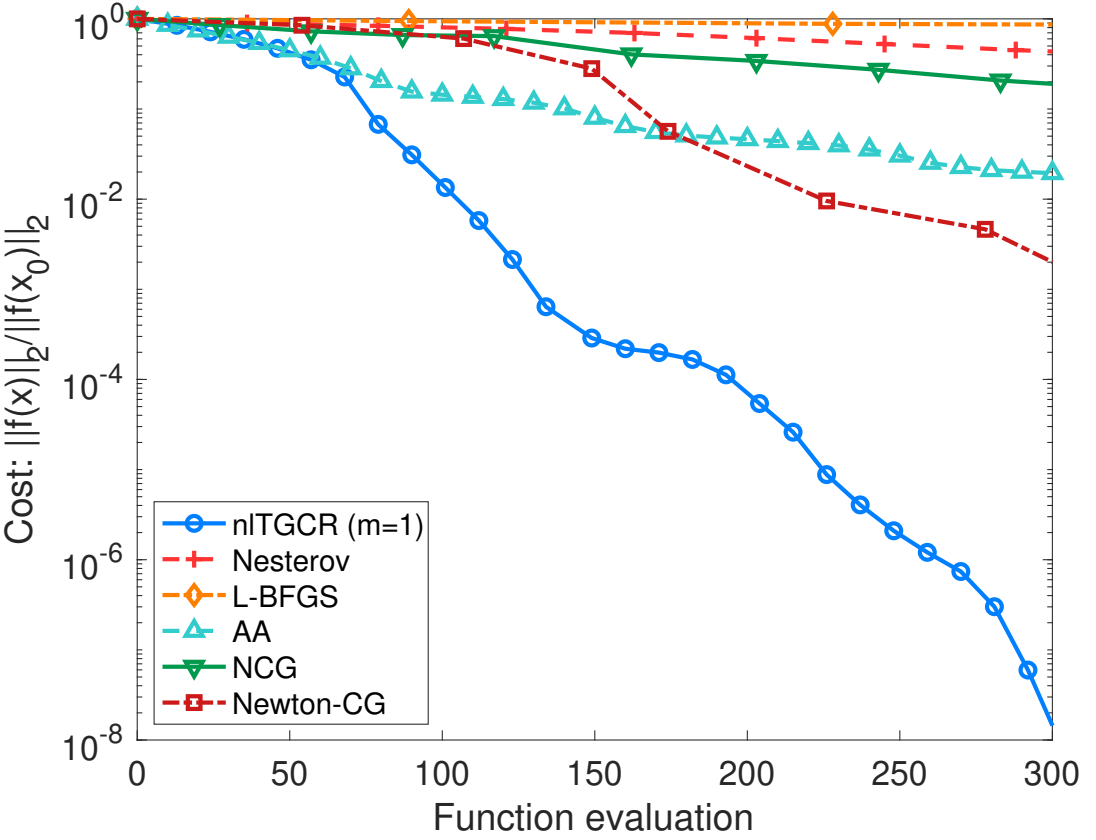
Function evaluations.



Iterations

Exploiting symmetry

Bratu problem with:
AA, L-BFGS, Nonlinear CG (NCG), [fletcher reeves], and Inexact Newton with CG (Newton-CG).



Molecular optimization with Lennard-Jones potential(*)

- Illustrates the importance of a global strategy - linesearch / backtracking + exploiting the Jacobian at multiple points
- Goal: find atom positions that minimize total potential enery:

Lennard-Jones Potential (x_i = position of atom i)

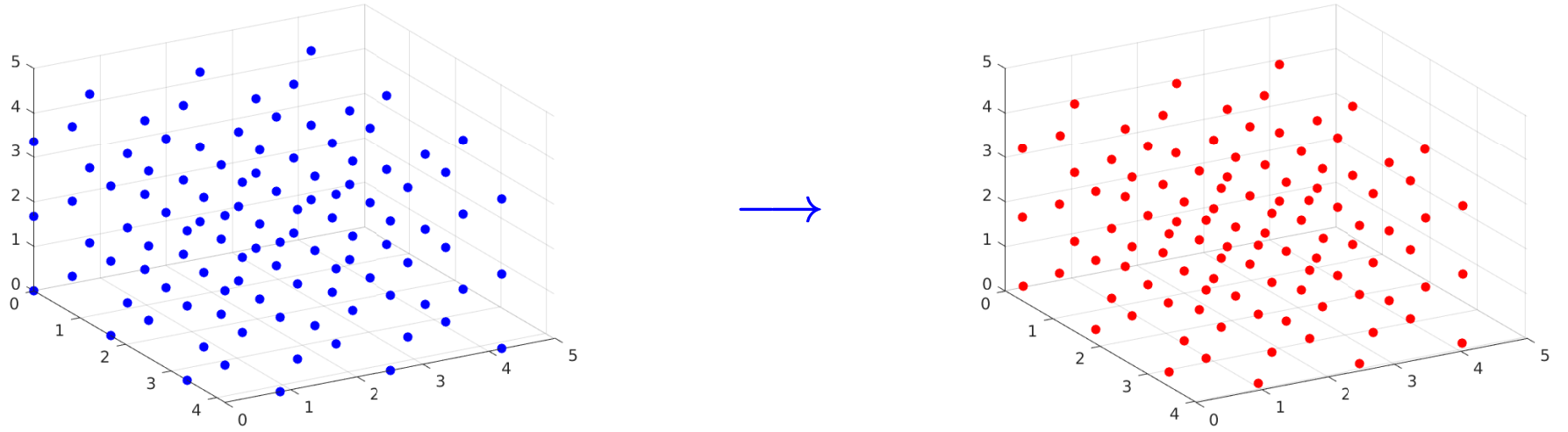
$$E = \sum_{i=1}^{Nat} \sum_{j=1}^{i-1} 4 \times \left[\frac{1}{\|x_i - x_j\|^{12}} - \frac{1}{\|x_i - x_j\|^{6}} \right]$$

Initial Config → Iterate to mininmize $\|\nabla E\|^2$ → Final Config

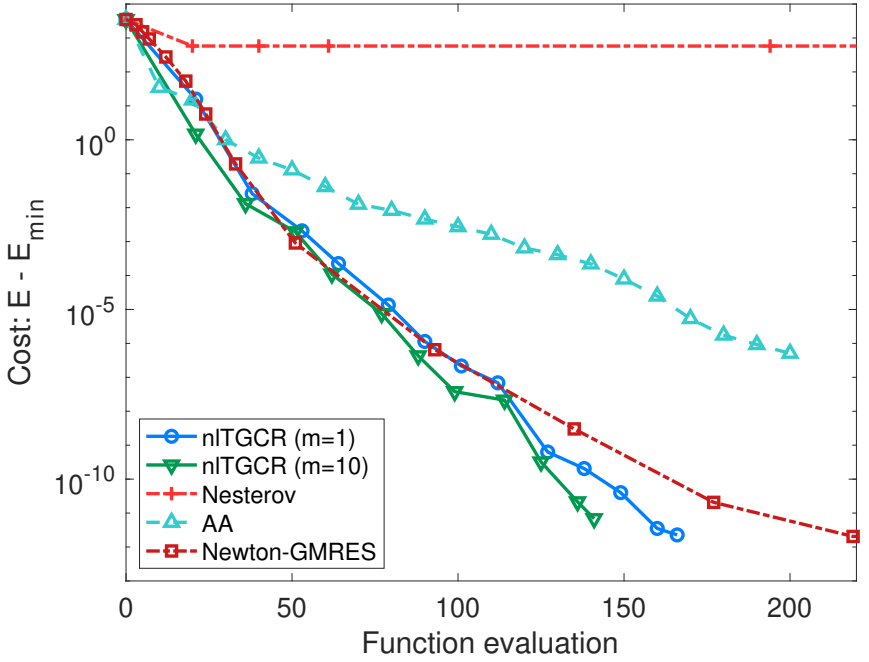
- Difficult problem due to high powers → Backtracking essential

---

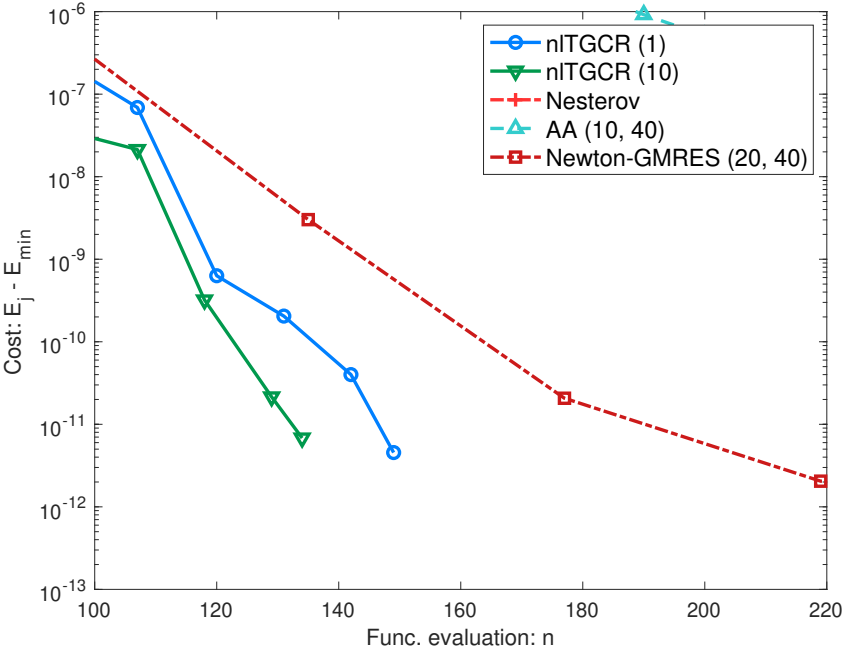
(*) *Thanks:* Stefan Goedecker's course site - Basel Univ.



- Initial geometry: 'Face-Centered Cube' + perturbation
- Adaptive gradient method: $x_{j+1} = x_j - t_j \nabla E(x_j)$ – with t_j adapted – can be made to work fairly well.
- AA will fail unless underlying fixed point iteration selected carefully: $x_{j+1} = x_j - \mu \nabla E(x_j)$ where $\mu \sim 10^{-3}$. Also must take $\beta \sim 10^{-2}$.

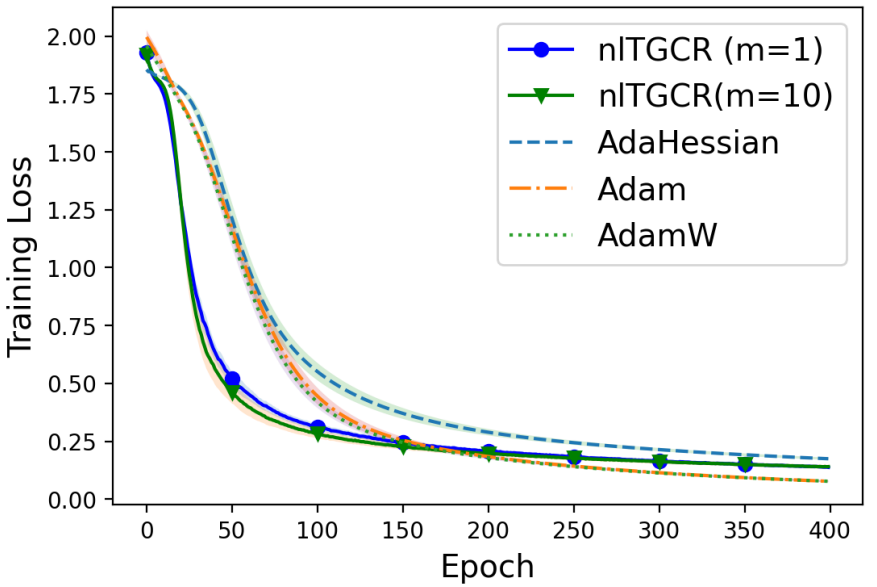


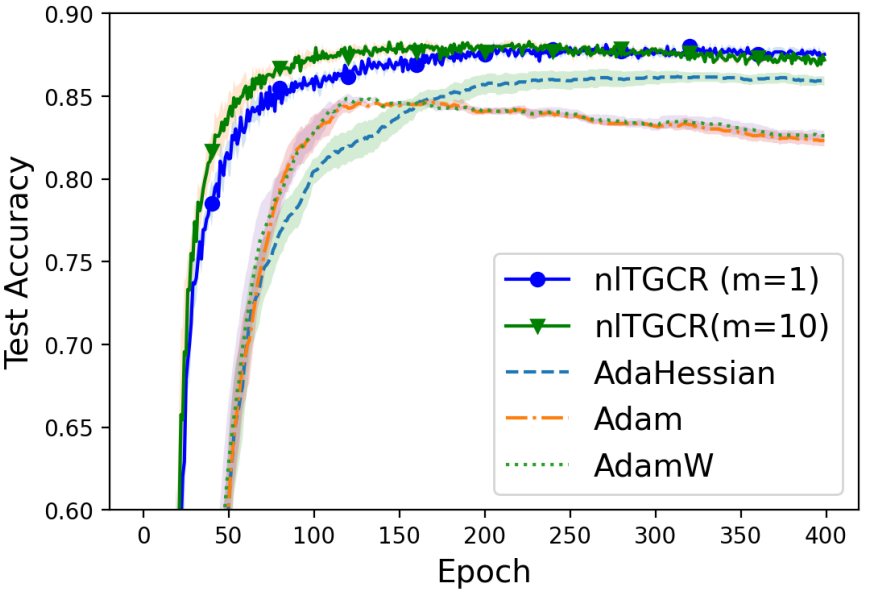
Lennard-Jones problem.)



Zoom near convergence

Dataset: ***Cora*** [2708 scientific pubs., 5429 links, 7 classes]. Goal: node classification [topic of paper from words and links]

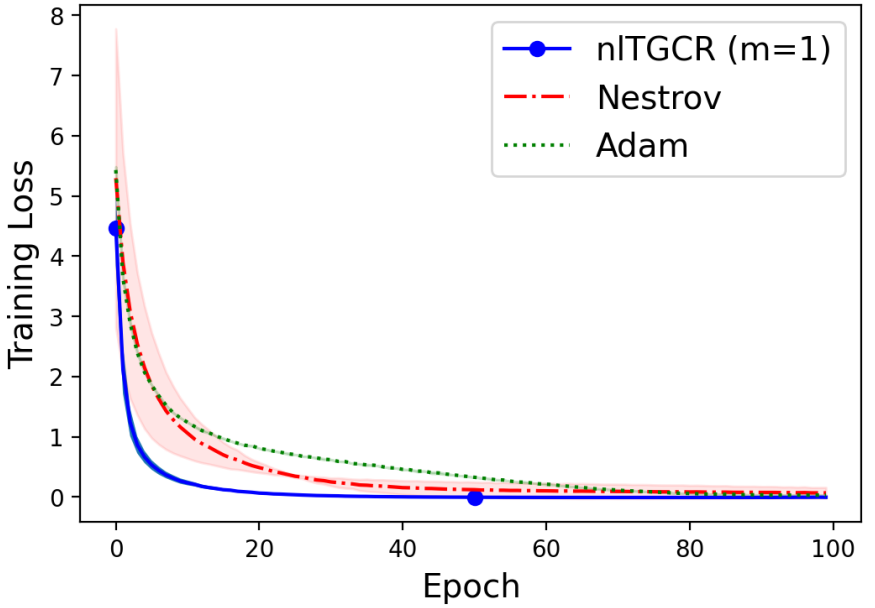


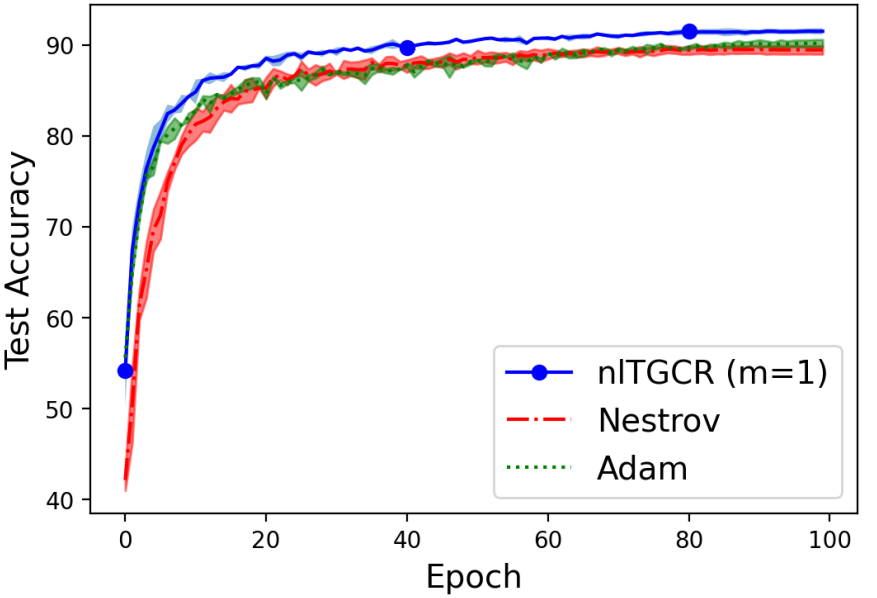


nlTGCR vs. Adam: training loss and validation accuracy

Image classification with CIFAR10 dataset

- Test using ResNet-18 architecture. Implemented with PyTorch -
- Compared nlTGCR with Adam and Momentum (Nesterov)





- Accuracies achieved: nlTGCR: 91.56, Adam: 90.13; Momemtum: 89.53

Concluding remarks

- Method can be adapted to context of stochastic gradient-type methods
- In deep learning: build P_j, V_j across different batches [i.e., ignore change of objective function with each batch]
- Challenge: QN-type methods exploit smoothness but ...
- ... Stochastic character limits smoothness.
- A lot more remains to be done to answer the question:

  *Can 2nd-order type methods be adapted to become *generally* superior to existing approaches in stochastic context?*