In the case of citing this presentation, cite it please as below:
Kecman V, The Good Old is The Good New Too, Gauss-Seidel, Newton-Raphson, Machine Learning & Data Mining, Seminar at VCU, Richmond, VA, 2014

The Good Old is The Good New Too

Gauss-Seidel, Newton-Raphson & Machine Learning

Vojislav **Kecman**

Learning Algorithms and Applications Laboratory (LAAL)
Seminar at VCU, Dec 5, 2014, Richmond, VA

Who can solve this system of equations?

$$4x_1 - x_2 = 5$$
$$-x_1 + 4x_2 = -5$$

Sure, all of you can do it. You would, for example, go this way

$x_2 = 4x_1 - 5$
$4(4x_1 - 5) - x_1 = -5$
$x_1 = 1,\quad x_2 = 4*1 - 5$
$x_2 = -1$

or, this one → **x = A**$^{-1}$**b**

But, what if you had this problem,

$$2x_1 - x_2 + \cdots - 3x_{100,000} = 4$$
$$\vdots$$
$$-x_1 + 3x_2 - \cdots + x_{100,000} = 13$$

or this one.

$$4x_1 - x_2 = 5$$
$$-x_1 + 4x_2 = -5$$
$$s.t.\quad x_i \ge 0,\quad i = 1, 2$$

80GB needed for storing system matrix here

Solve this, if you can?

$$4x_1 - x_2 = 5$$
$$-x_1 + 4x_2 = -5$$
$$s.t.\quad x_i \ge 0,\quad i = 1, 2$$

This is known as CONSTRAINTS

Ooops! It seems simple but **nobody** can do it! Well, any bold guess?

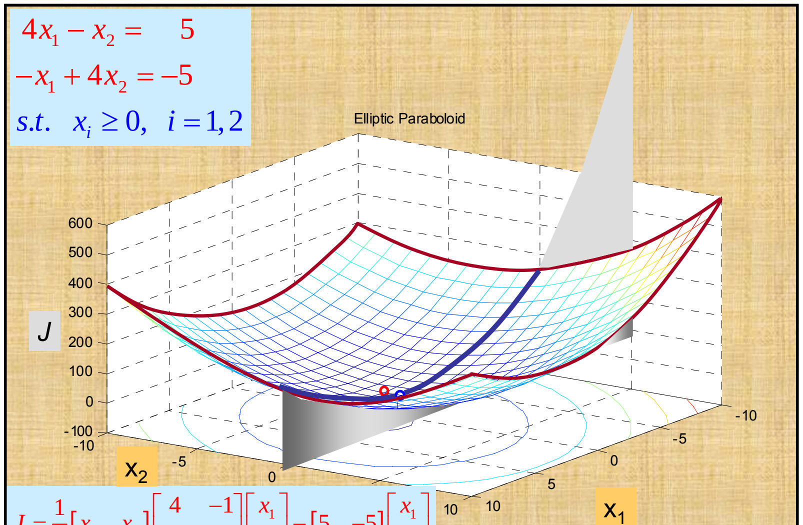
If there's anybody here who's got the solution to this problem,

LEAVE THE ROOM, GET RICH, HAVE A LIFE and **let the rest of us, losers, keep suffering!!!**
You deserved it,

providing, you've got x_1 = 1.25, x_2 = 0! If not, just **keep sitting** please!

Let's see this story above geometrically i.e., in graphs.

$$4x_1 - x_2 = 5$$
$$-x_1 + 4x_2 = -5$$
$$s.t.\quad x_i \ge 0,\quad i = 1, 2$$



$$J = \frac{1}{2}\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 5 & -5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$s.t.\quad x_i \ge 0,\quad i = 1, 2$$

$$J = \frac{1}{2}\left(4x_1^2 + 4x_2^2 - 2x_1x_2\right) - 5x_1 + 5x_2$$

Before going any further, just a word about the terminology and concepts

Previous example can be looked at as solving an optimization problem

$$\min_{\mathbf{x}} J = \frac{1}{2}\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 5 & -5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad s.t. \quad x_i \geq 0, \quad i = 1, 2$$

taking the derivative of *J*, **g = ∂J / ∂x**, one obtains the **gradient g**

$$\mathbf{g} = \begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 5 \\ -5 \end{bmatrix}, \quad or \quad g_1 = 4x_1 - x_2 - 5, \quad g_2 = -x_1 + 4x_2 + 5$$

The above is same as solving a system of equations

$$\begin{aligned} 4x_1 - x_2 &= 5 \\ -x_1 + 4x_2 &= -5, \quad s.t. \quad x_i \geq 0, \quad i = 1, 2 \end{aligned}$$

For which the **errors** in solutions can be expressed as

$$e_1 = 5 - 4x_1 + x_2, \quad e_2 = -5 + x_1 - 4x_2$$

Imagine now that your problem is posed as

$$\begin{aligned} 2x_1 - 1.2x_2 + \cdots - 1.5x_{999,999} + 1.1x_{1,000,000} &= 1 \\ \vdots \qquad \vdots \qquad & \vdots \\ 1.1x_1 + 1.9x_2 - \cdots + 1.2x_{999,999} + 2x_{1,000,000} &= 1 \\ s.t. \quad x_i \geq 0, \quad i = 1, 2, 3, \cdots, 1,000,000 & \end{aligned}$$

This kind of tasks is what VCU's
Learning Algorithms and Applications Laboratory (LAAL)
is trying to solve with both **decent accuracy** and **decent CPU time**.

Contents

Motivations for this work

Solve **Ax = b** for **x**

- No direct solution for either **small/medium** linear systems with constraints or the **huge** ones either with or without constraints.
- Such problems arise in modern machine learning (ML) i.e., data mining (DM) with ***millions of records*** as well as in almost all the other areas of **modern** science and engineering.

Definition of **HUGE linear systems**:

Huge = when system matrix **A can't be stored** and operated with / on / in a computer memory

Remark

In both cases mentioned, direct calculations of **x** (either by Gaussian elimination, or by inversion, or by LU, QR, Cholesky i.e., by any other factorization) are not feasible/possible and we must resort to the **iterative solutions!**

One more remark

- ML is not the first scientific field facing humongous number of equations. Many other areas have been doing it for decades e.g., **Solving PDEs**.
- What is so particular about (L1 and L2) **SVM** models?
  - **1st system of equation is not sparse. In fact, it is always extremely dense. It often has a system matrix with a high condition number, say > 10^8 i.e., system is very ill-conditioned**
  - **2nd system matrix is both symmetric & positive definite**
  - **3rd**
    - **for** L1 SVMs**, constraints** are usually ***box*** **constraints** accompanied by **1 only equality constraint**
    - **for** L2 SVMs**, constraints** are just **nonnegative ones.**

  In both cases, constraints make solution x to be sparse

- The last three facts are very different in respect to the classic huge linear system of equations originating from PDE solving
- They exclude all the conclusions about what iterative method is possibly the best
- One of the basic advices was that the **Conjugate-Gradient method is The Tool**
- **It is not for our very dense systems**!!!

Hence, we have to invent something better, more suitable, for the new problem setting.

Welcome to the Good Old Iterative Methods Ready for Renewal

- Jacobi
- Gauss-Seidel
- Successive Over Relaxation
- Steepest Descent
- Conjugate gradient
- Active-Set Approaches
- etc,…

For a **dense, symmetric positive definite (SPD), systems** we have shown that actually **only Gauss-Seidel** and **its SOR** can be efficient

- ~~Jacobi~~
- Gauss-Seidel
- Successive Over Relaxation
- ~~Steepest Descent~~
- ~~Conjugate gradient~~
- ~~Active-Set Approaches~~
- etc,…

As for the strikings shown, see the papers from **Zigic-Kecman** in 2013 & 2014

as well as my Springer book, 2005

SVMs & Linear System of Equations

- **For L1-SVM, learning means solving the QP problem below**

$$\arg\min_{\boldsymbol{\alpha}} L_d = \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{n} \alpha_i$$

$$s.t. \quad C \geq \alpha_i \geq 0, \quad i = 1, n \quad n \text{ inequality constraints}$$

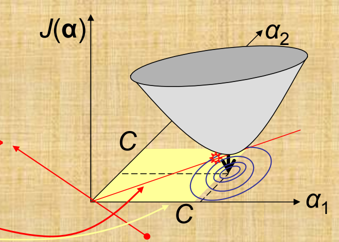
$$\sum_i^n \alpha_i y_i = 0 \qquad 1 \text{ only equality constraint}$$

$C \geq \alpha_i \geq 0$. These are known as the **BOX constraints**.

Matrix Notation and Geometry of QP Setting

$$\arg\min_{\boldsymbol{\alpha}} J = 0.5\boldsymbol{\alpha}^t \mathbf{H}\, \boldsymbol{\alpha} - \mathbf{1}^t\boldsymbol{\alpha}$$

$$s.t. \quad C \geq \alpha_i \geq 0, \quad \sum_{i=1}^{l} \alpha_i y_i = 0$$



Solving a QP problem is same as finding the solution of a linear system of equations

Hα - 1 = 0 or **Hα = 1**

subject to the very **same constraints.**

Matrix **H** is an (*n, n*) SPD matrix and *n* is a number of records.

Iterative Single Data Algorithm ISDA

- Between 2002 and 2005 **I** have**,** jointly with then PhD students and Drs. **Huang** and **Vogt** today**,** developed **ISDA** algorithm. We have shown that ISDA is actually equal to
- 1 – **SMO*** without bias term,
- 2 – Kernel **AdaTron** algorithm

*SMO = Sequential minimal optimization

and that **all** the **three approaches are** actually **Gauss-Seidel methods** for solving linear systems of equations under the given SVMs' constraints.

*SMO, developed by Dr. Platt at Microsoft Research, is the world most used (i.e., the working horse) algorithm for training SVMs

ISDA

- **ISDA is quite The Algorithm**, competing with and (often) beating the best methods for training SVMs.
- This is why in its newest releases **in 2014 MATLAB has** (out of few dozens SVM algorithms proposed) **implemented our ISDA** (together with SMO) **as the default algorithm for training large SVMs.**
- Check **fitcsvm.m** at Mathworks (MATLAB)

Fine, but if my students and I had already invented it, developed it and implemented it what is this seminar then about?

- Well, it's about what comes in next 34 slides.
- In essence, the story is as follows:

  Zigic, Strack, and Kecman have invented and recently proposed a novel **DL2 SVM** model for very large ML tasks which literally **cries** for an even more efficient training algorithm than ISDA

- **While searching for it, I've got some new insights which I want to share with you** today

The novel SVM model dubbed **Direct L2 SVM Algorithm** (**DL2 SVM**) boiles down to solving this problem:

H x = 1 subject to $x_i \geq 0, \; i = 1, \ldots, n$

and, this is a very well known

NONNEGATIVE (least squares) PROBLEM

See papers from **Zigic-Kecman** in 2013 & 2014

How to solve a system of linear equations having huuuuuuuuge symmetric positive definite matrix H and (possibly) some constraints

???

Well, let's go back to the

Good Old Gauss-Seidel iterative method for solving a system of linear equations having the SPD matrix H

Hx = b

$$h_{11}x_1 + h_{12}x_2 + h_{13}x_3 + \ldots + h_{1n}x_n = b_1$$
$$h_{21}x_1 + h_{22}x_2 + h_{23}x_3 + \ldots + h_{2n}x_n = b_2$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$h_{n1}x_1 + h_{n2}x_2 + h_{n3}x_3 + \ldots + h_{nn}x_n = b_n$$

See the rewritten i-th equation f_i and it's derivative $\partial f_i / \partial x_i$ below - **we'll need it soon**

$$f_i = h_{i1}x_1 + h_{i2}x_2 + \ldots + h_{ii}x_i + \ldots + h_{in}x_n - b_i = 0, \quad \frac{\partial f_i}{\partial x_i} = h_{ii}$$

Gauss-Seidel

Start with any $\mathbf{x} = [x_1 \; x_2 \; x_3 \ldots x_{n-1} \; x_n]^t$. Usually, $\mathbf{x} = \mathbf{0}$!

$$x_1 = \frac{b_1 - h_{12}x_2 - h_{13}x_3 \ldots\ldots - h_{1n}x_n}{h_{11}}$$

$$x_2 = \frac{b_2 - h_{21}x_1 - h_{23}x_3 \ldots\ldots - h_{2n}x_n}{h_{22}}$$

$$\vdots \quad \vdots \qquad \vdots$$

$$x_{n-1} = \frac{b_{n-1} - h_{n-1,1}x_1 - h_{n-1,2}x_2 \ldots\ldots - h_{n-1,n-2}x_{n-2} - h_{n-1,n}x_n}{h_{n-1,n-1}}$$

$$x_n = \frac{b_n - h_{n1}x_1 - h_{n2}x_2 - \ldots\ldots - h_{n,n-1}x_{n-1}}{h_{nn}}$$

Repeat until stopping criterion is satisfied

Gauss-Seidel

Note that these system can be rewritten as

$$x_1 = x_1 + \frac{b_1 - h_{11}x_1 - h_{12}x_2 - h_{13}x_3 \ldots\ldots - h_{1n}x_n}{h_{11}}$$

$$x_2 = x_2 + \frac{b_2 - h_{21}x_1 - h_{22}x_2 - h_{23}x_3 \ldots\ldots - h_{2n}x_n}{h_{22}}$$

$$\vdots \quad \vdots \qquad \vdots$$

$$x_{n-1} = x_{n-1} + \frac{b_{n-1} - h_{n-1,1}x_1 - h_{n-1,2}x_2 \ldots\ldots - h_{n-1,n-1}x_{n-1} - h_{n-1,n-2}x_{n-2} - h_{n-1,n}x_n}{h_{n-1,n-1}}$$

$$x_n = x_n + \frac{b_n - h_{n1}x_1 - h_{n2}x_2 - \ldots\ldots - h_{n,n-1}x_{n-1} - h_{n,n}x_n}{h_{nn}}$$

If you weren't focused till now, **start focusing**, because it's going to be very exciting from now on

Spot the iterative scheme $x_i = x_i + \Delta x_i$ here

Gauss-Seidel

Let's rewrite the last equation as

$$x_1 = x_1 - \frac{h_{11}x_1 + h_{12}x_2 + h_{13}x_3 \ldots\ldots + h_{1n}x_n - b_1}{h_{11}}$$

$$x_2 = x_2 - \frac{h_{21}x_1 + h_{22}x_2 + h_{23}x_3 \ldots\ldots + h_{2n}x_n - b_2}{h_{22}}$$

$$\vdots \quad \vdots \qquad \vdots$$

$$x_{n-1} = x_{n-1} - \frac{h_{n-1,1}x_1 + h_{n-1,2}x_2 \ldots\ldots + h_{n-1,n-1}x_{n-1} + h_{n-1,n-2}x_{n-2} + h_{n-1,n}x_n - b_{n-1}}{h_{n-1,n-1}}$$

$$x_n = x_n - \frac{h_{n1}x_1 + h_{n2}x_2 + \ldots\ldots + h_{n,n-1}x_{n-1} + h_{n,n}x_n - b_n}{h_{nn}}$$

Guys, these signs changes are EXTREMELY important for what we want to devise and show !!!

Δx

The n-th equation f_n

This is a pure N-R !!!

$$\frac{f_n}{f_n'}$$

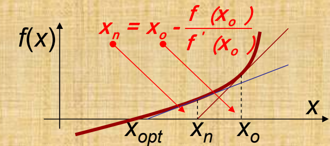
1st derivative f_n' of n-th equation in respect of x_n

Newton-Raphson

Fine, but how are the G-S & N-R methods related?

In order to **find** the **roots** of some *f*(**x**), or for what value of **x** the function *f*(**x**) will equal zero, Newton-Raphson method proposes the iterative scheme below

$$x = x - \frac{f(x)}{f'(x)}$$



In our case **$f(\mathbf{x})$ = g = Hx - b** and **f' = H,** and when working with matrices **division** is a **multiplication by an inverse**. With this in mind, the last equation becomes

$$\mathbf{x} = \mathbf{x} - \mathbf{H}^{-1}(\mathbf{Hx} - \mathbf{b}) = \mathbf{x} - \mathbf{H}^{-1}\mathbf{g}$$

Writing the equation above component-wise one would get the expressions on previous slide.

Well, comparing slides 25 & 26 one can say that Newton-Raphson method, even without knowing it, was used for iterative solving of linear system of equations **hidden in the Gauss-Seidel method** ! It seems **nobody has taken to much care about it?!**

The reason for such a "**neglect**" is due to the fact that **Newton**, i.e. **Newton-Raphson**, **method is tied with the root finding** in (a system of) NL equation(s) **so deeply and strongly up**

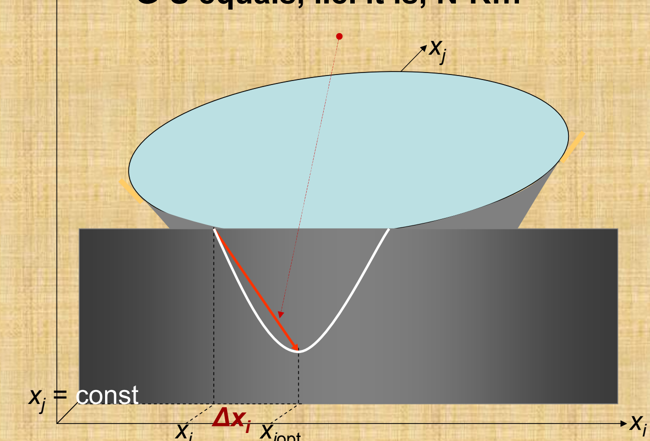
that **some books on linear algebra don't mention Newton-Raphson method whatsoever** e.g., the book from
R. Varga (on Matrix iterative methods), B. Noble, J. Dieudonne, S. Roman, K. Kuttler, …, and many other books … .

I can't believe it !

Gauss-Seidel's & Newton-Raphson's

geometries when optimizing along a single coordinate

G-S equals, i.e. it is, N-R!!!



Gauss-Seidel & Newton-Raphson

Let's show it analytically too!

$$J = 4x_1^2 + 2x_2^2 + x_3^2 - x_1x_2 + 2x_1x_3 - 2x_2x_3 - x_1 - x_2 - x_3$$

$\mathbf{x}_0 = [1\ 1\ 1]^t$, *and*

$J = \mathbf{x}^T\mathbf{H}\mathbf{x} - \mathbf{1}^T\mathbf{x}$, $\mathbf{H} = \begin{bmatrix} 4 & -0.5 & 1 \\ -0.5 & 2 & -1 \\ 1 & -1 & 1 \end{bmatrix}$

$J(x_1)$ for the 1^{st} iteration along x_1 is a parabola

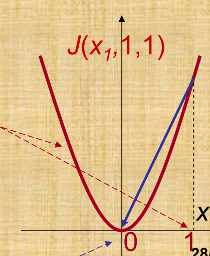
$$J(x_1) = 4x_1^2 + 2*1^2 + 1^2 - x_1*1 + 2x_1*1 - 2*1*1 - x_1 - 1 - 1$$

$$J(x_1) = 4x_1^2 - 1$$

$$f_1 = g_1 = \frac{\partial J}{\partial x_1} = 8x_1, \quad f_1' = H_1 = \frac{\partial^2 J}{\partial x_1^2} = 8$$

and the new x_1 will be

$$x_1 = x_1 - \frac{8x_1}{8} = 1 - \frac{8*1}{8} = 0$$



Gauss-Seidel & Newton-Raphson

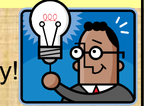
G-S procedure **must not update the variables in a cyclic order** (starting with the 1st eq., ending with the last one and repeating those sweeps).

A more **efficient way** is to update the variable having **the largest absolute value** of a **gradient vector**. This corresponds to selecting the variable with the **biggest absolute error**.

In **ML** this variable is called **the worst violator**. Such a choice ensures the fastest convergence to the minimum value of the (hyper)quadrics *J*.

Gauss-Seidel & Newton-Raphson

The key proposal of the seminar is an Expansion of G-S i.e., N-R, over The Subspace Spanned by *k* Worst Violators



Remind, in each iteration step G-S updates 1 variable only!

Idea, why don't we choose 2 worst violators/errors, or 3, or more, say *k*, and update them in a single N-R step?

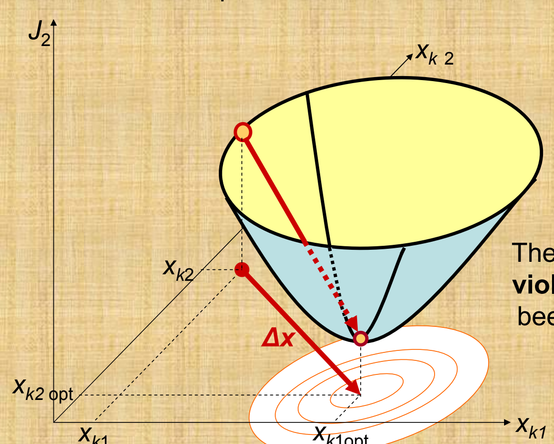
In other words – why not to perform the updates as given on the slide 26 but written for *k* coordinates below

$$\mathbf{x}_k = \mathbf{x}_k - \mathbf{H}_k^{-1}\mathbf{g}_k$$

Index *k* denotes that *k* worst violating variables are being updated. **In a geometric sense, we are finding the *k* *optimal* values of x defining the minimum of the elliptic paraboloid over a *k* dimensional subspace spanned by the *k* worst violators.**

Gauss-Seidel & Newton-Raphson

What's the geometry in a 2 dimensional subspace (***k* = 2**) of an *n*-dimensional quadrics when **H** is an **SPD matrix.**

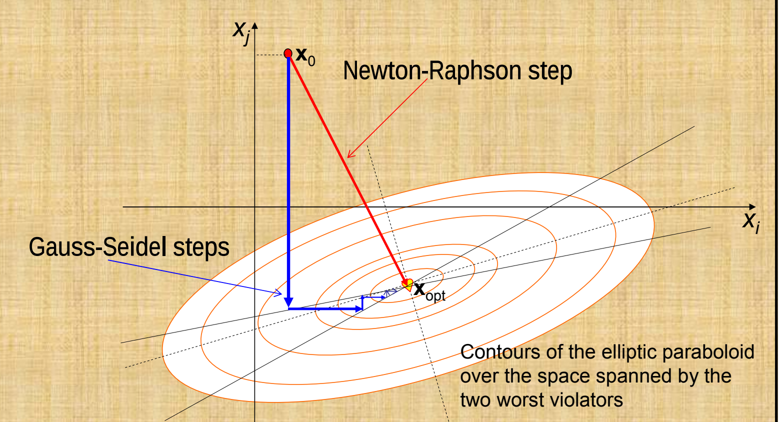


The **two worst violators** have been selected

***k* = 2**

Gauss-Seidel & Newton-Raphson

The difference between **G-S** and **N-R** for *k* = 2 can be readily seen in the figure below.



Gauss-Seidel & Newton-Raphson

H is **SPD** matrix and **the solution must be same for any** *k* we choose**, meaning accuracy must be 'equal' for any** chosen *k*.

Hence, **the basic issue when using *k* violators is the speed** or, **the CPU time needed to find the solution.**

Well, there is a **tradeoff** here:

An increase in *k* **reduces the number of sweeps through the datasets** but **the calculations are more complex because:**

first, **instead of finding a single worst violator with a cost of O{*n*}, we have to find *k* of them. This costs *k*O{*n*}** and,

second, **in each Newton-Raphson iteration step we must solve a system of *k* equations which costs ~*O*{*k*³}.**

At the moment we have only **experimental answers** which **support using more violators. How many, hard to tell right now, or, *k* = ?**

Novel Iterative Algorithm for Solving SPD System of Linear Equations - Pseudocode

$\mathbf{x} = 0$ (*but, any other good guess helps too*)

$\mathbf{g} = -\mathbf{b}$ (*if* $\mathbf{x} \neq 0$, $\mathbf{g} = \mathbf{H}\mathbf{x} - \mathbf{b}$) $\quad \mathbf{x}_{old} = \mathbf{x}$, $\quad err = 1$

$stopping = 0.001$ (*or, use other metrics here*)

while $err > stopping$

find indices vector **I** *of k worst violators in* **g** ⇐ Selection of a *k*-dim subspace over which the elliptic hyper-paraboloid J_k is minimized.

calculate $\mathbf{H}_k(\mathbf{I},\mathbf{I})$ *and* $\mathbf{g}_k(\mathbf{I})$

$\mathbf{x}_k(\mathbf{I}) = \mathbf{x}_k(\mathbf{I}) - \mathbf{H}_k^{-1}\mathbf{g}_k$ ⇐ **Only *k* components** of vector **x** given by **I** are updated.

ensure $\mathbf{x}_k(\mathbf{I})$ *satisfies given constraints*

$\mathbf{g} = \mathbf{g} + \mathbf{H}(:,\mathbf{I})\left(\mathbf{x}_k(\mathbf{I}) - \mathbf{x}_{k\,old}(\mathbf{I})\right)$ ⇐ **The whole gradient** (error, residual) vector **g** using only *k* columns of **H** given by **I** is updated.

$err = \dfrac{\|\mathbf{x} - \mathbf{x}_{old}\|}{\|\mathbf{x}\|}$ (*or, any other suitable metrics*)

Note that, if starting from **x=0**, we don't **ever** have to calculate the huge matrix **H** ☺ **!!!**

$\mathbf{x}_{old} = \mathbf{x}$

end

g = **r** (residual, error) in a classic numerical algebra literature!

Novel Iterative Algorithm for Solving SPD System of Linear Equations

Replacing the Calculation of an Inverse of a Matrix $\mathbf{H}_k$

Note that the update step

$$\mathbf{x}(\mathbf{I}) = \mathbf{x}(\mathbf{I}) - \mathbf{H}_k^{-1}\mathbf{g}_k$$

can also be rewritten as

$$\mathbf{x}(\mathbf{I}) = \mathbf{x}(\mathbf{I}) - \Delta\mathbf{x},$$

where $\Delta\mathbf{x}$ is a solution of the equation

$$\mathbf{H}_k\Delta\mathbf{x} = \mathbf{g}_k$$

and, because $\mathbf{H}_k$ is an **SPD** matrix,

$\Delta\mathbf{x}$ can be found **quicker by** using **Cholesky decomposition**. This may bring a 'significant' CPU time speedup (up to **3 times**).

Remark on the Successive Over-Relaxation (SOR)

- I am sure that the proposed model must also work with the SOR, which is given below. (However, I didn't check it and this claim asks for some investigations).

$$\mathbf{x}_k = \mathbf{x}_k - \omega\mathbf{H}_k^{-1}\mathbf{g}_k = \mathbf{x}_k - \omega\Delta\mathbf{x}$$

As always, the tricky part will be to pick up a correct value for ω!

Remark on the number of violators k used

My, the very first and thus the wildest, guess is to **pickup k according to the memory size**.

Use the highest possible k which still enables both the storing of the (k, k) matrix $\mathbf{H}_k$ and the calculation of the updates by performing $\mathbf{H}_k^{-1}\mathbf{g}_k$ within the memory .

Let's Christen The Algorithm

There is a good habit in both all times and all civilizations which goes as:

Soon after a baby's born give him/her the name

Being **The Very Humble Person** I'll name the approach proposed **just**

Kecman Algorithm

or **just**

Kecman Method

Let's Position the Proposed Method i.e., Let's Compare It with Other Algorithms

Well, depending upon k, it is somewhere
in between **Gauss-Seidel** and an **exact, one step, solution**
with the ***fleur*** of **Newton-Raphson**, and a ***scent*** of a **Block G-S** ⇨ meaning

For

$k = 1$, it's a pure Gauss-Seidel Method

$k = n$, it's a pure N-R having a, one step, exact solution which is, however, prohibitive when dealing with huge matrices **H**. $x=x-H^{-1}(Hx-b)=H^{-1}b$

$1 < k < n$, **the proposed method** is **original**, working in a k-dimensional subspace of k worst violators, finding there a **local** $\mathbf{x}_{k\_opt}$ **in a single step**, and iteratively approaching the **global** optimal solution $\mathbf{x}_{opt}$.

The Proposed Algorithm Falls Into the Category of Projection Methods such as

- Galerkin, Kaczmarz, Cimmino, G-S, **Block G-S** Richardson, Southwell (these are different methods, but sometimes 'equal', i.e., similar, too)
- Conjugate Gradient i.e. Krylov Subspace Methods
- ABS (named after Abaffy, Broyden and Spedicato), Row projection methods, Steepest Descent

Block G-S is different because it works with **predetermined blocks** & **it is cyclic => block G-S** algorithm is entirely different (but similar in the spirit). For its cyclic nature, I expect it to be much slower too. Note also that block G-S (usually) assumes knowing the whole matrix **H**

If interested, check Brezinski's book, "*Projection Methods for Systems of Equations*" North-Holland, **1997** & Hackbusch's, Iterative Solution of Large Sparse Systems of Equations, Springer, **1994**

Experimental Results

newtonraphson_iterat_lin_system.m, $\omega = 1$

are finally in order - for linear systems without constraints (the solution vector x_{opt} is dense) and for different both

sizes

from $n = 100$ to $n = 10{,}000$

&

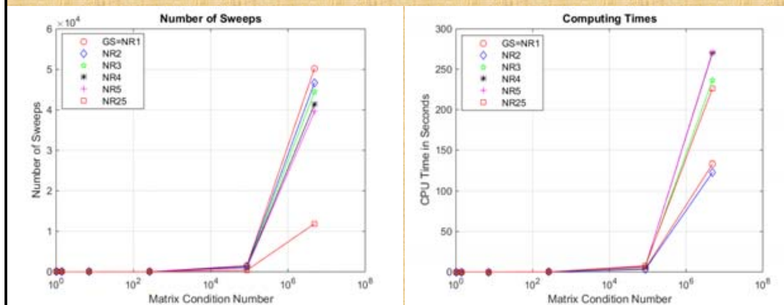
condition numbers

from ~2 to 10^8

Gauss-Seidel & Newton-Raphson i.e., **the Proposed Method in *k*-dimensional subspace of *k* worst violators (coordinates)**

Some experimental runs

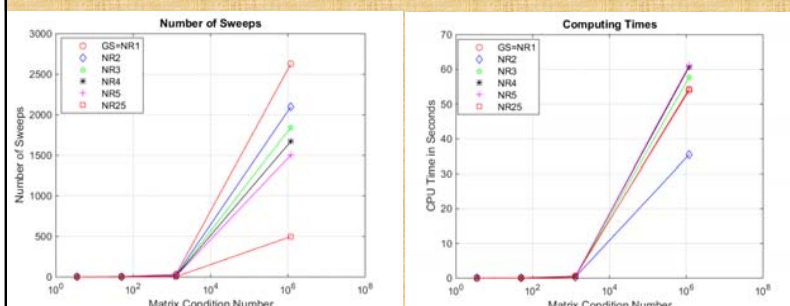
n = 100



Gauss-Seidel & Newton-Raphson i.e., **the Proposed Method in *k*-dimensional subspace of *k* worst violators (coordinates)**

Some experimental runs

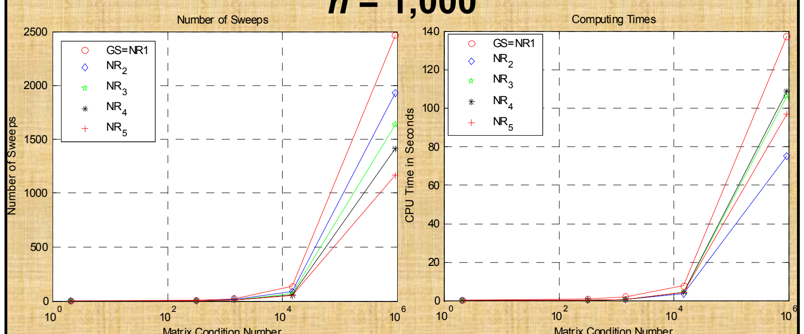
n = 500



Gauss-Seidel & Newton-Raphson i.e., **the Proposed Method in *k*-dimensional subspace of *k* worst violators (coordinates)**

Some experimental runs

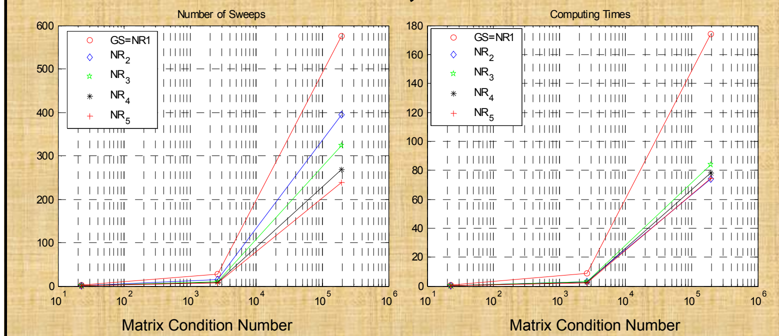
n = 1,000



Gauss-Seidel & Newton-Raphson i.e., **the Proposed Method in *k*-dimensional subspace of *k* worst violators (coordinates)**

Some experimental runs

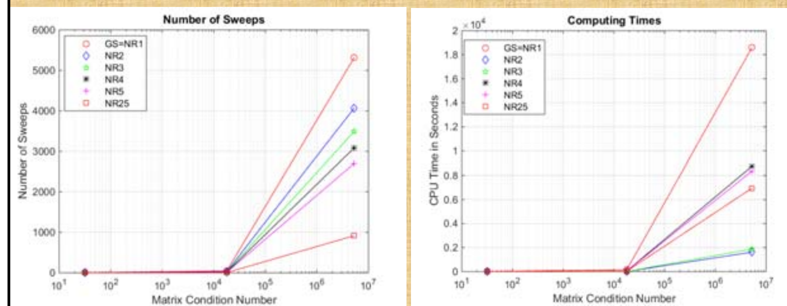
***n* = 2,500**



Gauss-Seidel & Newton-Raphson i.e., **the Proposed Method in *k*-dimensional subspace of *k* worst violators (coordinates)**

Some experimental runs

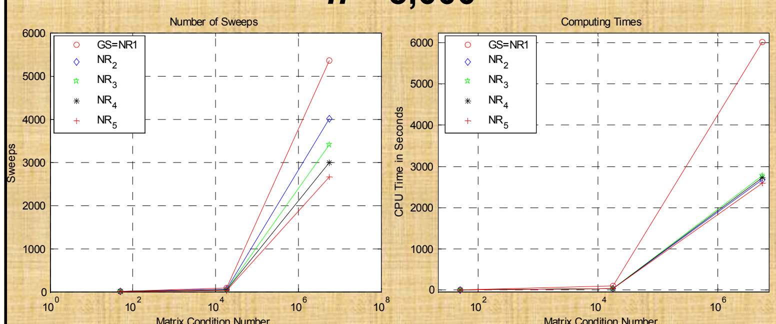
***n* = 4,000**



Gauss-Seidel & Newton-Raphson i.e., **the Proposed Method in *k*-dimensional subspace of *k* worst violators (coordinates)**

Some experimental runs

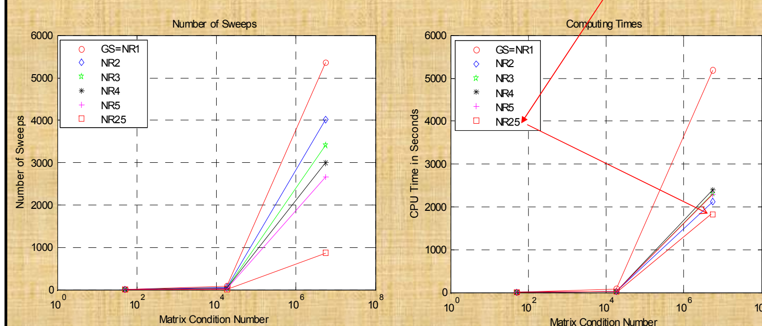
***n* = 5,000**



Gauss-Seidel & Newton-Raphson i.e., **the Proposed Method in *k*-dimensional subspace of *k* worst violators (coordinates)**

Some experimental runs. Note, here we have *k* = 25 too

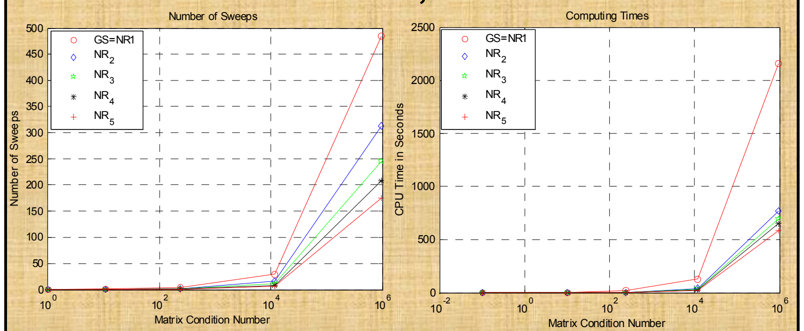
***n* = 5,000**



Gauss-Seidel & Newton-Raphson i.e., **the Proposed Method in *k*-dimensional subspace of *k* worst violators (coordinates)**

Some experimental runs

n = 10,000



Basic **Remarks** and **Comments**

1) Solving LARGE systems is possible only with iterative algorithms
2) All the results shown are WITHOUT CONSTRAINTS, hence, **no sparseness** in a solution vector **x** whatsoever
3) Results may be different for SVM tasks or, when solving any other linear system of equations with constraints
4) **It is highly recommended to use more violators. Value of *k* is ???**
5) The bigger the system, the higher the speed up
6) The proposed method **using *k* worst violators**, is the strongest candidate method for HUGE **SVM classification** tasks, i.e. HUGE **linear system of equations with nonnegativity, and other, constraints**
7) **Whether the last claim is true** is under an investigation for **DL2 SVMs** by LAAL's PhD student **Ljiljana Zigic**
8) **Warning** – implementation of a proposed method when having constraints is **a particularly challenging research task because**

There Is No King's Way to Algorithms Implementations Either

Basic **Remarks** and **Comments**

9) I didn't check it, but I believe that **the algorithm on slide 35 converges whenever the G-S does**. Hence, the question **whether a symmetry and positive definiteness is required** for a proposed method **should be investigated** in more details.

10) Sure, the proposed algorithm shown here for a linear system can also (but, for issues of convergence, with a lot of caution) be applied for a system of nonlinear equations as follows:

do the Gauss-Newton algorithm along the lines of the pseudocode for the proposed method on the slide 35, meaning don't build Jacobian for all F(x). Go in chunks of *k*. It may be faster, but be aware that the proposed method converges because our linear system is a symmetric positive definite one.

With nonlinear systems there will be many additional issues and convergence is generally not quite guaranteed!

Wait, wait, wait!

- **This is not the end of the story yet !!!**
- This was just what **I** was doing lately. Vojo's stuff! So, forget it !
- **The Very Big and The True Story of The Day Is**
  - each of you is facing some problem you have to solve
  - look at it, see what is in the very root of your problem
  - find out was there anybody else who was facing it, or who was doing similar stuff (remind, this is a heavy digging)
- If there is nobody, check it twice. After checking it 2nd time, check it one more time (I mean, check it indeed) just to be sure. (As for me, I believe, **your problem, possibly disguised, has already been solved**).
- If, after all the **thorough** checking, there was nobody who was entertaining your problem, **think about your problem again**
  - Is it right?
    - Is it properly posed?
      - Is it novel?
        - » Is your advisor 'pleased' about your PhD topic?
- **If All the Answers are Positive You Are at the Blessed Spot!**

The Whole Big World is Waiting for Your Solutions

and **The Fame Is in Front of You!!!**

You won’t believe it, but this is the **last** slide !!!

Thanks!