NMFLibrary

An Open-Source Toolbox for Non-negative Matrix Factorization

Hiroyuki Kasai

Waseda University, Tokyo, Japan

GitHub MATLAB® python™

Existing libraries of NMF

Name	language	Features	Latest rele.
NIMFA [1]	Python		Aug. 2019
nmftool [2]	MATLAB	biological data mining	Feb. 2013
NMF Toolbox [3]	Python & MATLAB	audio signal processing	Aug. 2019
NMF package [4]	R	solvers and initialization	June 2020
libNMF package [5]	C	solvers and initialization	Jan. 2011
NMF Book [6]	MATLAB	Chapter codes, solvers, models in book [7]	June 2022
NMFLibrary	MATLAB	models and solvers	July 2022

[1] https://nimfa.biolab.si/

[2] `https://sites.google.com/site/nmftool/`

[3] `https://www.audiolabs-erlangen.de/resources/MIR/NMFtoolbox/`

[4] `https://github.com/renozao/NMF`

[5] `http://rlcta.univie.ac.at/software/`

[6] `https://gitlab.com/ngillis/nmfbook`

[7] Nonnegative Matrix Factorization (SIAM) by Nicolas Gillis, `https://gitlab.com/ngillis/nmfbook`

Why is NMFLibrary needed?

- Need an evaluation framework to test and compare solvers at hand for fair and comprehensive experiments, because
  - Performances of solver solvers are strongly influenced not only by the distribution of data but also by the initialization solvers, and
  - Evaluators encounter results that are completely deviated from data reported in papers.
- Need to allow researchers and implementers to easily extend or add solvers and models for further evaluations.
- Need to accelerate researchers to devise new solvers for further improvements.

What is NMFLibrary?

- A readable, flexible and extensible software library of a collection of NMF solvers and its test environment.
- Operable and executable on MATLAB, and can be usable from python without any changes.
- Provide researchers and implementers a collection of
  - State-of-the-art NMF solvers to solve NMF models,
  - Plotting and drawing tools of performances, such as cost, optimality gap, and classification accuracies

Supported models and solvers (1/2)

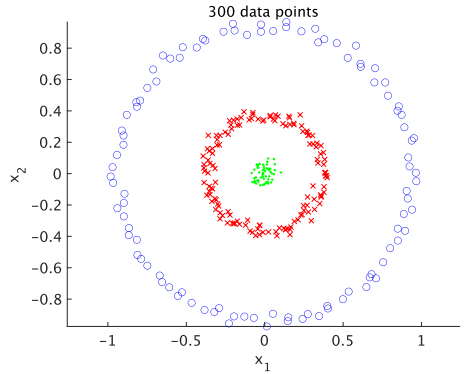
Category	Algorithms
Frobenius-norm	Fro-MU (multiplicative updates), Modified Fro-MU, Accelerated Fro-MU, PGD (projected gradient descent), ALS (alternative least squares), Hierarchical ALS, Accelerated H ALS
Divergence	Div-MU, Div-ADMM (Alternating direction method of multipliers), KL-FPA, KL-BMD
Semi	Semi-MU, Semi-BCD
Sparse	Sparse-MU, Sparse-MU-V, sparseNMF, SC-NMF, Nonsmooth-NMF, NS-NMF, Proj-Sparse,PALM-Sparse-Smooth-NMF
Orthogonal	DTPP, Orth-MU, ALT-ONMF, HALS-SO
Online/stochastic	Incremental (online)-MU, SPG, Robust-Online-MU, ASAG-MU. SVRMU,SAGMU
Probabilistic	PNMF-GIBBS, Prob-NMF
Deep	Deep-Semi, Deep-Bidir-Semi, Deep-nsNMF, Deep-Multiview-Semi
Convex	Convex-MU, Kernel-Convex-MU

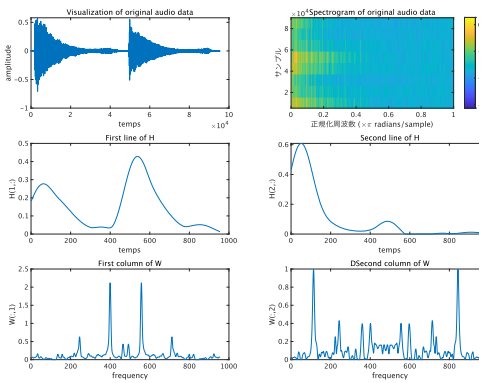
Supported models and solvers (2/2)

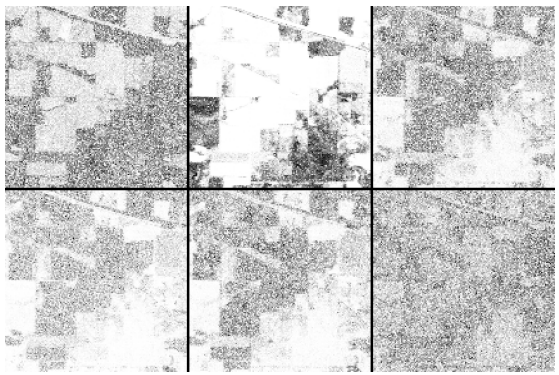
Category	Algorithms
Separable	SPA, SNPA
Variant	GNMF, NeNMF, SDNMF
Robust	Robust-MU
Symmetric	SymmANLS, SymmHALS, SymmNewton
Convolutive	MU-Conv, Heur-MU-Conv, ADMM-Y-Conv, ADMM-Seq-Conv
Projective	projectiveNMF
Rank2	Rank2-NMF
Nonnegative matrix tri-factorization	Sep-Symm-NMTF
Nonnegative under-approximation	Recursive-NMU
Minimum-volume	minvol-NMF
Weighted low-rank matrix approximation	Weighted Low-Rank matrix approximation

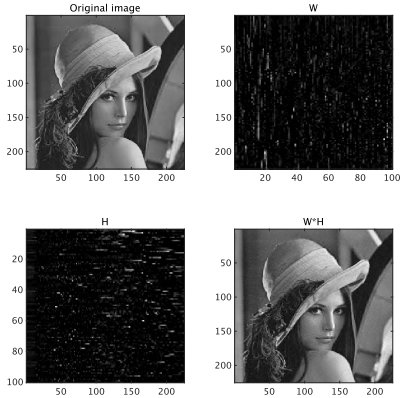
Application examples

- document topic extraction
- image (hyper-spectral image) analysis
- audio analysis/separation/denoising
- document/face image/graph clustering
- ....



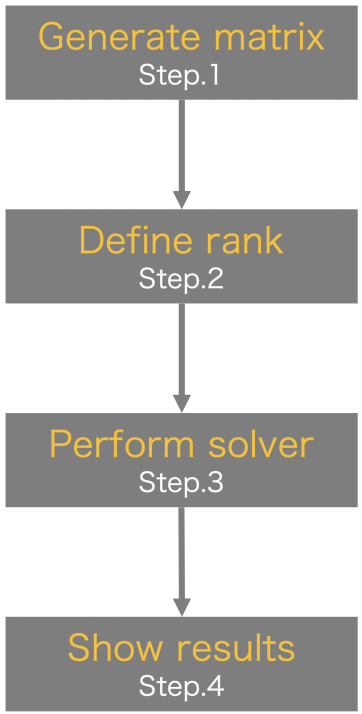






Tour of NMFLibrary

Only 4 steps for simple use !




```
V = rand(500, 100)
```

```
rank = 5
```

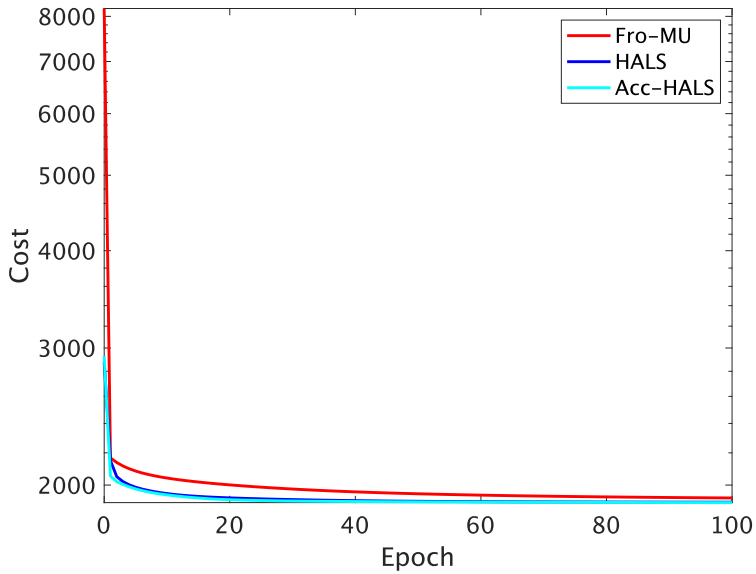
```
nmf(V,rank)
```

```
display_graph
```

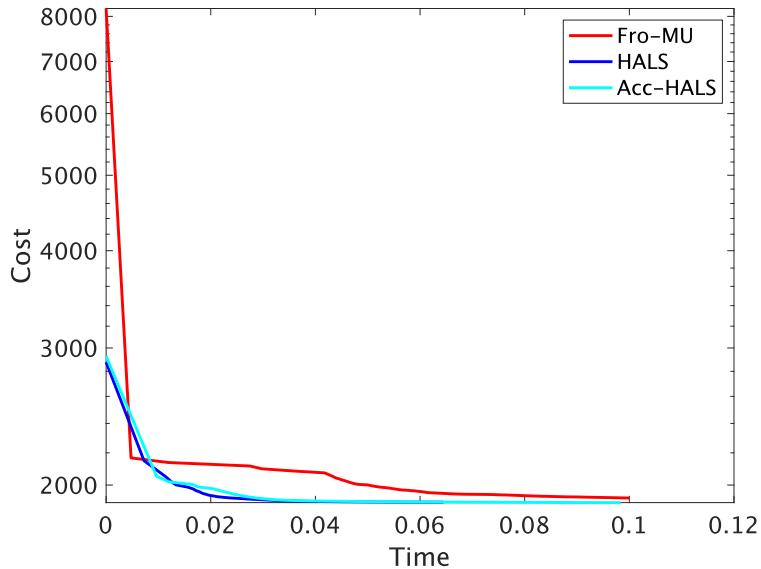
Demonstration code


```
% Step.1: generate a matrix size of 500x300
V = rand(500, 100);

% Step.2: define rank
rank = 5;

% Step.3: perform solvers

[w_mu,infos_mu] = fro_mu_nmf(V,rank);  % Frobenius-norm MU

options.alg = 'hals';
[w_hals,infos_hals] = als_nmf(V,rank); % Hierarchical ALS

options.alg = 'acc_hals';
[w_ahals,infos_ahals] = als_nmf(V,rank);  % Accelerated HALS

% Step.4: sfhow result
display_graph('epoch','cost',{'Fro-MU','HALS','Acc-HALS'},{
    w_mu,w_hals,w_ahals},{infos_mu,infos_hals,infos_ahals});
```

Plot results



(a) Epoch vs. Cost



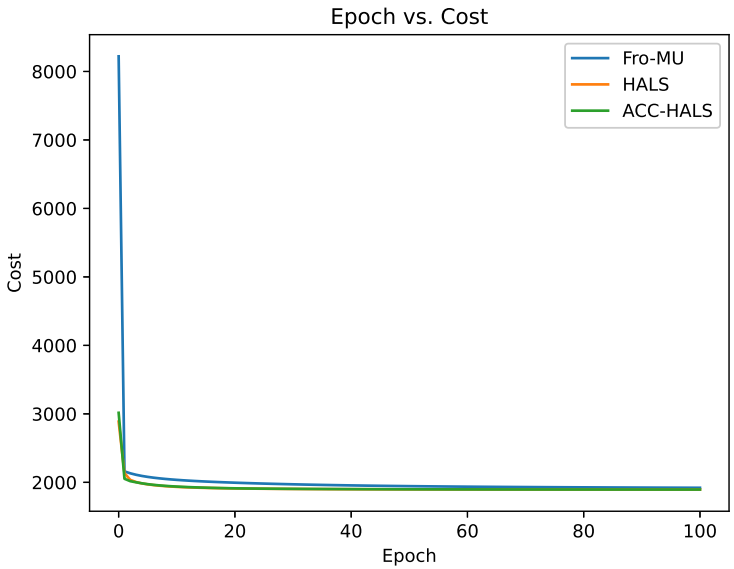
(b) Time vs. Cost

Figure: Convergence of Fro-MU, HALS, and ACC-HALS.

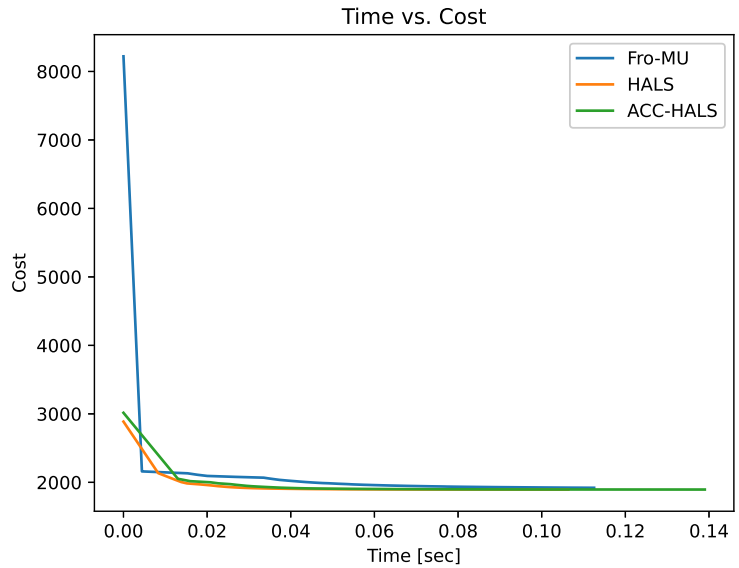
Easy to use from python

```python
V = np.random.random_sample((500,100)) # generate matrix
rank = 5                               # define rank

# import MATLAB module
import matlab.engine
eng = matlab.engine.start_matlab()

# add MATLAB path
eng.run_me_first(0, nargout=0)

# convert python data to matlab format
V_m = matlab.double(V.tolist())

# perform Frobenius-norm multiplicative update in MATLAB
[w,info] = eng.fro_mu_nmf(V_m, rank, nargout=2)

# convert matlab data to python format
epoch = list(info['epoch'][0])
cost  = list(info['cost'][0])

# show result of convergence
plt.plot(epoch, cost, label="Fro-MU")
```

python results



(a) Epoch vs. Cost



(b) Time vs. Cost

Figure: Convergence of Fro-MU, HALS, and ACC-HALS.

https://github.com/hiroyuki-kasai/NMFLibrary

GitHub

Thank you for listening!