I ♥ my Ex
Accelerating Nonnegative Matrix Factorization Algorithms using Extrapolation
A talk with no theory at all

Andersen Ang & Nicolas Gillis
Mathématique et recherche opérationnelle, UMONS, Belgium
Homepage: angms.science

July 4, 2018
ISMP2018 Bordeaux, France
This talk, for (serious) optimization theorist:

theory content $= \epsilon$

For (applied) machine learner:

application content $= \epsilon$

where $\epsilon \downarrow 0$. 
Non-negative Matrix Factorization (NMF)

Given

- Input matrix: \( X \in \mathbb{R}^{m \times n} \)
- A positive integer \( r \)

Find the matrices \( W \in \mathbb{R}^{m \times r} \), \( H \in \mathbb{R}^{r \times n} \) such that

\[ X = WH. \]

Everything is non-negative

Notational note: we use \( WH \) instead of \( WH^\top \)
The only 2 pages with positive amount of content (1/2)

**Theoretical side**

- $r$ is factorization rank / non-negative rank $\text{rank}^+ := \text{smallest } r \text{ such that}$
  
  $$X = \sum_{i=1}^{r} X_i, \quad X_i \text{ rank-1 and non-negative}$$

- How to find $\text{rank}^+$ / estimate $\text{rank}^+$ / bounding $\text{rank}^+$, e.g. $\text{rank}_{\text{psd}}(X) \leq \text{rank}^+(X)$
- Combinatorial optimization, extended formulations
- Log-rank Conjecture, communication system
- Vavasis: NMF is NP-hard to compute 😞 😞 😞
- Arora-Ge-Kannan-Moitra: $(mn)^{O(2^r r^2)}$, $(2^r nm)^{O(r^2)}$, $(nm)^{o(r^2)}$?
- Donoho-Stodden: Separability
- Separable NMF: pure pixel, anchord words, extreme ray / extreme point generators
- Gillis-Vavasis: Fast and Robust methods for Separable NMF
- Lin-Ma-Li-Chi-Ambikapathi: what if no separability????
- **Now**: Minimum Volume NMF / separable NMF with hidden generating vertices

↑ theoretical picture of NMF from Fu-Huang-Sidiropoulos-Ma2018
Application sides

- **Area that I don’t really care**: hyperspectral imaging, chemical spectrum unmixing, information retrieval, text mining, image segmentation, collaborative filtering, probabilistic models, face recognition, cognitive radio DNA-processing

- **Area that I ♥**: Music signal processing, Brain-wave signal processing / Brain-computer Interface / Mind-control technology ♥♥♥

Artwork preservation, picture from Grabowski-Masarczyk-Glomb-Mendys2018
Optimization problem

Given \( X \in \mathbb{R}^{m \times n} \), integer \( r \), solve

\[
[W \ H] = \arg \min_{W \geq 0, H \geq 0} \|X - WH\|_F^2
\]

- \( W \in \mathbb{R}^{m \times r} \), \( H \in \mathbb{R}^{r \times n} \) : optimization variables
- \( \geq \) is element-wise, not the positive semidefinite thing!
- a non-convex, NP-hard, ill-posed problem 😞 😞 😞
- Assume \( r \) known and low rank ❤
- Initialize \( W, H \) by random (not going to talk about initialization)
Standard solution framework – 2-Block Coordinate Descent

Problem: given \( (X, r) \), solve

\[
\min_{\substack{W \geq 0 \\ H \geq 0}} \|X - WH\|_F^2
\]

Algorithm 1 BCD framework for NMF

Input: \( X \in \mathbb{R}^{m \times n}, r \in \mathbb{N}_+ \), an initialization \( W \in \mathbb{R}^{m \times r}_+, H \in \mathbb{R}^{r \times n}_+ \)

Output: \( W \) and \( H \)

1: for \( k = 1, 2, \ldots \) do

2: Update[\( W \)].

\[ \text{e.g. exact coordinate minimization } W \leftarrow \arg \min_{W \geq 0} \|X - WH\|_F^2. \]

3: Update[\( H \)].

\[ \text{e.g. exact coordinate minimization } H \leftarrow \arg \min_{H \geq 0} \|X - WH\|_F^2. \]

4: end for

\[
\|X - WH\|_F^2 = \|X^\top - H^\top W^\top\|_F^2
\]

\[ \rightarrow \text{the exact coordinate minimization subproblems are symmetric} \]

\[ \rightarrow \text{so we can focus on one variable, says } H. \] (update of \( W \) is similar)
Variations of BCD

The BCD update

\[
\text{Update}[\mathbf{H}] : \quad \mathbf{H} \leftarrow \arg \min_{\mathbf{H} \geq 0} \| \mathbf{X} - \mathbf{W} \mathbf{H} \|_F^2,
\]

has the following variations

1. Block partitions: on how coordinate is defined
2. Index selection (indexing): on how coordinate is selected
3. Update scheme: on how coordinate is updated
4. Other variations
Variations of BCD: 1. Block partitions

**For example**: block coordinate = vector

- sub-divide $\mathbf{H}$ into $r$ blocks of row vector

![Diagram of block partition]

**Algorithm 2** $\text{Update}(\mathbf{H})$: vector-wise

1. Pick a index $i_k \in [1, 2, ..., r]$
2. $\text{Update}[\mathbf{H}(i_k, :)]$

Or:

- treat whole $\mathbf{H}$ as one coordinate
- treat each scalar $\mathbf{H}_{i,j}$ as one coordinate (but then too many loops!)
Variations of BCD: 2. Indexing

For example, simple cyclic on vector: \[ i_k = \text{mod}(k - 1, r) + 1 \]

- \[ i_k = 1, 2, 3, 1, 2, 3, ... \]
  - 1 cycle
  - 1 cycle

Algorithm 3 Update(H), cyclic vector-wise

1: \[ i_k = \text{mod}(k - 1, r) + 1 \]
2: Update[H(i_k, :)]

Other variants

- Random, random shuffle (random cyclic)
- Nesterov’s ACDM and generalization by Lee-Sidford 2013
- Greedy: Gauss-Southwell – by a ”score” – pick index with highest score
  - higher per iteration cost (\( \therefore \) compute scores for each index)
  - but there are some tricks to bypass the score computation (for NMF)
Variations of BCD : 3. Update scheme

For example: simple coordinate minimization with exact model – given $i_k$

\[
\text{Update}[H(i_k, :)] : H(i_k, :) = \arg \min_{H(i_k, :) \geq 0} \left( \| X - W(:, i_k)H(i_k, :) \|_F^2 + f(H(i_k, :)) \right)
\]

- using original exact model $f$ and solve it exactly/approximately

**Algorithm 4 Update($H$)**

1: $i_k = \text{mod}(k - 1, r) + 1$
2: $\text{Update}[H(i_k, :) ] : H(i_k, :) = \arg \min_{H(i_k, :) \geq 0} f(H(i_k, :))$

Other variants: inexact model – not working exactly on $f$

- Proximal point:

\[
\min_{H(i,:) \geq 0} f(H(i,:)) + \frac{1}{2\alpha} \| H(i,:) - H(i,:)^- \|_2^2,
\]

- Linearized Proximal point:

\[
\min_{H(i,:) \geq 0} f(H(i,:)^-) + \langle \nabla_i f(H(i,:)), H(i,:) - H(i,:)^- \rangle + \frac{1}{2\alpha} \| H(i,:) - H(i,:)^- \|_2^2,
\]

- Frank-Wolfe / Conditional gradient:

\[
\min_{H(i,:) \geq 0} f(H(i,:)) + \langle \nabla_i f(H(i,:)), H(i,:) - H(i,:)^- \rangle,
\]

and then solve it exactly/approximately
Avoiding repeated computation: HALS and A-HALS

For a 2-variable problem, there is even more indexing possibilities:

**HALS**

\[
\begin{align*}
W(:, 1) & \rightarrow H(1, :) \\
& \rightarrow W(:, 2) \rightarrow H(2, :) \rightarrow W(:, 3) \rightarrow H(3, :) \rightarrow ...
\end{align*}
\]

**A-HALS**

\[
\begin{align*}
W(:, 1) & \rightarrow W(:, 2) \rightarrow W(:, 3) \rightarrow \underbrace{H(1, :)}_{\text{several times}} \rightarrow H(2, :) \rightarrow H(3, :) \rightarrow ...
\end{align*}
\]

A-HALS is better: avoid repeated computations of certain parameters by reusing them.
Avoiding repeated computation: HALS and A-HALS

Suppose we use projected gradient descent, we have

\[ W = W - t (WHH^T - XH^T), \quad H = H - t (W^T WH - W^T X) \]

A-HALS avoids repeated computations of

\[ HH^T_{(2n-1)m^2}, \quad XH^T_{(2n-1)mr}, \quad W^T W_{(2r-1)m^2}, \quad W^T X_{(2m-1)rn} \]

these terms can be pre-computed first and reuse several times

\[ \Rightarrow \] efficiency improvement, significant if big big: should always use A-HALS!

\[ \text{Algorithm 5 HALS} \]

1: \( W(:, 1) = W(:, 1) - t(W(:, 1)HH^T - XH^T) \)
2: \( H(1, :) = H(1, :) - t(W^T WH(1, :) - W^T X) \)
3: \( W(:, 2) = W(:, 2) - t(W(:, 2)HH^T - XH^T) \)
4: \( H(2, :) = H(2, :) - t(W^T WH(2, :) - W^T X) \)
5: \( W(:, 3) = W(:, 3) - t(W(:, 3)HH^T - XH^T) \)
6: \( H(3, :) = H(3, :) - t(W^T WH(3, :) - W^T X) \)
7: ...

\[ \text{Algorithm 6 A-HALS} \]

1: \( W(:, 1) = W(:, 1) - t(W(:, 1)HH^T - XH^T) \)
2: \( W(:, 2) = W(:, 2) - t(W(:, 2)HH^T - XH^T) \)
3: \( W(:, 3) = W(:, 3) - t(W(:, 3)HH^T - XH^T) \)
4: \( H(1, :) = H(1, :) - t(W^T WH(1, :) - W^T X) \)
5: \( H(2, :) = H(2, :) - t(W^T WH(2, :) - W^T X) \)
6: \( H(3, :) = H(3, :) - t(W^T WH(3, :) - W^T X) \)
7: ...

†Projection step not shown here
Problem: given \((X, r)\), solve

\[
\min_{W \geq 0, H \geq 0} \|X - WH\|_F^2
\]

**Algorithm 7 BCD framework for NMF**

**Input:** \(X \in \mathbb{R}^{m \times n}, r \in \mathbb{N}_+, \) an initialization \(W \in \mathbb{R}^{m \times r}, H \in \mathbb{R}^{r \times n}.\)

**Output:** \(W, H.\)

1: **for** \(k = 1, 2, \ldots\) **do**
2: Update\([W]\) : \(W \leftarrow \arg \min_{W \geq 0} \|X - WH\|_F^2.\)
3: Update\([H]\) : \(H \leftarrow \arg \min_{H \geq 0} \|X - WH\|_F^2.\)
4: **end for**

We consider:

- matrix block with exact model
- cyclic indexing
- exact coordinate minimization
Last ingredient: acceleration via extrapolation

Recall: for one-variable problem $$\min_{x \in C} f(x)$$, at step $$k$$

Update $$x_{k+1} = \text{Update}[x_k]$$

Linear extrapolation $$x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = \text{Extrapolate}[x_{k+1}, x_k]$$

To be specific:

GD Update $$x_{k+1} = x_k - t_k \nabla f(x_k)$$.

Linear extrapolation $$x_{k+1} = x_k - t_k \nabla f(x_k), \quad y_{k+1} = x_{k+1} + \beta_k (x_{k+1} - x_k)$$. i.e. $$\text{Extrapolate}[x_{k+1}, x_k]$$ is modeled by $$\beta_k := \text{extrapolation parameter}$$ a single number.
Why extrapolation? Because gradient descent zig-zag

Fact: standard gradient descent has zig-zag behavior → slow
Example: moving along a long narrow valley

What machine learning people do to counter zig-zag

Do tricks on step size: don’t move with step size $t$ but $\frac{t}{\text{damping factor}}$

The length of each pink segment is shorter than that of the corresponding red segment $\implies$ the points on pink segment is closer to the axis $y = 0$ $\implies$ gradient vector there have stronger $x$-component $\implies$ less oscillation along the $y$-direction!

This is the idea behind **AdaGrad** and **AdaDelta** from machine learning community: slow down step size when you see zig-zag (trace of the objective function appears to plateau)
What optimization people do to counter zig-zag

Do tricks on direction: by extrapolation with momentum

Idea: apply extrapolation
Extrapolate = add gradient history

(1) if gradients in consecutive steps consistently point in same direction
⇒ extrapolate = accelerate

(2) if gradients in consecutive steps oscillates (continuously changing direction)
⇒ extrapolate = damp oscillation, which is also acceleration

Figure shows the traces of point decomposed into $x$-component and $y$-component. It can be observed that the $x$-components have consistent direction while $y$-components keep changing the direction.
\[ x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k). \]
\[ x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k (x_{k+1} - x_k). \]
\[ x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k). \]
The geometry of extrapolation (4/7)

\[ x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k). \]
$$x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k).$$
Now consider $x_{k+1} - y_k$, $x_{k+2} - x_{x+1}$ and $x_{k+2} - y_{k+1}$
We always have

$$\angle(x_{k+1} - y_k) \geq \angle(x_{k+2} - x_{k+1}) \geq \angle(x_{k+2} - y_{k+1})$$

i.e. the direction of the last step is **in between** the directions of previous two gradient steps: zig-zag effect is reduced!
The details of Nesterov’s acceleration

1. For **single-variable convex** function,

\[ \beta_k = \frac{1 - \alpha_k}{\alpha_{k+1}}, \quad \alpha_{k+1} = \frac{1 + \sqrt{1 + 4\alpha_k^2}}{2}, \quad \alpha_1 \in (0, 1) \]

2. For **single-variable smooth strongly convex** function with *conditional number* \( Q \),

\[ \beta_k = \frac{1 - \sqrt{Q}}{1 + \sqrt{Q}}, \quad \text{where} \quad Q = \frac{L}{\mu} = \frac{\text{Smoothness parameter}}{\text{Strong convexity parameter}} \]

With convergence improvement: from \( \mathcal{O}(Q \log \frac{1}{\epsilon}) \) to \( \mathcal{O}(\sqrt{Q} \log \frac{1}{\epsilon}) \)

Conclusion: Nesterov’s acceleration has a close form update for \( \beta_k \).
Our case

Problem: given \((X, r)\), solve

\[
\min_{W \geq 0, H \geq 0} \|X - WH\|_F^2
\]

1. The problem has **two variables** \(W, H\)
2. Two variables: **nonconvex**
3. Non-convex problem \(\implies\) no strong convexity parameter \(\mu\)
4. \(\beta_k = \frac{1 - \sqrt{Q}}{1 + \sqrt{Q}}\) cannot be used

Our case: need to find a way (close form / no close form) update of \(\beta_k\)
Why gradient descent (GD) zig-zag?
GD step is **local**: only using **local** info. to move.
GD only descent to the local minimum.
Such direction does not point to the global minimum $x^*_\text{global}$

**GD is greedy**
It move greedily w.r.t. current local info.
greedy method $\implies$ no guarantee will move to $x^*_\text{global}$.

**GD is "descent"** $f(x_{k+1}) < f(x_k) \forall k$.

What about the non-negativity constraints?
Well handled by projection.
Remark (2/2)

Extrapolation is not greedy nor descent

- GD is locally optimal/greedy $\implies$ extrapolation may $\uparrow$ objective value
- Extrapolation = a risky move

Acceleration comes from doing that risky move:

"sacrifice the increases of objective value now for the better future"

Actually also sacrifice robustness: accelerated gradient is not stable to noise! (Devolder-Glineur-Nesterov14)
Details of our extrapolation

The update-then-extrapolate step for the non-convex two-variable problem

\[
\begin{align*}
\text{On } W \quad & \begin{cases}
\text{Update} & W_{\text{new}} = \text{Update}[Y_{\text{old}}, H_{\text{old}}] \\
\text{Extrapolate} & Y_{\text{new}} = W_{\text{new}} + \beta_k^W (W_{\text{new}} - W_{\text{old}}) \\
\end{cases} \\
\text{On } H \quad & \begin{cases}
\text{Update} & H_{\text{new}} = \text{Update}[W_{\text{new}}, G_{\text{old}}] \\
\text{Extrapolate} & G_{\text{new}} = H_{\text{new}} + \beta_k^H (H_{\text{new}} - H_{\text{old}}) \\
\end{cases}
\end{align*}
\]

The key $\beta_k$

- If $\beta \in (0, 1)$: extrapolation, doing risky step
- If $\beta = 1$: doing very risky extrapolation
- If $\beta = 0$: no extrapolation, reduce to standard gradient descent
- Cannot use line search\(^\dagger\) to find $\beta$: experimentally found $\beta$ very close to 0 — very minor extrapolation, effectively doing nothing
- $\beta$ has to be smaller than 1 (as stated in the cvx. case, proved)

Two-variable non-convex \(\rightarrow\) too hard

\(\rightarrow\) no structural formula on $\beta_k$ \(\rightarrow\) ad hoc heuristics

\(\therefore\) No theory : science \(\rightarrow\) engineering

\(\dagger\)Line search to minimize the objective function directly — performed before the update
Details of our extrapolation: update of $\beta_k$

Landscape of variable at each iteration is different, so dynamically update

Algorithm 8 An ad hoc heuristics of Update[$\beta_k$] in the line search style$^\dagger$

Input: Parameters $1 < \tilde{\gamma} < \gamma < \eta$, an initialization $\beta_1 \in (0, 1)$
Output: $\beta_k$

1: Set $\bar{\beta} = 1$
2: if the error decreases at iteration $k$ then
3:     Increase $\beta_{k+1}$: $\beta_{k+1} = \min(\bar{\beta}, \gamma \beta_k)$
4:     Increase $\bar{\beta}$: $\bar{\beta} = \min(1, \tilde{\gamma} \bar{\beta})$
5: else
6:     Decrease $\beta_{k+1}$: $\beta_{k+1} = \beta_k / \eta$
7:     Set $\bar{\beta} = \beta_{k-1}$
8: end if

Meaning:
- Go further / "Speed up" when suitable - if error $\downarrow$: more ambitious, make $\beta \uparrow$, take more risk
- Go back / "Slow down" when not suitable - if error $\uparrow$: less ambitious, make $\beta \downarrow$, take less risk

$^\dagger$Line search after updates of $W$ and $H$ – performed after the update!
The algorithm of Accelerated NMF using extrapolation

Input: $X$, initialization $W, H$, parameters $hp \in \{1, 2, 3\}$ (extrapolation/projection of $H$).

1: $W_y = W$;
2: $H_y = H$;
3: $e(0) = ||X - WH||_F$.
4: for $k = 1, 2, \ldots$ do
5: \hspace{1em} Compute $H_n$ by $\min_{H_n \geq 0} ||X - W_y H_n||_F^2$ using $H_y$ as initial iterate.
6: \hspace{1em} if $hp \geq 2$ then
7: \hspace{2em} Extrapolate: $H_y = H_n + \beta_k (H_n - H)$.
8: \hspace{1em} end if
9: \hspace{1em} if $hp = 3$ then
10: \hspace{2em} Project: $H_y = \max(0, H_y)$.
11: \hspace{1em} end if
12: \hspace{1em} Compute $W_n$ by $\min_{W_n \geq 0} ||X - W_n H_y||_F^2$ using $W_y$ as initial iterate.
13: \hspace{1em} Extrapolate: $W_y = W_n + \beta_k (W_n - W)$.
14: \hspace{1em} if $hp = 1$ then
15: \hspace{2em} Extrapolate: $H_y = H_n + \beta_k (H_n - H)$.
16: \hspace{1em} end if
17: \hspace{1em} Compute the error: $e(k) = ||X - W_n H_y||_F$.
18: \hspace{1em} if $e(k) > e(k - 1)$ then
19: \hspace{2em} Restart: $H_y = H_n; W_y = W_n$.
20: \hspace{1em} else
21: \hspace{2em} $H = H_n; W = W_n$.
22: \hspace{1em} end if
23: \hspace{1em} end for

Notation: $W_n$ - normal variable, $W_y$ – extrapolate variable, $W$ – previous $W_n$ (⊙⊙⊙) hard to read !!
The algorithm if $h_p = 1$, simplified

**Input:** $X$, initialization $W, H$

**Output:** $W, H$

1. $W_y = W; H_y = H; e(0) = \|X - WH\|_F$.
2. **for** $k = 1, 2, \ldots$ **do**
3. Update[$H_n$] w.r.t. $H_n \geq 0$ with $X, W_y, H_n$ using $H_y$ as initial iterate. 
4. Update[$W_n$] w.r.t. $W_n \geq 0$ with $X, W_n, H_y$ using $W_y$ as initial iterate. 
5. Extrapolate[$W_y$]: $W_y = W_n + \beta_k(W_n - W)$.
6. Extrapolate[$H_y$]: $H_y = H_n + \beta_k(H_n - H)$.
7. Compute error: $e(k) = \|X - W_n H_y\|_F$.
8. **if** $e(k) > e(k - 1)$ **then**
9.     Restart: $H_y = H_n; W_y = W_n$.
10. **else**
11.     $H = H_n; W = W_n$.
12. **end if**
13. **end for**

"Up, Up, Ex, Ex"
The algorithm if $h_p = 2$, simplified

**Input:** $X$, initialization $W, H$

**Output:** $W, H$

1: $W_y = W; H_y = H; e(0) = ||X - WH||_F.$

2: for $k = 1, 2, \ldots$ do

3: Update[$H_n$] w.r.t. $H_n \geq 0$ with $X, W_y, H_n$ using $H_y$ as initial iterate.

4: Extrapolate[$H_y$] : $H_y = H_n + \beta_k(H_n - H)$.

5: Update[$W_n$] wr.t. $W_n \geq 0$ with $X, W_n, H_y$ using $W_y$ as initial iterate.

6: Extrapolate[$W_y$] : $W_y = W_n + \beta_k(W_n - W)$.

7: Compute the error: $e(k) = ||X - W_n H_y||_F.$

8: if $e(k) > e(k - 1)$ then

9: Restart: $H_y = H_n; W_y = W_n.$

10: else

11: $H = H_n; W = W_n.$

12: end if

13: end for

"Up, Ex, Up, Ex"
The algorithm if $h_p = 3$, simplified

**Input:** $X$, initialization $W, H$

**Output:** $W, H$

1. $W_y = W; H_y = H; e(0) = ||X - WH||_F.$
2. for $k = 1, 2, \ldots$ do
3.   Update[$H_n$] w.r.t. $H_n \geq 0$ with $X, W_y, H_n$ using $H_y$ as initial iterate.
4.   Extrapolate[$H_y$] : $H_y = H_n + \beta_k(H_n - H)$.
5.   Project: $H_y = \max(0, H_y)$.
6.   Update[$W_n$] wr.t. $W_n \geq 0$ with $X, W_n, H_y$ using $W_y$ as initial iterate.
7.   Extrapolate[$W_y$] : $W_y = W_n + \beta_k(W_n - W)$.
8.   Compute the error: $e(k) = ||X - W_n H_y||_F$.
9.   if $e(k) > e(k - 1)$ then
10.      Restart: $H_y = H_n; W_y = W_n$.
11.   else
12.      $H = H; W = W_n$.
13.   end if
14. end for

"Up, Ex, Pro, Up, Ex"
Summary and notes

1. Extrapolation may break the NN ($\geq 0$) constraint:

<table>
<thead>
<tr>
<th>$hp = 1$</th>
<th>$hp = 2$</th>
<th>$hp = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Up-Up-Ex-Ex)</td>
<td>(Up-Ex-Up-Ex)</td>
<td>(Up-Ex-Pro-Up-Ex)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Up-Up-Ex-Ex Step</th>
<th>NN?</th>
<th>Step</th>
<th>NN?</th>
<th>Step</th>
<th>NN?</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update[$H_n$]</td>
<td>Y</td>
<td>Update[$H_n$]</td>
<td>Y</td>
<td>Update[$H_n$]</td>
<td>Y</td>
<td>Update[$H_n$]</td>
</tr>
<tr>
<td>Update[$W_n$]</td>
<td>Y</td>
<td>Extrapol[$H_y$]</td>
<td>N</td>
<td>Extrapol[$H_y$]</td>
<td>N</td>
<td>Project[$H_y$]</td>
</tr>
</tbody>
</table>

2. Update using matrix with negative values:

- Update[$H_n$] w.r.t. $H_n \geq 0$ with $X$, $W_y$, $H_n$ using $H_y$ as initial iterate.
- Update[$W_n$] w.r.t. $W_n \geq 0$ with $X$, $W_n$, $H_y$ using $W_y$ as initial iterate.

3. Restart using $e(k)$ as $\|X - W_n H_y\|_F$ not $\|X - W_n H_n\|_F$

   Why: (i) $W_n$ was updated according to $H_y$
   (ii) it gives the algorithm some degrees of freedom to possibly increase the objective function
   (iii) computationally cheaper, as compute $\|X - W_n H_n\|_F$ need $O(mnr)$ operations instead of $O(mr^2)$ by reusing previous computed terms: $\|X - WH\|_F^2 = \|X\|_F^2 - 2 \langle W, XH^\top \rangle + \langle W^\top W, HH^\top \rangle$
Experiments

Notations
- A-HALS: vector-wise update, compute approximate solution
- ANLS: subproblem are solved exactly using active-set methods
- E: extrapolation

Set up
- Average error over 10 trials
- $\mathbf{W}, \mathbf{H}, \mathbf{X}$ randomly generated $\sim \mathcal{U}[0, 1]$, $m = n = 200$, $r = 20$
- Error comparisons: using lowest relative error $e_{\min}$ across all algorithms, at step $k$,

$$E(k) = \frac{\| \mathbf{X} - \mathbf{W}^k \mathbf{H}^k \|_F}{\| \mathbf{X} \|_F} - e_{\min}$$

- Extrapolation parameter $\beta_0 = [0.25, 0.5, 0.75]$
- $\eta_0 = [1.5, 2, 3]$
- $\gamma, \bar{\gamma} = [1.01, 1.005], [1.05, 1.01], [1.1, 1.05]$
- For display: only best and worst to illustrate sensitivity (for $hp = 2$)
Results on ANLS

Low-rank synthetic data

Image data

Fast conclusion: no matter what, Ex always wins.
Results on A-HALS

Image data

Text data

Fast conclusion: no matter what, Ex always wins.
On computational time (speed)

Low rank synthetic data

Full rank synthetic data

Fast conclusion: no matter what, Ex always wins.
## Overall results: the Ex wins!

<table>
<thead>
<tr>
<th>Method</th>
<th>Data</th>
<th>Ex wins?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-HALS</td>
<td>Low rank synthetic data</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Full rank synthetic data</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Dense Image data†</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Sparse text data#</td>
<td>YES</td>
</tr>
<tr>
<td>ANLS</td>
<td>Low rank synthetic data</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Full rank synthetic data</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Dense Image data†</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Sparse text data#</td>
<td>YES</td>
</tr>
</tbody>
</table>

† ORL, Umist, CBCL, Frey


Conclusion: no matter what method XXX, E-XXX > XXX.
Questions: what about between E-XXX vs E-YYY?
E-ANLS vs E-A-HALS

Summary / findings:
- Low rank synthetic data: E-ANLS $\gg$ everything
- Dense data: E-A-HALS $\approx$ E-ANLS, although A-HALS $>$ ANLS
- Sparse data: E-A-HALS $\gg$ everything

Notations
- "$>$" means better (it wins)
- "$\gg$" means significantly better (it wins)
- "$\approx$" means similar, (draw)

Don’t trust me? Go https://arxiv.org/abs/1805.06604, try the code!
Accelerating NMF using extrapolation with **ac hoc heuristic line search style** Update $[\beta_k]$  

No matter what, I ♥ my Ex*

**Future work**
- Theory on systemic $\beta_k$ selection : how $(>)$ ????
- Theory on convergence : how $(>)$ ????
- From $\| \cdot \|_F$ to $\| \cdot \|_1$ to $\| \cdot \|_p$ to $D_\beta(\cdot)$ for applications

All dumb things go to me, all glory goes to my supervisor

**Figure:** Nicolas Gillis

END OF PRESENTATION : slide, MATLAB code** in angms.science

* If you want speed and OK to lost somethings  
**I know Julia good, but no time to learn it yet.

Acknowledgment : European Research Council Grant #679515