Background overview and select applications.

Sergey Voronin, Ph.D.
Last modified: 09.10.2024

Sergey Voronin, Ph.D. | sergey.voronin @ outlook.com | https://svoronin.neocities.org | github.com/sergeyvoronin

Math, engineering, and computer science education from U.S. and Internationally. Applied math, scientific and high performance computing, large scale data analysis background.

B.S. in Applied Mathematics (minor Comp. Science) from Engineering School.

M.A. in Applied and Computational Mathematics.

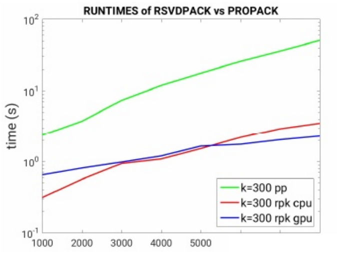
(Numerical differential equations, fluid mechanics and research in benchmarking general circulation climate models.)

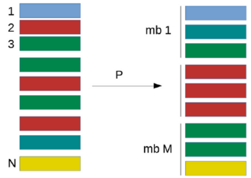
Ph.D. in Applied and Computational Mathematics, Princeton Univ., 2012.

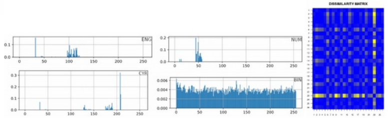
(Optimization problems with sparsity constraints, compressive sensing methods, applications to Geophysics and imaging, HPC implementations).

Work in academia post Ph.D. defense (2012 - 2017): signal processing, optimization, randomized algorithms for matrix manipulations / factorizations, imaging.

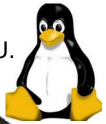
Work in industry (2017 - present): lossless compression, audio and video analysis / classification, anomaly detection / localization in electrical systems, multivariate time series, network analysis, etc. PI on multiple SBIR/STTR grants.

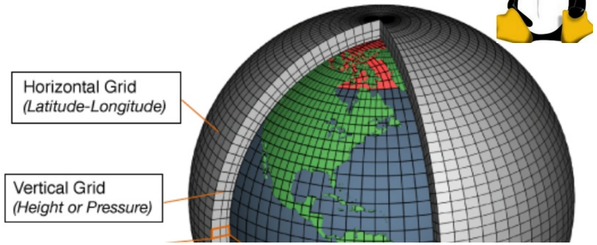






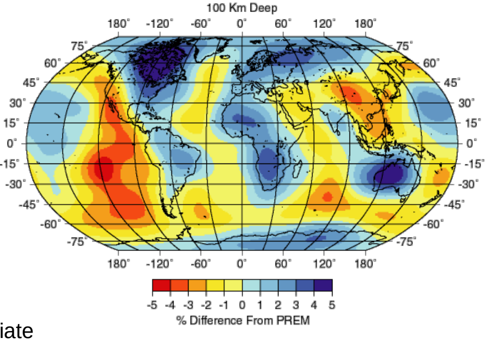
Different HPC and accelerated implementations with OpenMP, MPI, GPU.

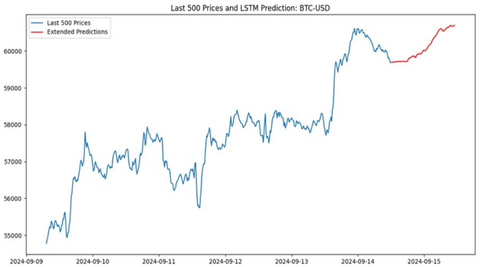




Parallelized implementations with novel approaches for dimensionality reduction and lossless data compression.

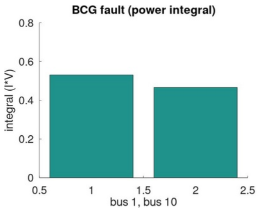
Algorithms and software for Geotomographical inversion from seismic measurements.

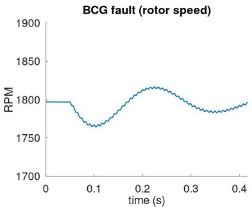


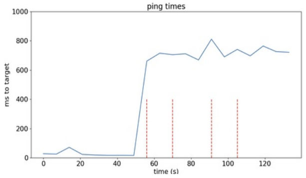


Parameter optimized machine learning implementations for multivariate time series (e.g. financial price/vol data). LSTM based iterative predictions, Gaussian processes.

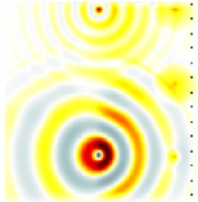
Compressed sensing developments. Sparse signal / transformed recovery.





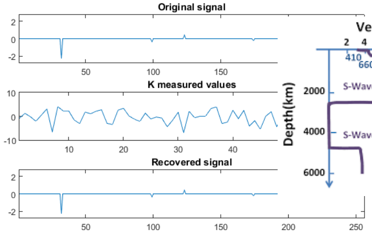


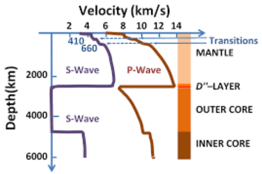
Development of statistical and AI-based methods for anomaly detection and localization in electrical systems and computer network applications.



Signal processing algorithms and software for audio, imagery, video. Microphone and antenna array applications.

Development of approaches for high noise / blur image reconstruction.

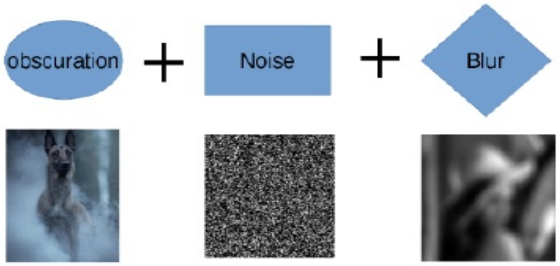


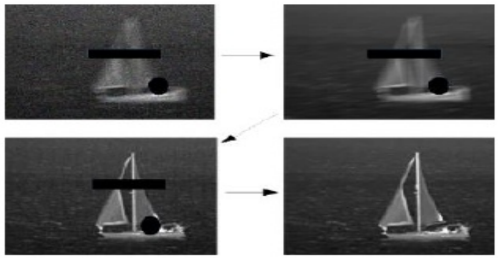


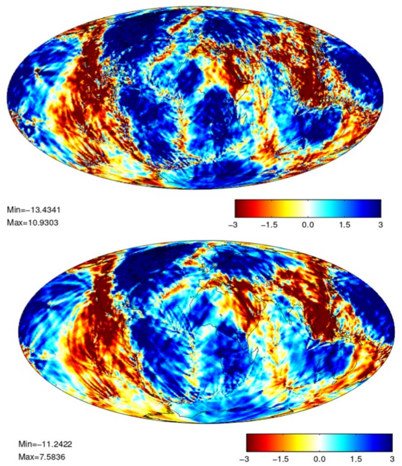
Background

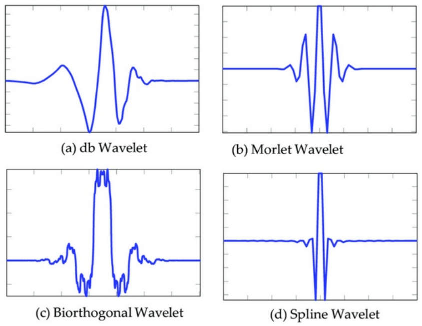
Postdoc (CNRS). Investigation of optimization based seismic inversion schemes for large data sizes on limited hardware. Developed projection and splitting methods. MPI implementation.

$$\bar{x} = \arg\min_{x} \left\| \begin{bmatrix} A \\ \sqrt{\lambda_1} I \\ \sqrt{\lambda_2} L \end{bmatrix} x - \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix} \right\|_2^2 \implies \begin{bmatrix} A \\ \sqrt{\lambda_1} I \\ \sqrt{\lambda_2} L \end{bmatrix}^T \begin{bmatrix} A \\ \sqrt{\lambda_1} I \\ \sqrt{\lambda_2} L \end{bmatrix} \bar{x} = \begin{bmatrix} A \\ \sqrt{\lambda_1} I \\ \sqrt{\lambda_2} L \end{bmatrix}^T \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix}$$

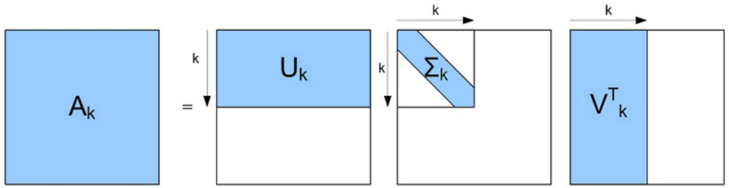


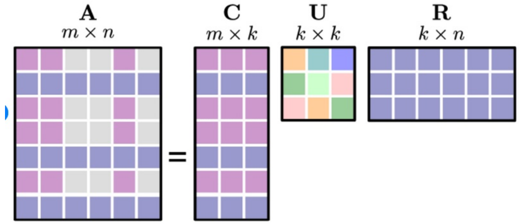






Postdoc (CU Boulder). Investigated randomized algorithms for obtaining low rank matrix factorizations (e.g. SVD, ID, CUR). Implemented RSVDPACK package. Instructor, N. Wiener Assistant Prof. (Tufts). Statistics, HPC.

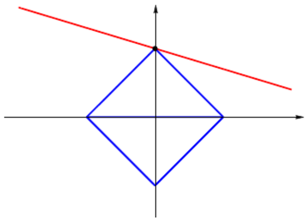


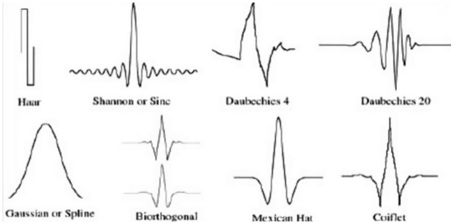


Research / Sr. Scientist (Intelligent Automation, Inc.). Filed proposals and white papers to DOD/DOT/DOE. PI on different topics including data compression / multi-channel systems / waveform formation with antenna arrays. Contributor to projects on PTSD detection, aircraft trajectory analysis, interceptor models, multi-fidelity simulations, traffic management, etc.

Research Scientist (Intel Corp.). Network data collector, analyzer, anomaly detector. Multivariate time series predictors, sorting, compression.

Compressive sensing and sparsity constrained opt





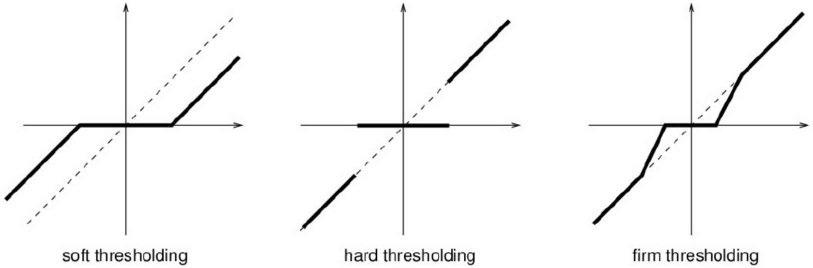
Seek most efficient representation in some basis.

$$(I)\; Ax=b\,,\ min\|x\|_0$$

$$\bar{w} = \arg\min_{w} \|RW^{-1}w - b\|_l + \lambda\|w\|_p \quad ; \quad x = W^{-1}\bar{w}$$

$$(II)\; Ax=b\,,\ min\|x\|_1 \rightarrow Ax\approx b\,,\ min\|x\|_1 \rightarrow min\|Ax-b\|+\lambda\|x\|_1$$

(I) and (II) equivalent under some conditions on A. (II) is a tractable problem. Much interest in minimization of \ell_1 penalized functional.



$$y^0 = x^0 \quad , \quad t_1 = 1 \quad , \quad x^{n+1} = \mathbb{S}_\tau(y^n - A^T(Ay^n - b))$$

$$t_{n+1} = \frac{1+\sqrt{1+4t_n^2}}{2}$$

$$y^{n+1} = x^n + \frac{t_n - 1}{t_{n+1}}(x^n - x^{n-1})$$

Simple iterative schemes depend on weighing factors and thresholding. BLAS 2/3 parallelization potential.

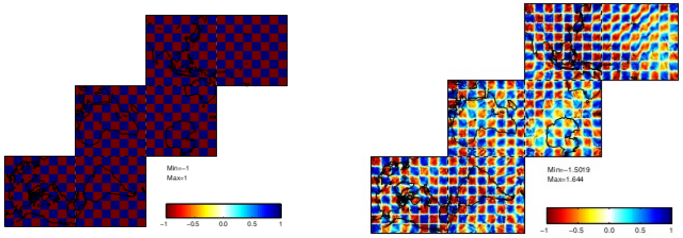
$$|x_k| = \frac{x_k^2}{|x_k|} = \frac{x_k^2}{\sqrt{x_k^2}} \approx \frac{x_k^2}{\sqrt{x_k^2+\epsilon^2}}$$

$$x_k^{n+1} = \frac{1}{1+\lambda_k q_k w_k^n}\left(x_k^n + (A^Tb)_k - (A^TAx^n)_k\right) \quad \text{for} \quad k = 1,\dots,N,$$

Generalized variable residual and solution norm scheme, involves multiple iterations of CG solves.

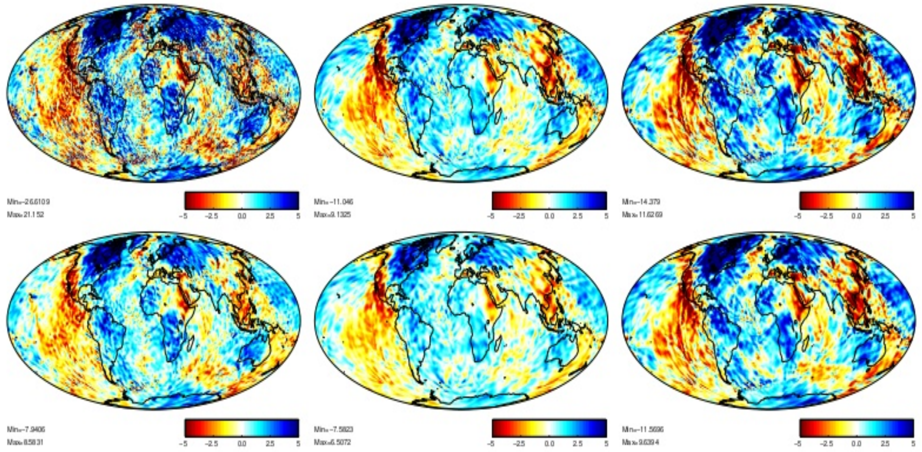
$$\left(A^TR^nA + (D^n)^T(D^n)\right)x^{n+1} = A^TR^nb$$

Approximation / projection techniques for large data sizes



Wavelet thresholding and low rank projection methods.

$$A = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix} \rightarrow M = \begin{bmatrix} \mathbb{T}(W r_1^T)^T \\ \mathbb{T}(W r_2^T)^T \\ \vdots \\ \mathbb{T}(W r_m^T)^T \end{bmatrix} = \mathbb{T}(AW^T) \approx AW^T$$



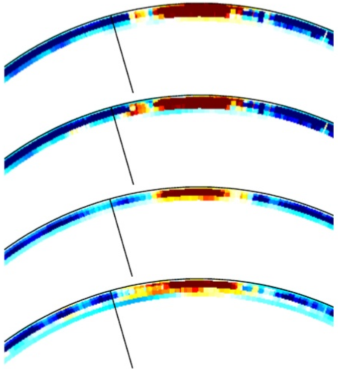
$$Mx \approx AW^T x \quad \text{and} \quad M^T y \approx (AW^T)^T y = WA^T y,$$

$$Ax \approx MW^{-T} x \quad \text{and} \quad A^T y \approx W^{-1} M^T y.$$

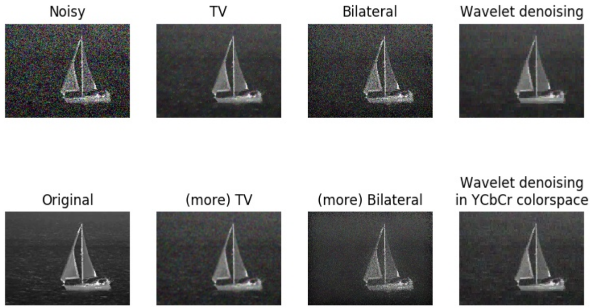
Projectors from first k eigenvectors.

$$\begin{bmatrix} U_{k_1}^T A_1 \\ U_{k_2}^T A_2 \\ \vdots \\ U_{k_p}^T A_p \end{bmatrix} x = \begin{bmatrix} U_{k_1}^T b_1 \\ U_{k_2}^T b_2 \\ \vdots \\ U_{k_p}^T b_p \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \Sigma_{k_1} V_{k_1}^T \\ \Sigma_{k_2} V_{k_2}^T \\ \vdots \\ \Sigma_{k_p} V_{k_p}^T \end{bmatrix} x = \begin{bmatrix} U_{k_1}^T b_1 \\ U_{k_2}^T b_2 \\ \vdots \\ U_{k_p}^T b_p \end{bmatrix}$$

Decreased runtimes enabling more data to be used, but less detail at greater depths, where resolution is poor.



Applications to image enhancement



$$W^{\{-1\}}\left(Thr(Wx)\right) \qquad V(I) = \Sigma_{ij}\sqrt{\left|I_{i+1,j} - I_{i,j}\right|^2 + \left|I_{i,j+1} - I_{i,j}\right|^2}$$

$$g(x,y) = h(x,y) \star f(x,y) + n(x,y),$$

$$G(u,v) = H(u,v)F(u,v) + N(u,v)$$

$$\hat{F}(u,v) = W(u,v)G(u,v)$$

$$W(u,v) = \frac{H^{\{*\}}(u,v)}{|H(u,v)|^2 + K(u,v)}$$

CS based (e.g. matrix completion) pixel reconstruction

$$\mathcal{D}_\tau(\boldsymbol{X}) := \boldsymbol{U}\mathcal{D}_\tau(\boldsymbol{\Sigma})\boldsymbol{V}^*,$$

$$\boldsymbol{A} \mapsto \arg\min \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\|_F^2 + \lambda\|\boldsymbol{X}\|_*.$$

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{X}\|_* \\ \text{subject to} & X_{ij} = M_{ij}, \quad (i,j) \in \Omega, \end{array}$$

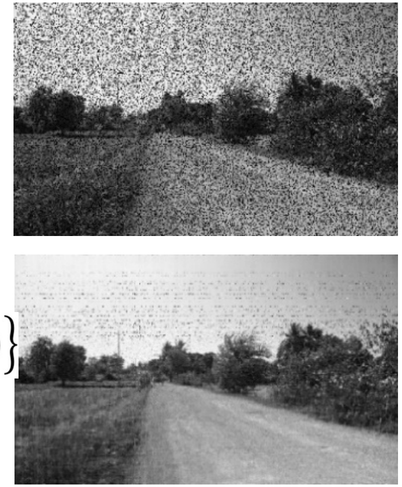
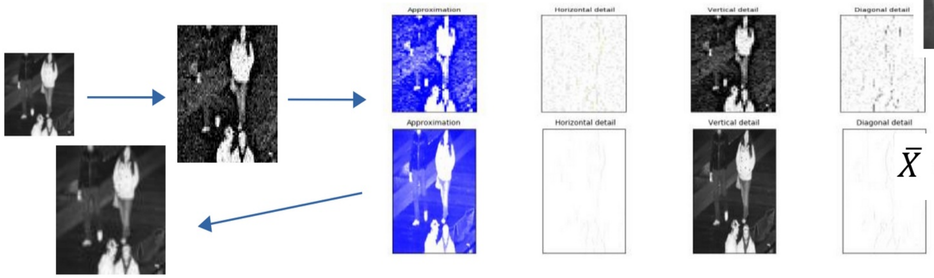
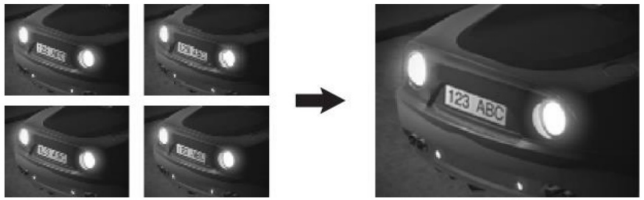


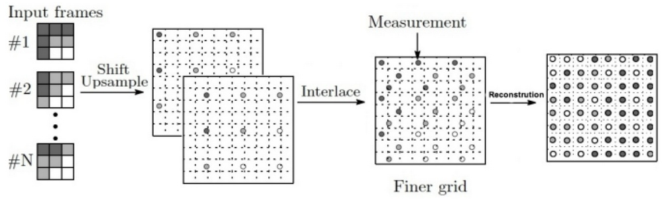
Image upscaling via CS + residual correction

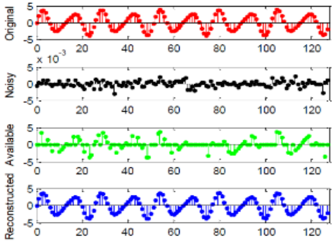




Super-resolution

$$\bar{X} = argmin_x \left\{\Sigma_{\{k=1\}}^{M} \left|\left|D_k H_k F_k X - Y_k\right|\right|_2^2 + \lambda R(X)\right\}$$





Gradient based reconstruction of missing pixels [Stankovic et al].

$$x_a^{(0)}(n) = \begin{cases} 0 & \text{for missing samples, } n \in \mathbb{N}_Q \\ x(n) & \text{for available samples, } n \in \mathbb{N}_A \end{cases}$$

$$\begin{array}{ll} \min & \|\mathbf{X}_a\|_1 \\ \text{subject to} & x_a^{(m)}(n) = x(n) \quad \text{for } n \in \mathbb{N}_A. \end{array}$$

Research combinations of transform / thresholding, optimization based and machine learning methods.

Randomized algorithms

Choose large N,

```
>N = 1e5; x = randn(N,1); y = randn(N,1);
> x = x/norm(x); y = y/norm(y);
> abs(x'*y)
ans =  0.0033332
>
```

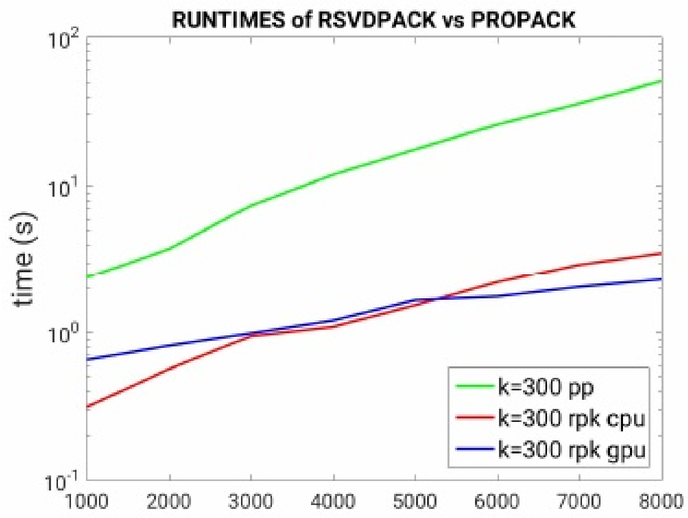
O(mnk) vs O(mn^2)

$$A = [u_1 \dots u_r] \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_r \end{bmatrix} \begin{bmatrix} v_1^* \\ \vdots \\ v_r{}^* \end{bmatrix} = U\Sigma V^*$$

$$\approx U_k \Sigma_k V_k^* = [u_1 \dots u_k] \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_k \end{bmatrix} \begin{bmatrix} v_1^* \\ \vdots \\ v_k{}^* \end{bmatrix}$$

Sample range of A with $k+p$ lin. indep. vectors, so that $QQ^*A \approx A$.

- Draw an $n \times (k+p)$ Gaussian random matrix Ω.
  `Omega = randn(n,k+p)`
- Form the $m \times (k+p)$ sample matrix $Y = A\Omega$.
  `Y = A * Omega` ; ran$Y \approx$ ran$A$
- Form an $m \times (k+p)$ orthonormal matrix Q such that $Y = QR$.
  `[Q, R] = qr(Y)` ; ran$Q \approx$ ran$A$
- Form the $(k+p) \times n$ matrix Q^*A.
  `B = Q' * A`
- Compute the SVD of the smaller $(k+p) \times n$ matrix B: $B = \hat{U}\Sigma V^*$.
  `[Uhat, Sigma, V] = svd(B)`
- Form the matrix $U = Q\hat{U}$.
  `U = Q * Uhat` ; $QQ^*A \approx A$
- $U_k = U(:, 1:k), \Sigma_k = \Sigma(1:k, 1:k), V_k = V(:, 1:k)$.



How to determine factorization rank adaptively based on tolerance? General form for different factorizations, parallelizable block based hierarchical approach for large sizes.

$$\|A - QB\| < \epsilon \Rightarrow QB = QQ^T A \Rightarrow QB \approx Q\hat{U}SV^T$$

$$\begin{aligned} \bar{y} &= (I - QQ^T)y \\ q &= \frac{\bar{y}}{\|\bar{y}\|} \\ \bar{Q} &= [Q, q] \end{aligned}$$

$$\mathrm{range}(\bar{Q}) = \mathrm{span}(\mathrm{range}(Q) \cup y).$$

$$\bar{Q}^T \bar{Q} = I_{r+1}.$$

Instead of adding one vector at a time, add blocks at once.

function $[Q, B] = \texttt{randQB\_pb}(M, \varepsilon, q, b_p)$

(1) **for** $i = 1, 2, 3, \ldots$

(2) $\Omega_i = \texttt{randn}(n, b_p)$.

(3) $Q_i = \texttt{orth}(M\Omega_i)$.

(4) **for** $j = 1 : q$

(5) $Q_i = \texttt{orth}(M^T Q_i)$.

(6) $Q_i = \texttt{orth}(M Q_i)$.

(7) **end for**

(8) $Q_i = \texttt{orth}(Q_i - \sum_{j=1}^{i-1} Q_j Q_j^T Q_i)$

(9) $B_i = Q_i^T M$

(10) $M = M - Q_i B_i$

(11) **if** $\|M\| < \varepsilon$ **then stop**

(12) **end while**

(13) Set $Q = [Q_1 \ \cdots \ Q_i]$ and $B = [B_1^T \ \cdots \ B_i^T]^T$.

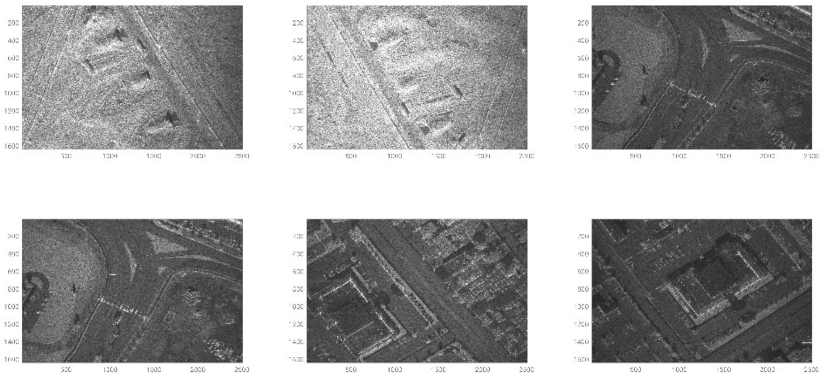
Hierarchical parallel implementation:

$$M = \begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} \approx \begin{bmatrix} Q_1 B_1 \\ Q_2 B_2 \\ Q_3 B_3 \\ Q_4 B_4 \end{bmatrix} = \begin{bmatrix} Q_1 & 0 & 0 & 0 \\ 0 & Q_2 & 0 & 0 \\ 0 & 0 & Q_3 & 0 \\ 0 & 0 & 0 & Q_4 \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix}$$

$$M^{(1)} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \approx Q_{12} B_{12} \quad ; \quad M^{(2)} = \begin{bmatrix} B_3 \\ B_4 \end{bmatrix} \approx Q_{34} B_{34} \qquad M^{(3)} = \begin{bmatrix} B_{12} \\ B_{34} \end{bmatrix} \approx Q_{1234} B_{1234}$$

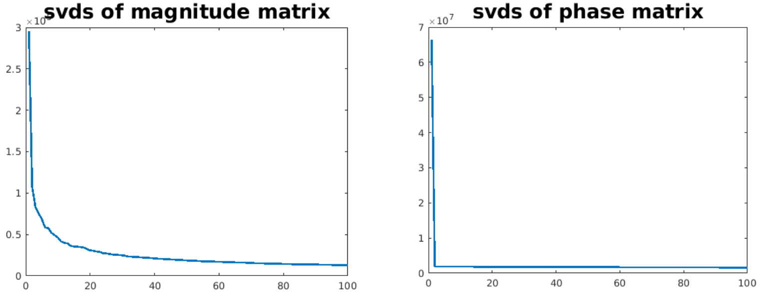
$$\begin{aligned} M &\approx \begin{bmatrix} Q_1 & 0 & 0 & 0 \\ 0 & Q_2 & 0 & 0 \\ 0 & 0 & Q_3 & 0 \\ 0 & 0 & 0 & Q_4 \end{bmatrix} \begin{bmatrix} Q_{12} & 0 \\ 0 & Q_{34} \end{bmatrix} \begin{bmatrix} B_{12} \\ B_{34} \end{bmatrix} \approx \begin{bmatrix} Q_1 & 0 & 0 & 0 \\ 0 & Q_2 & 0 & 0 \\ 0 & 0 & Q_3 & 0 \\ 0 & 0 & 0 & Q_4 \end{bmatrix} \begin{bmatrix} Q_{12} & 0 \\ 0 & Q_{34} \end{bmatrix} Q_{1234} B_{1234} \\ &= Q^{(3)} Q^{(2)} Q^{(1)} B^{(1)} = QB \end{aligned}$$

Many applications of low rank matrix / tensor factorizations … e.g. complex SAR imagery



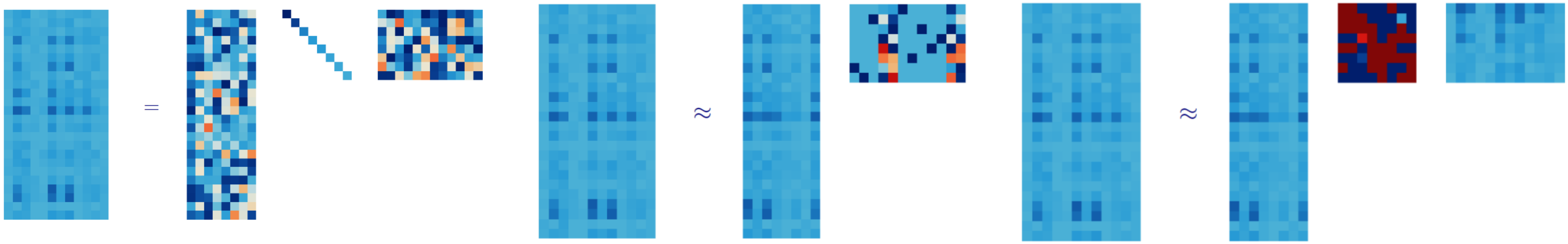
Sample SAR data from open Sandia set.

im = magd_sar .* exp (j *phased_sar)



Decay of data matrices (also common to other applications).

Rank-k svd gives best error bound, but ID, CUR useful for applications: $A \approx CV^T, \quad \text{where} \quad C = A(:, J_c(1:k)), \quad V^T = \begin{bmatrix} I_k & T_l \end{bmatrix} P^T$



Can build multiple representations (SVD, ID, CUR) efficiently with randomized schemes.

Some recent project directions..

() Machine Learning
Classification and regression problems. Parameter optimizations. Boosting and ensemble schemes. Data imputation and correction strategies. Anomaly detection.

(I) Antenna array systems
How to construct a ‘fancy’ signal in the far field using multiple transmitters in place of single large / expensive transmitter.

(II) PTSD detection
How to process audio data from medical interview, segment out the patient voice, extract relevant features, and assign likelihood score of emotion state or PTSD likelihood.

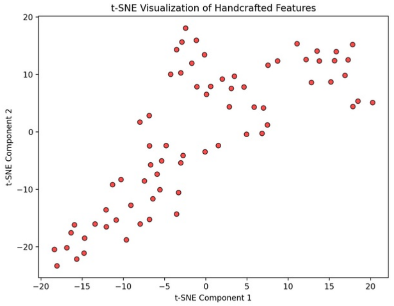
(III) Data compression for heterogeneous bundles, audio, and image sequences.
Heterogeneous data. Sets of similar signals (e.g. microphone arrays).

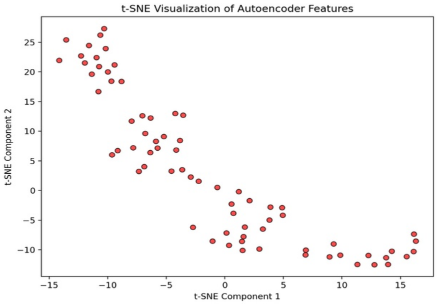
(IV) Gaussian process regression for multi-fidelity data applications.

(V) Accelerated implementation. (VI) Network data analysis. (VII) Multivariate time series.

Machine learning

Implementations for several classification and regression problems with varying data sizes and quality (time series, text, audio, and imagery data applications).

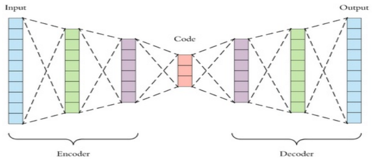




Feature set projections.

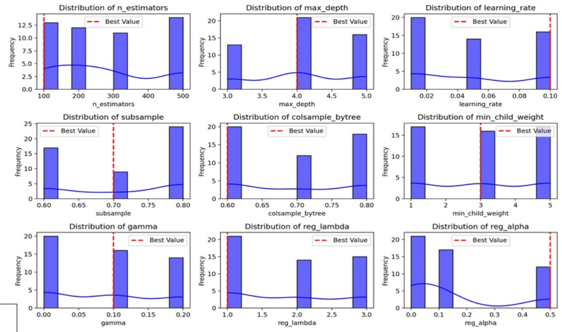
Using autoencoders to supplement handcrafted feature sets by considering low dimensional data or feature set representation.

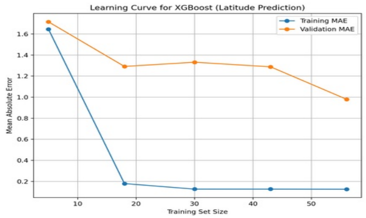
```
autoencoder = Model(inputs=input_layer, outputs=decoded)
encoder = Model(inputs=input_layer, outputs=encoded)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
history = autoencoder.fit(X_scaled, X_scaled,
                          epochs=250,
                          batch_size=16,
                          shuffle=True,
                          validation_split=0.2,
                          verbose=1)

encoded_features = encoder.predict(X_scaled)
```



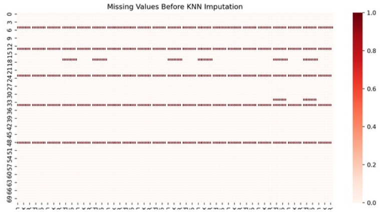

```
param_grid = {
    'n_estimators': [100, 200, 300, 500],  # Number of boosting trees
    'max_depth': [3, 4, 5],  # Depth of each tree (shallower trees prevent overfitting)
    'learning_rate': [0.01, 0.05, 0.1],  # Controls how much each tree contributes
    'subsample': [0.6, 0.7, 0.8],  # Use less data per tree to add randomness
    'colsample_bytree': [0.6, 0.7, 0.8],  # Use fewer features per tree to prevent overfitting
    'min_child_weight': [1, 3, 5],  # Minimum weight required to split a node
    'gamma': [0, 0.1, 0.2],  # Penalizes small splits to avoid overfitting
    'reg_lambda': [1, 2, 3],  # L2 regularization (reduces model complexity)
    'reg_alpha': [0, 0.1, 0.5]  # L1 regularization (adds sparsity, feature selection)
}
```

Parameter tuning via cross validation or Bayesian methods allows for optimized model performance.

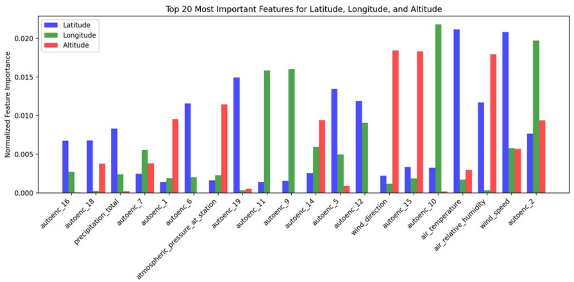




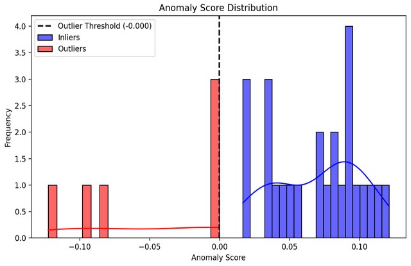
Learning curve for under and over fitting detection.



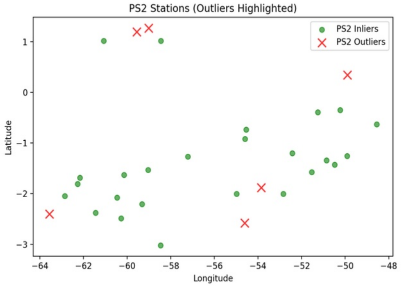
Missing data analysis and imputation.



Feature importance analysis, mapping to augmented feature matrix columns.



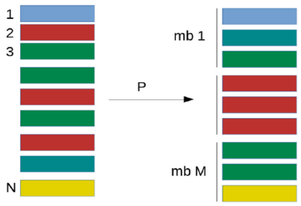
Anomaly identification and scoring.

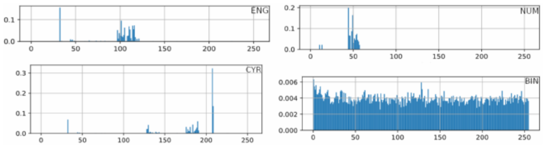


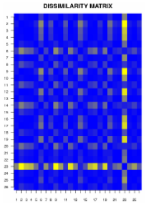
Cluster analysis.

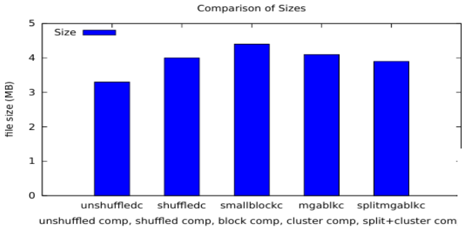
> Parallel data compression for heterogeneous data.

Achieving good compression ratios by clustering. Sets of similar signals (e.g. microphone arrays).

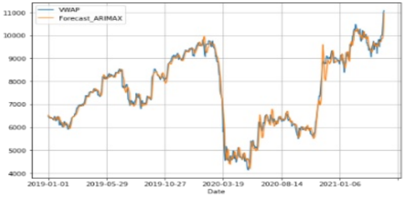


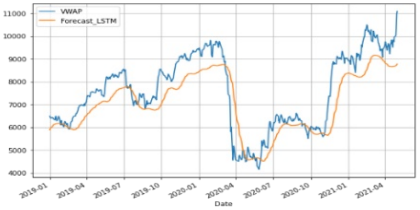


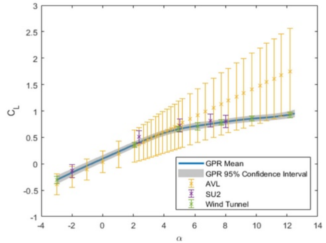


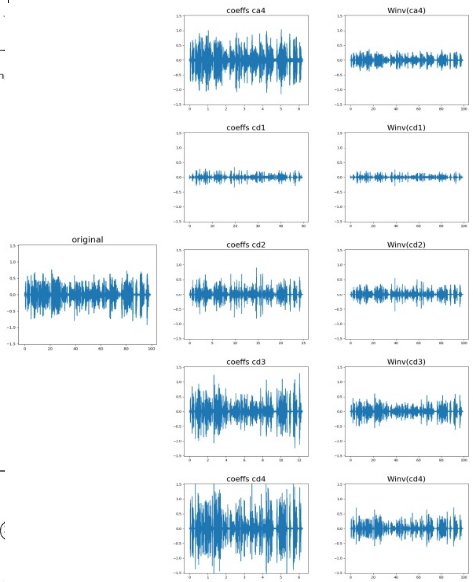


> Multivariate time series: prediction, data integration, anomaly detection.

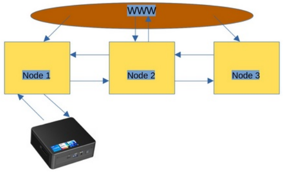


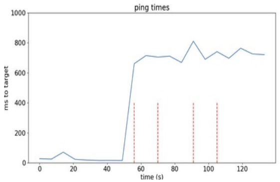


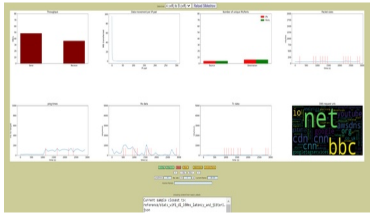


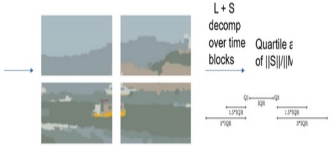


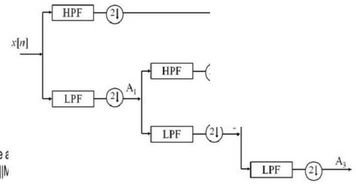
> Network data analysis. Data collection and anomaly detection with FPGAs.







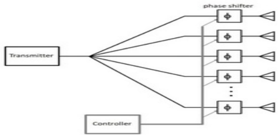


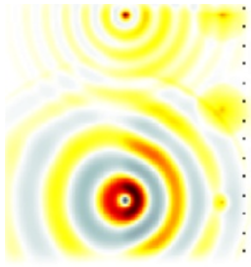


> Audio analysis / Antenna array systems / Video / etc.

How to process audio data from medical interview, segment out the patient voice, extract relevant features, and assign likelihood score of emotion state or medical indicators.

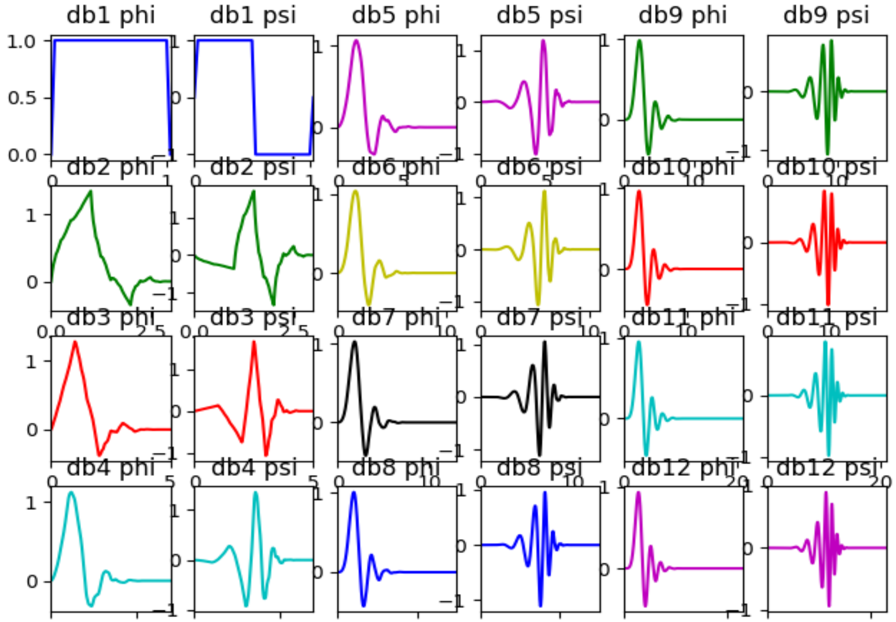
How to construct a signal in the far field using multiple transmitters?

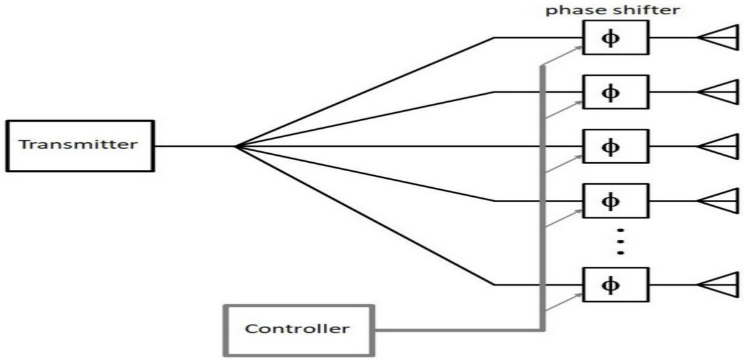


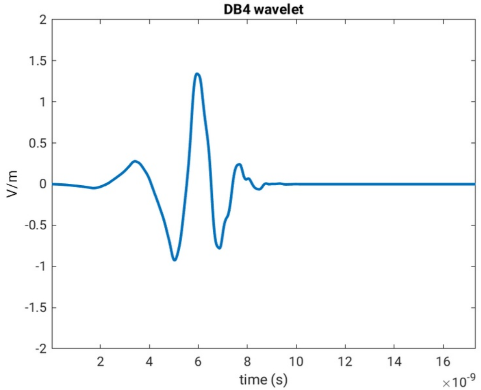


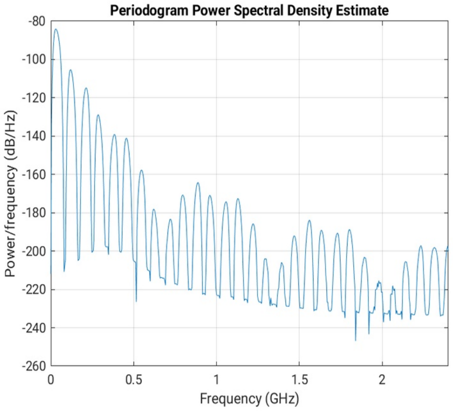
Antenna array systems

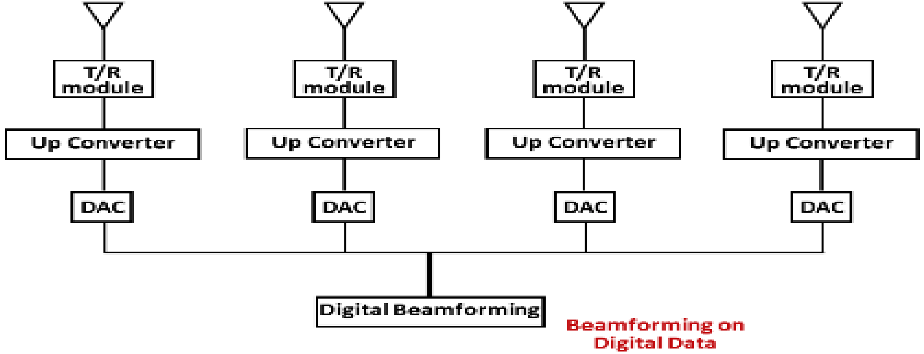
Project considered antennas which can emit compactly supported wavelets. (non-trivial hardware implementation).







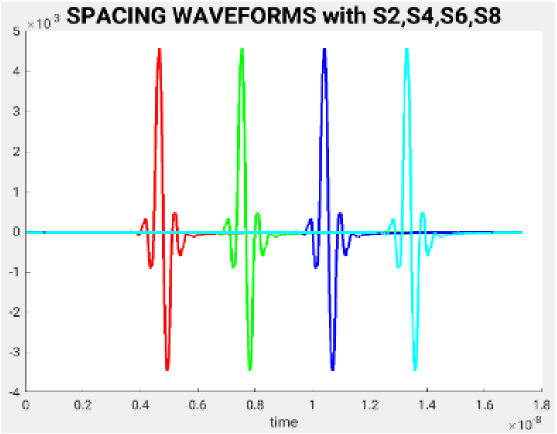




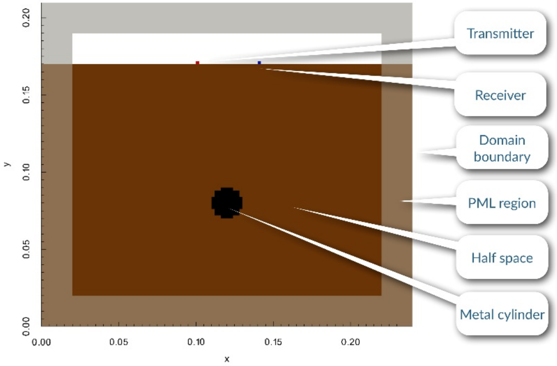
superimposed signal at receiver:

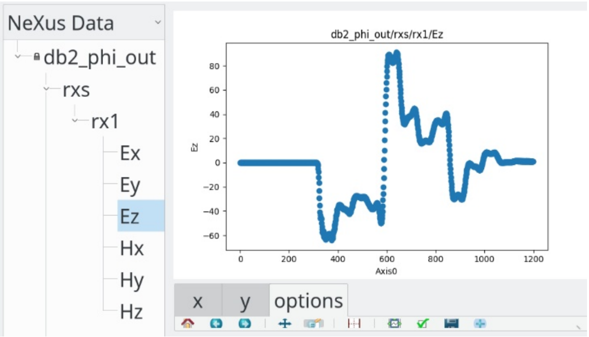
$$\sum_i [a_i x_i(b_i t - t_i) \star h_i] \star c_i$$

Time delayed signals, convolved with antenna filter and propagation term (numerically determined).

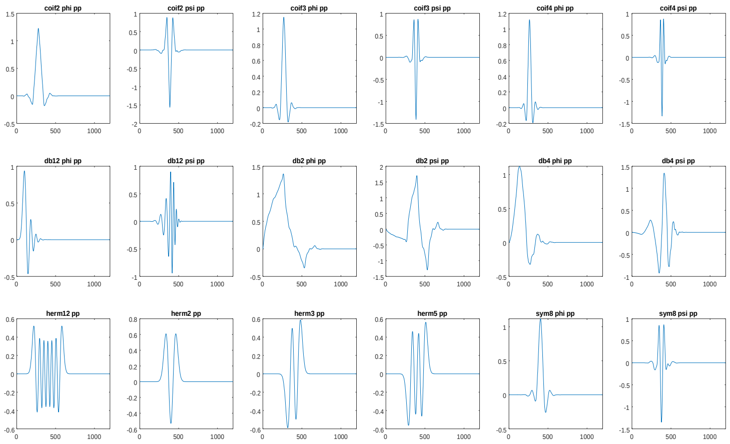


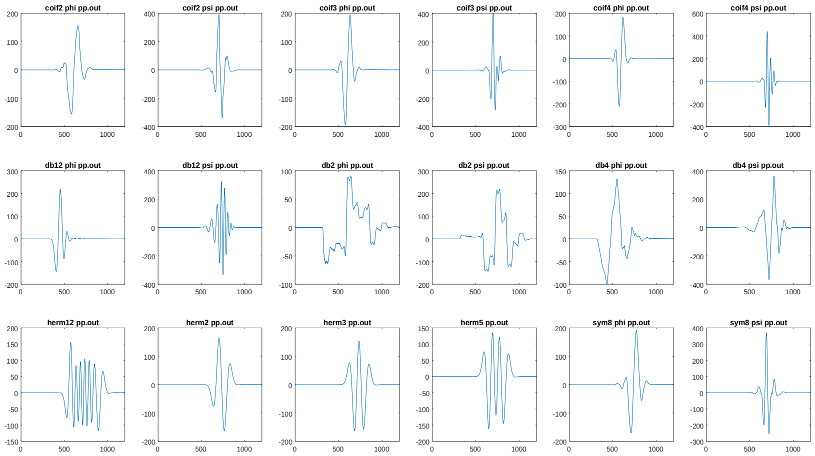
FDTD modeling for the environment.

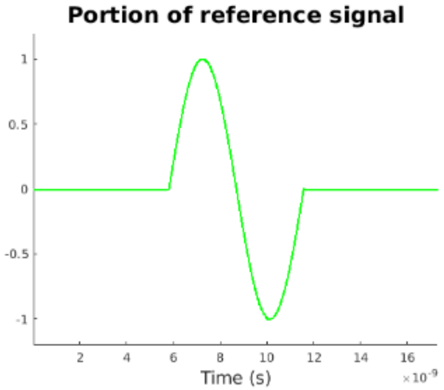


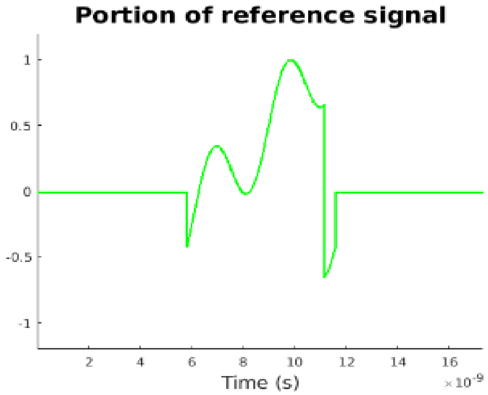


$$f_k(t) \approx \sum_{i=1}^{n} \alpha_i S^{(t_k+\delta_i)} w_i(t)$$









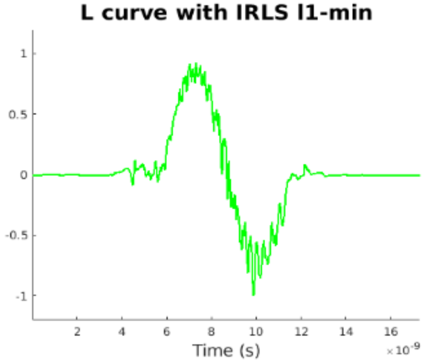
Desired reference signal at given window level.

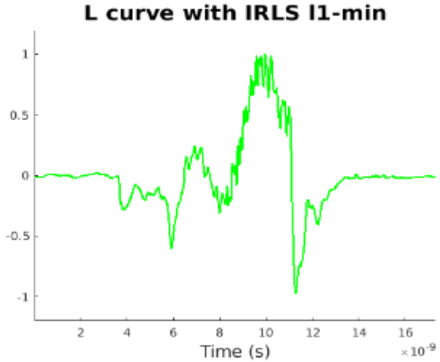
$$\begin{bmatrix} \hat{w}_1(t_1,\delta_1) & \dots & \hat{w}_n(t_1,\delta_n) \\ \dots & \dots & \dots \\ \hat{w}_1(t_m,\delta_1) & \dots & \hat{w}_n(t_m,\delta_n) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \dots \\ \alpha_n \end{bmatrix} \approx \begin{bmatrix} f_k(t_1) \\ \dots \\ f_k(t_m) \end{bmatrix}$$

$$f_k(t) \approx \sum_{\{i=1\}}^{n} \alpha_i \, S^{\{t_k+\delta_i\}} w_i(t)$$

Use regularization and coordinate descent scheme to determine weighing factors and time delays.

$S^{\{t_k+\delta_i\}} w_i(t)$ Time delayed signal via linear operator.





Can be solved as a regularized least squares problem + outer opt loop for remaining parameters.

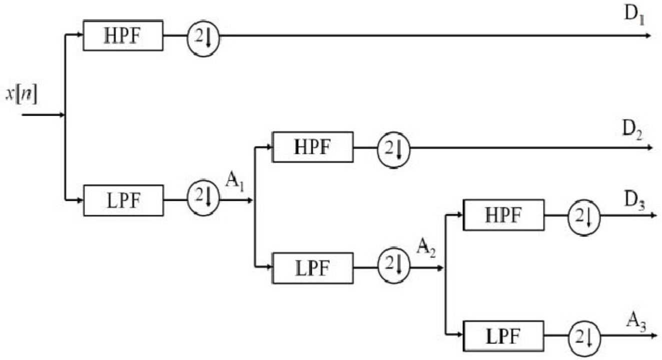
$$\min_x ||Ax - b||_l^l + \lambda\phi(x), \quad \phi(x) = ||x||_p^p$$

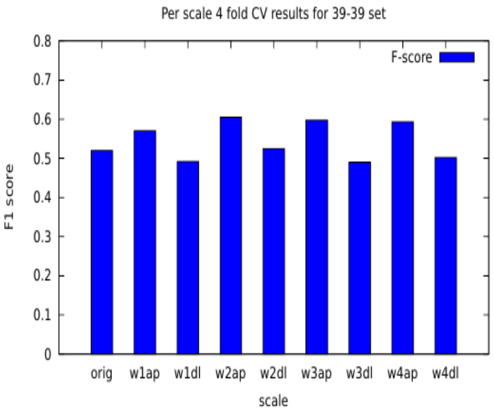
$(A^T S_k A + \lambda D_k^T D_k)x = A^T S_k b$ with iteration dependent diagonal matrices S_k, D_k.

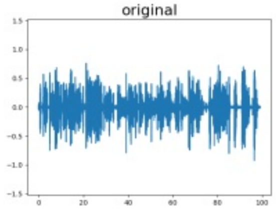
Application: emotion / PTSD detection from audio interviews.

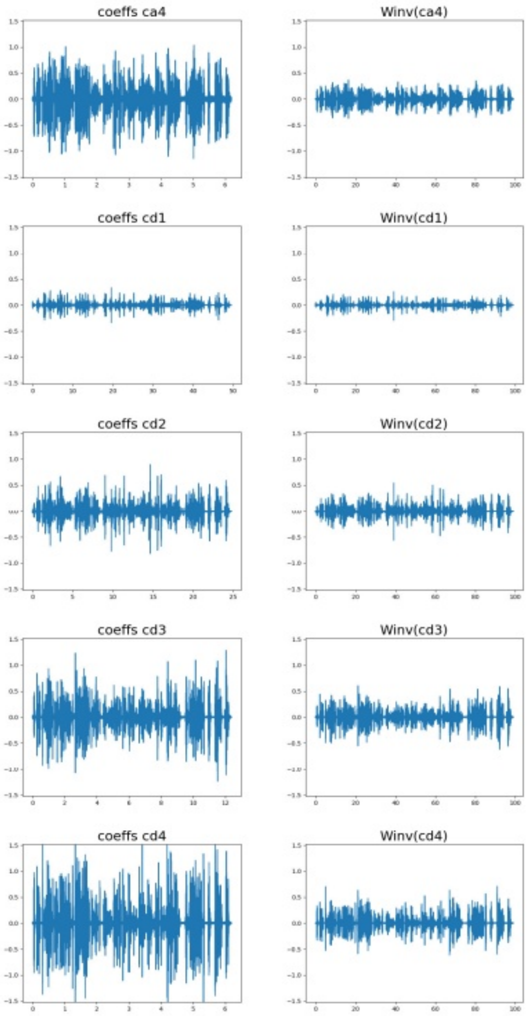
Input: unsegmented audio interview, output: condition assessment score

Input audio can be decomposed into approximation and scaling coefficients using high / low pass filters and downsampling.









Different representations give different performance.

For original signal x, compute multi-level Wavelet transform:

- Collect approximation and detail coefficients.
- Apply inverse transform to obtain approximation and high level details.
- Repeat for 4 different bases (sharp and smooth).

x Original waveform

$a_{s_i} = W_i^{-1}[0; w_a]$ Coarse Approximation

$d_{s_i} = W_i^{-1}[w_d; 0]$ Fine Details

We extract ~45 features per set, corresponding to spectral (e.g. MFCC), audio (e.g. tempo), and time series (e.g. mean auto-correlation) statistics.

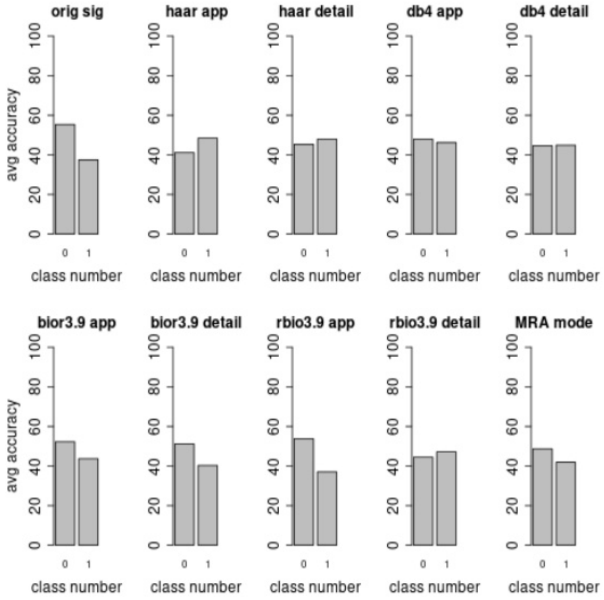
LibRosa, Aubio libraries

Algorithms (Java Weka, Scikit, TF) : Tree-based methods, LSTM (with all 9 feature sets in megatensor)

```
for features from file i in train_set:
seq = [np.array(unt_rows[i]), np.array(wav1ap_rows[i]),...,
      np.array(wav4dl_rows[i])]
data.append(seq)

Xnew = np.zeros((num_files,nsets,nfeatures));
for i in range(0,num_files):
    for j in range(0,nsets):
        X_tr[i][j][:] = data[i][j];
```

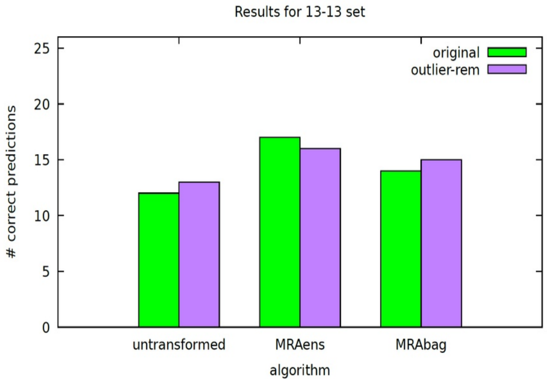
We obtain probabilities for class_0 and class_1 with each algorithm j=1,..,M and feature set k=1,..,9 :

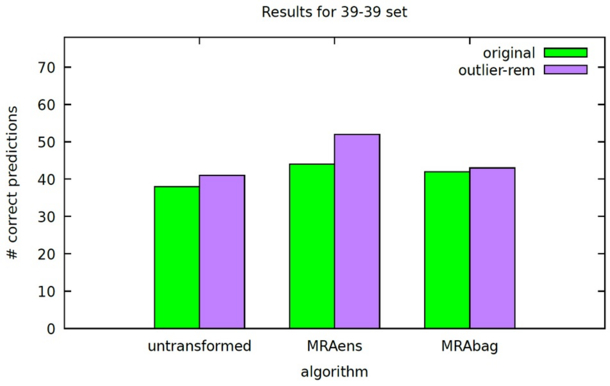
$$P(C\_0 \mid A\_j, S\_k) \;,\; P(C\_1 \mid A\_j, S\_k)$$



Use an ensemble scheme, with weighted mean.

$$\sum w_{\{j,k\}} P(C_l | A_j, S_k) / \sum \{w_{\{j,k\}}\}$$



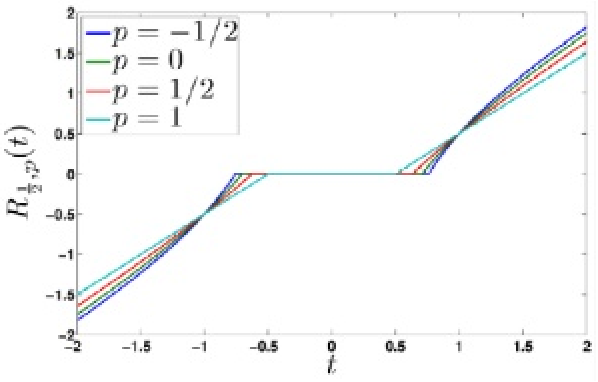


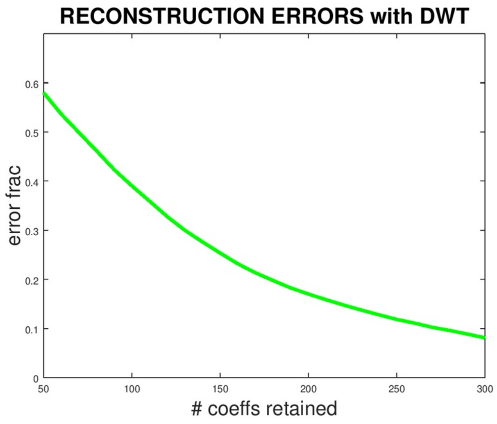
Application: Data compression

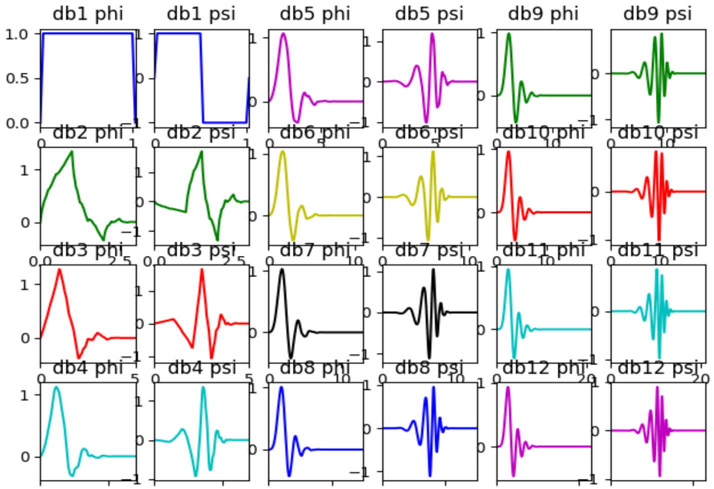
Increasing adoption of high resolution content (e.g. 4K, high fidelity auido) various big data applications, effective parallel compression algorithms are becoming increasingly important. Both lossless and lossy compression are of interest (e.g. for text documents, where loss of information is not acceptable and for audio and image data where some losses are often plausible).

Lossy compression based on transform / thresholding schemes for small coefficients. Can use e.g. CDF 9-7 wavelets and firm thresholding. Idea based on relation:

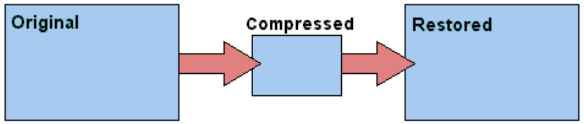
$$s \approx W_i^{\{-1\}}(Thr(W_i s))$$







Lossless compression based on reducing alphabet size and encoding frequently occurring symbols with fewer bits. Needed for e.g. text/numerical data, when loss of information is not acceptable.



Entropy of a set of elements $e_1,\ldots,e_n$ with probabilities $p_1, \ldots p_n$ is:

$$H(p_1 \ldots p_n) \equiv -\sum_{\forall i} p_i \log_2 p_i$$

Critically, one must seek to reduce data entropy to improve compression performance.

$$Max\ \ when\ \ p_1 = \cdots = p_n = \frac{1}{n} \rightarrow H = \log_2(n)$$

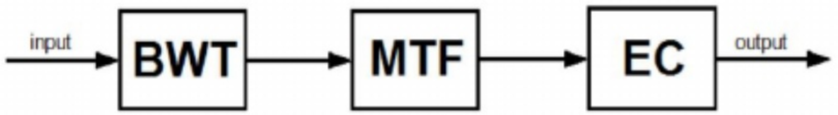
Example 1: A p_A=0.5, B p_B=0.5

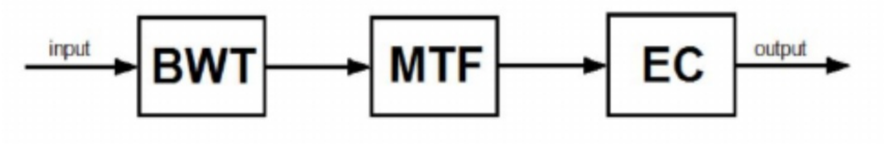
$$H(A,B) = -p_A \log_2 p_A - p_B \log_2 p_B = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

Example 2: A p_A=0.8, B p_B=0.2

$$H(A,B) = -p_A \log_2 p_A - p_B \log_2 p_B = -0.8 \log_2 0.8 - 0.2 \log_2 0.2 \cong 0.7219$$

Burrows-Wheeler transform based compression.





BWT rearranges input to reveal patterns, MTF/RLE move common symbols to front, compress sequences of identical digits, EC (Huffman or Arithmetic coding), encodes remaining data in fewer bits.

BWT typically needs to be performed over small chunks due to expensive string sorting. Can use suffix arrays and take advantage of triangular structure .. towards O(n log n).

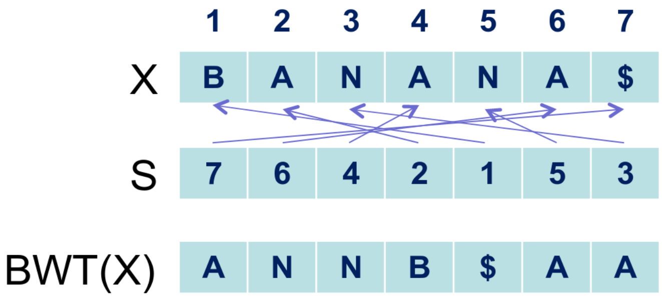
SA[i]>0 ? BWT[i]=T[SA[i]-1] : $

general purpose compressors (bzip2, lbzip2)

Need index permutation information from the sort. Developed O(n) counting sort with permutation information.

```
count = array of k+1 zeros
for x in input do
    count[key(x)] += 1

total = 0
for i in 0, 1, ... k do
    count[i], total = total, count[i] + total

output = array of the same length as input
for x in input do
    output[count[key(x)]] = x
    count[key(x)] += 1

return output
```

$BANANA	1 $BANANA	1 banana$		7 $
A$BANAN	2 A$BANAN	2 anana$		6 a$
ANA$BAN	3 ANA$BAN	3 nana$	Sorting	4 ana$
ANANA$B	4 ANANA$B	4 ana$	----------->	2 anana$
BANANA$	5 BANANA$	5 na$	alphabetically	1 banana$
NA$BANA	6 NA$BANA	6 a$		5 na$
NANA$BA	7 NANA$BA	7 $		3 nana$



Move to front transform and Entropy coding: Huffman or Arithmetic coding.

Iteration	Index sequence	Symbol list
mississippi	12	abcdefghijklmnopqrstuvwxyz
m**i**ssissippi	12,9	mabcdefghijklnopqrstuvwxyz
mi**s**sissippi	12,9,19	imabcdefghjklnopqrstuvwxyz
mis**s**issippi	12,9,19,0	simabcdefghjklnopqrtuvwxyz

```
encode_symbol(symbol,cum_freq)

    range = high - low

    high  = low + range*cum_freq[symbol-1]

    low   = low + range*cum_freq[symbol]
```

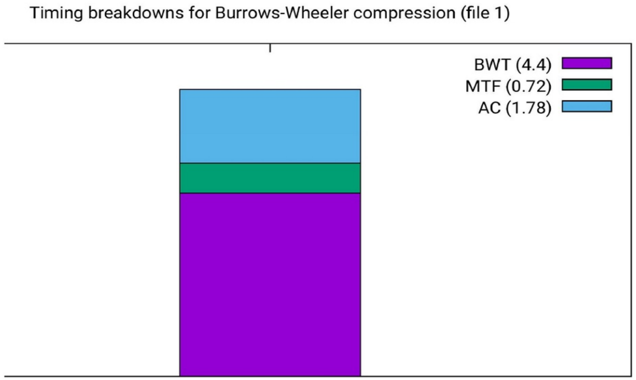
MTF is recency ranking scheme from symbol dictionary, converts to integer set.

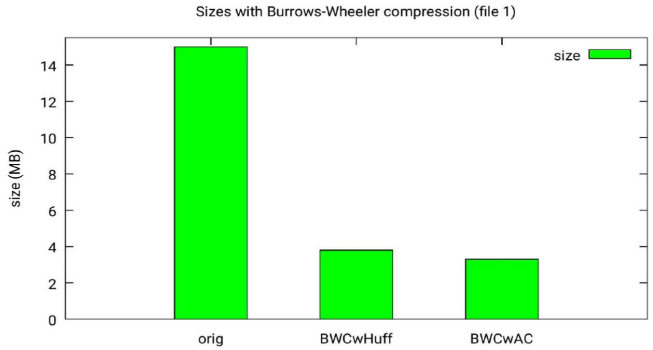
Arithmetic coding represent input by a small interval (or some number within that interval). Better ratio than Huffman trees for heterogeneous inputs.

```
decode_symbol(encoded_val,cum_freq)

    # find symbol such that the following is satisfied

    cum_freq[symbol] <= (encoded_val - low)/(high - low) < cum_freq[symbol-1]

    range = high - low

    high  = low + range*cum_freq[symbol-1] ; low   = low + range*cum_freq[symbol]

    return x[symbol]
```

	[0.0,	1.0)
"c"	[0.0,	0.1)
"o"	[0.03,	0.05)
"m"	[0.034,	0.036)
"p"	[0.0350,	0.0352)
"r"	[0.03512,	0.03516)
"e"	[0.035124,	0.035128)
"s"	[0.0351272,	0.0351280)
"s"	[0.03512764,	0.03512800)
"o"	[0.035123748,	0.035127820)
"r"	[0.0351239912,	0.0351239056)

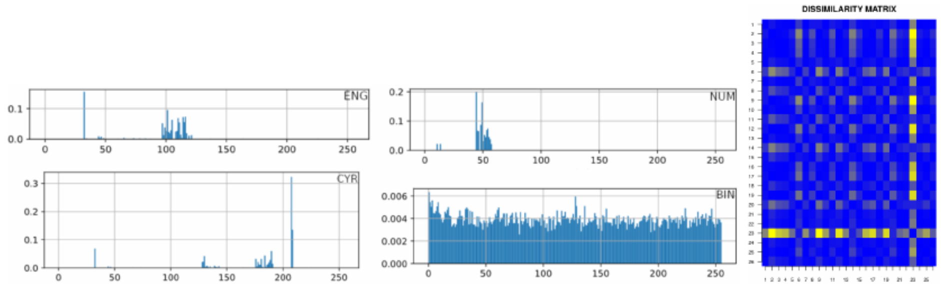
Best done on larger chunks. Relatively slow. Need cumulative frequency count of encountered symbols; best done adaptively.

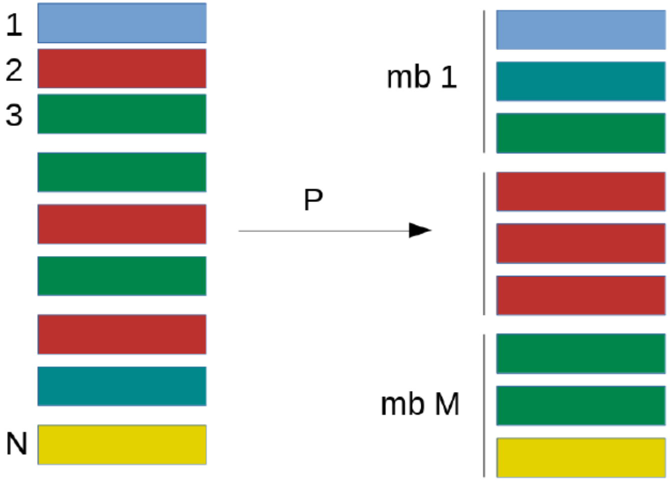




Implemented enhancements for parallel compressor:

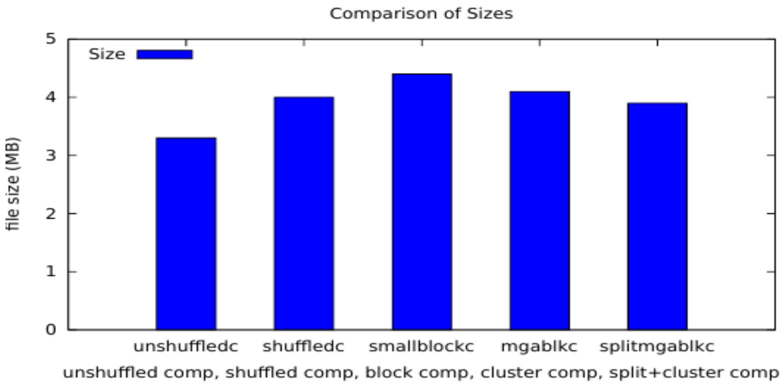
1) Subdivision into mega-blocks via symbol distribution clustering.





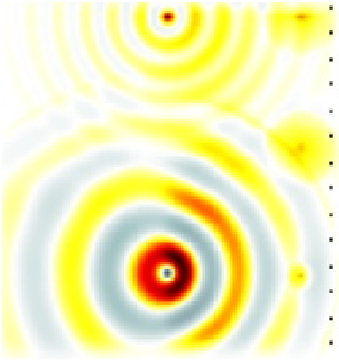
2) O(n) counting sort with indexing permutation output.

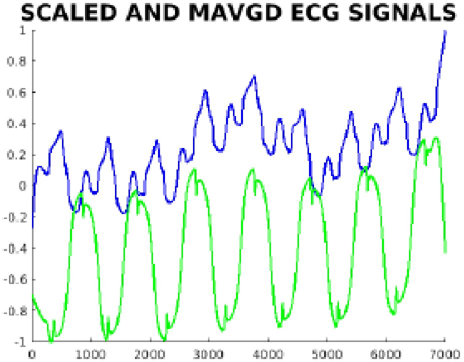
```
struct val_inds      before sort:
{                    5  4  2  1  1  3  4  12  10
    int val;         after sort:
    int num_inds;    1  1  2  3  4  4  5  10  12
    int *inds;       inds after sort:
};                   3  4  2  5  1  6  0  8  7
```



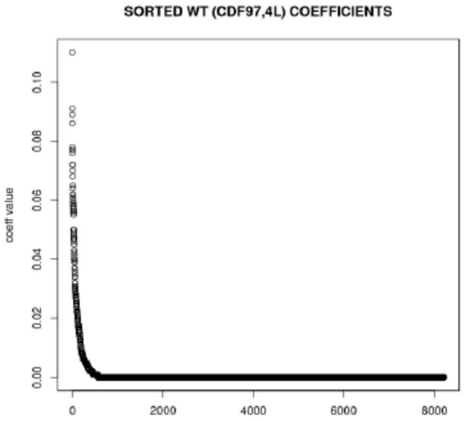

```
                                  nbucket = 1, cur =  (10)
bananasale        ale             nbucket = 2, cur = ananasale (97)
ananasale         ananasale       nbucket = 2, cur = anasale (97)
nanasale          anasale         nbucket = 2, cur = asale (97)
anasale           asale           nbucket = 2, cur = ale (97)
nasale      -->   bananasale      nbucket = 3, cur = bananasale (98)
asale             e               nbucket = 4, cur = e (101)
sale              le              nbucket = 5, cur = le (108)
ale               nanasale        nbucket = 6, cur = nanasale (110)
le                nasale          nbucket = 6, cur = nasale (110)
e                 sale            nbucket = 7, cur = sale (115)
```

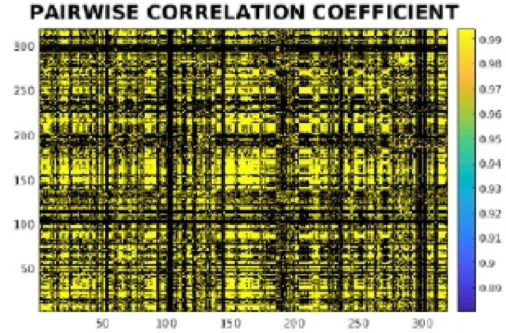
Application (similar signals): microphone array, ecg signals



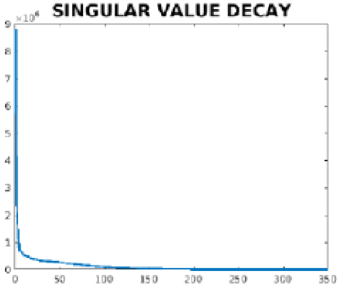


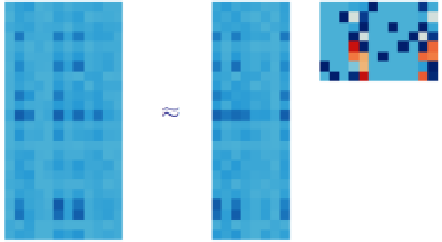
For remaining data, sorted abs values of transformed coefficients are exponentially decaying.






```
sig_ref = log(E(ind,:));
p = polyfit(sig_ref,sig_new,2);
yfit = p(1) * sig_ref.^2 + p(2) * sig_ref + p(3);
```





The ID can be constructed from the partial pivoted QR factorization.

$$A(:,J_c) = m\begin{bmatrix} \overset{k}{Q_1} & \overset{r-k}{Q_2} \end{bmatrix} \times \begin{matrix} k \\ r-k \end{matrix}\begin{bmatrix} \overset{n}{S_1} \\ S_2 \end{bmatrix} = Q_1S_1 + Q_2S_2.$$

$$A(:,J_c) = Q_1\begin{bmatrix} S_{11} & S_{12} \end{bmatrix} + Q_2\begin{bmatrix} 0 & S_{22} \end{bmatrix} = m\begin{bmatrix} \overset{k}{Q_1S_{11}} & \overset{n-k}{Q_1S_{12} + Q_2S_{22}} \end{bmatrix}.$$

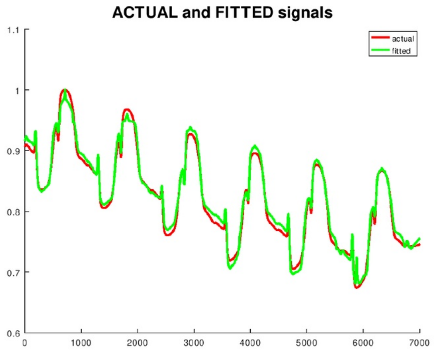
$$Q_1S_1 = \begin{bmatrix} Q_1S_{11} & Q_1S_{12} \end{bmatrix} = Q_1S_{11}[I_k \quad S_{11}^{-1}S_{12}] = C\,[I_k \quad T_l],$$

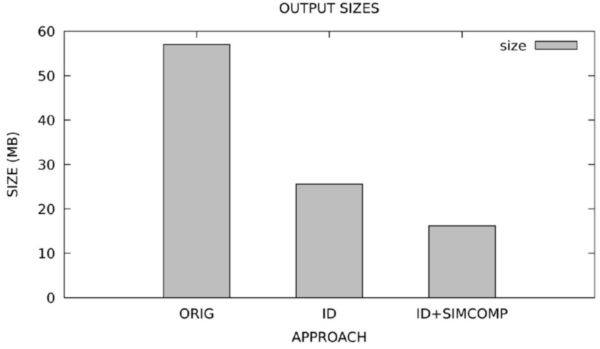
$$A \approx CV^T, \quad \text{where} \quad C = A(:,J_c(1:k)), \quad V^T = \begin{bmatrix} I_k & T_l \end{bmatrix} P^T$$

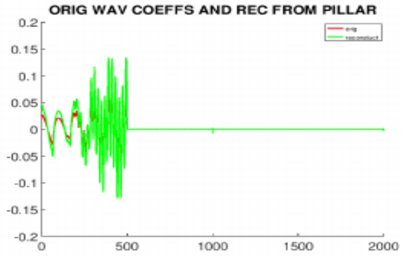
Applied on matrix transpose, yields a subset of the rows.

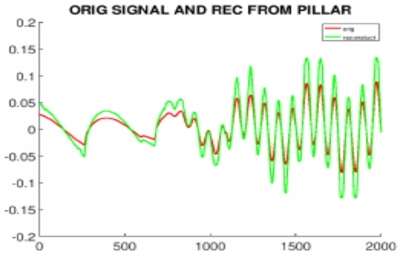
This allows only a portion of the most distinct channel data to be retained. Can then use high correlation modeling for remaining data.

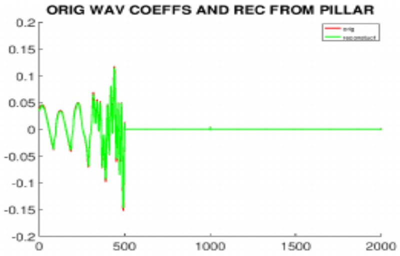
Data: Floating point data from multiple channels. Tolerance and pillar block parameters $(\epsilon_{1,2,3}, L)$, Wavelet transform, and thresholding function.
Result: Compressed representation of data for all channels.
Insert floating point data into matrix A, one channel per row.
Perform ID decomposition on the transpose of the matrix, $A^T \approx A(J_r(1:k),:)V$ with rank chosen per ϵ_1 tolerance.
Set $C = A(J_r(1:k),:)$ to be the subset of retained channels.
Form matrices $\bar{M}, M_{num}, M_{sgn}$ from C.
for $j = 1,\ldots,k$ **do**
  Compute $w_j = transform(C(j,:))$
  $[v_j, I_j] = sort(abs(w_j), 'd')$
  Store permutation inds I_j from sort and signs of $w_j(I_j)$ in $M_{\text{num}}(j,:)$ and $M_{\text{sgn}}(j,:)$.
  Set $\bar{M}(j,:) = Thr(v_j)$ per ϵ_2.
end
Set $M_E = 1e6, i = 0$. Initialize E to hold subset (the pillars) of $\bar{M}$ and F to hold linear fitting information.
while $M_E > \epsilon_3$ **do**
  Add $C(i+1,\ldots,i+L,:)$ to E.
  for $j = i+L+1,\ldots,k$ **do**
    Compute low order polynomial fit model between $\log(\bar{M}(j,:))$ and each of the saved channels $\log(E(i,:))$.
    Record scaling factor s_j, modeling coefficients a, b and index to pillar model corresponding to smallest error against $E(i,:)$ in $F(j,:) = [a, b, si, i]$.
    Record reconstruction error as e_j.
  end
  Let $M_E = \max(e_j), i = i + L$.
end
Lossless compress saved floating point data E, fitting coefficient set F, as well as the integer and bit sign matrices M_{num} and M_{sgn} and ID matrix V.

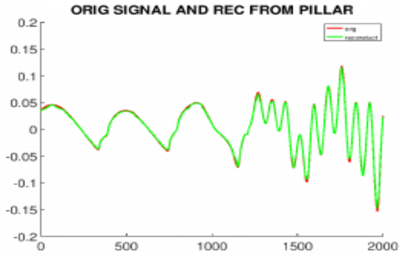




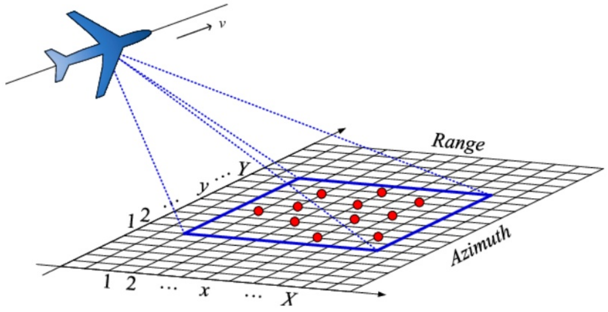


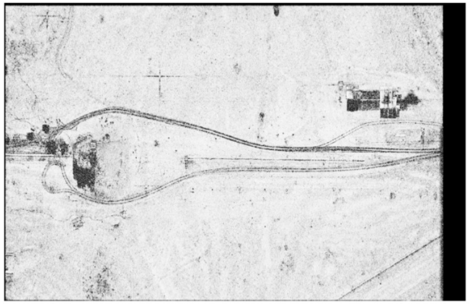


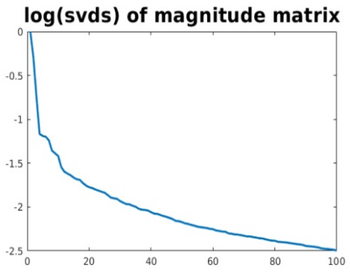


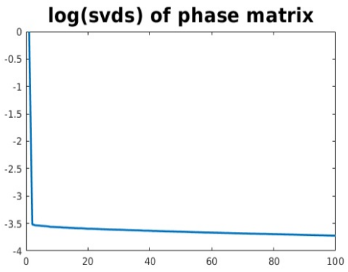


Analysis and compression of SAR data

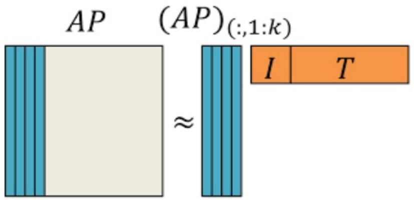








-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	0	0	1	1	1
-1	0	0	1	1	0	0	-1
-1	0	1	1	-1	-1	0	1
-1	1	1	-1	-1	1	1	-1
-1	1	0	-1	1	0	-1	1
0	1	-1	0	0	-1	1	0
0	1	-1	1	-1	1	-1	0

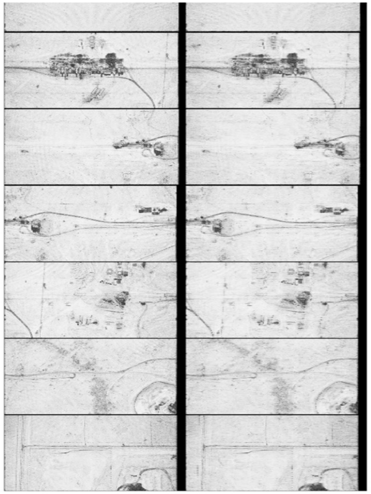


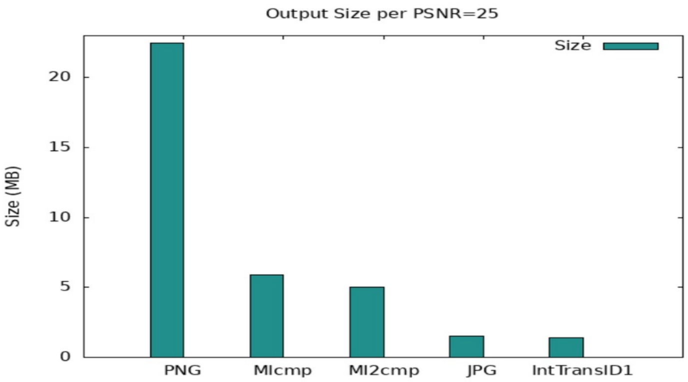
Algorithm 1: SAR BLOCK PNG COMPRESS

Input: A set $C = \{I_1, I_2, \ldots, I_r\}$ of SAR images in PNG (or similar) format, block size $l \times l$ and adaptive tolerance ϵ or rank k.

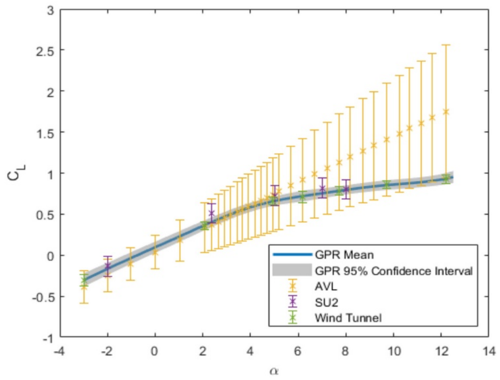
Output: A compressed representation consisting of losslessly compressed ID components and scaling factors.

Break the image pixel set in $l \times l$ blocks for a total of N_r blocks $\{b_i\}$ representing the set.
Initialize transform matrix $T_l = \text{round}\left(\text{dctmtx}[l]/\min(\min(\text{dctmtx}[l]))\right)$.
Apply transform and subtract smallest number from each block.
for $i \leftarrow 1$ ***to*** N_r **do**
  $bt_i = T_l b_i$
  $mv_i = \min(\min(bt_i))$
  $bt_i = bt_i - mv_i$
  $M = [M; bt_i]$
Decompose matrix of transformed blocks $M \approx M(:, I(1:k))Vt$ via pivoted QR factorization to tolerance level ϵ.
Lossless compress remaining ID and scaling factors.





Multi-fidelity with Gaussian Processes



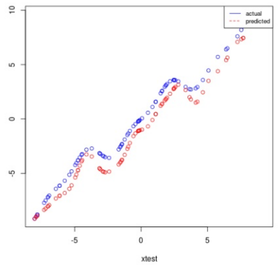
Goal is to comine data with multiple fidelities (from simulations, testing) to build databases for aerodynamic modeling. Using Gaussian process regression (an interpolation method that pre-supposes a multi-normial Normal distribution on the target data).

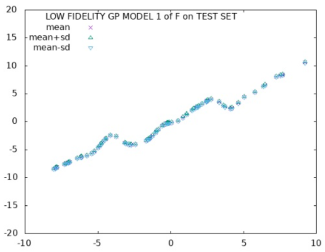
$$p(x|\mu, K) = 1/\text{sqrt}\{\det(2\pi K)\} \exp\left[-\frac{1}{2}(x-\mu)^T K^{\{-1\}}(x-\mu)\right]$$

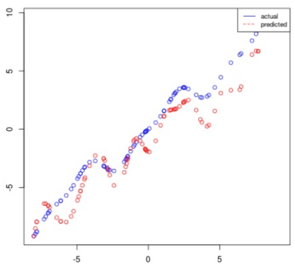
$$[y_{\{test\}}; y_{\{train\}}] \sim N([\mu_1; \mu_2], [\Sigma_{\{11\}}, \Sigma_{\{1,2\}};\ \Sigma_{\{21\}}\Sigma_{\{22\}})$$

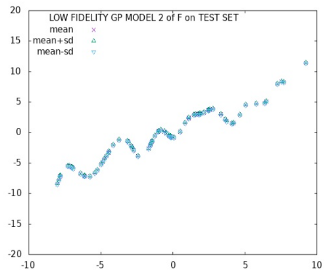
$$y_{\{test\}}|\ y_{\{train\}} \sim N\left(\mu_1 + \Sigma_{\{12\}}\Sigma_{\{22\}}^{\{-1\}}(y_{\{train\}} - \mu_2), \Sigma_{\{11\}}\Sigma_{\{22\}}^{\{-1\}}\Sigma_{\{21\}}\right).$$

$$y_{\{test\}}|\ y_{\{train\}} \sim N[(\ \mathrm{K}_{\{s\}}(K + \lambda I)^{\{-1\}} y_{\{train\}}, \mathrm{K}_{\{ss\}} - \mathrm{K}_s(\mathrm{K} + \lambda\mathrm{I})^{\{-1\}}\mathrm{K}_s^{\mathrm{T}}\ )]$$

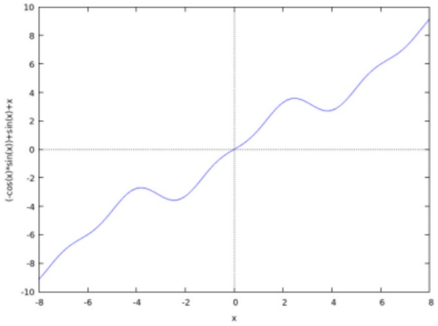




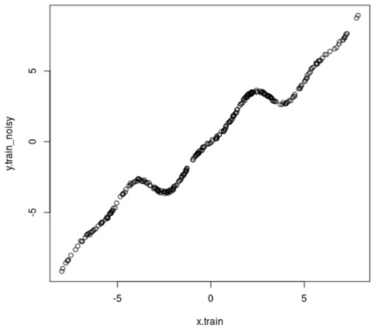




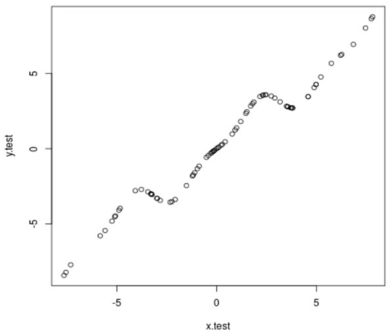
actual



train

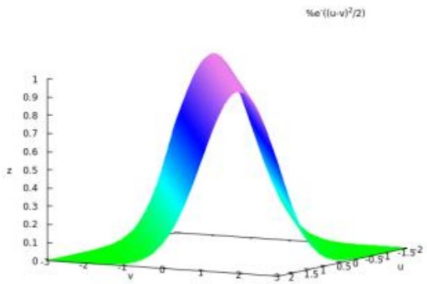


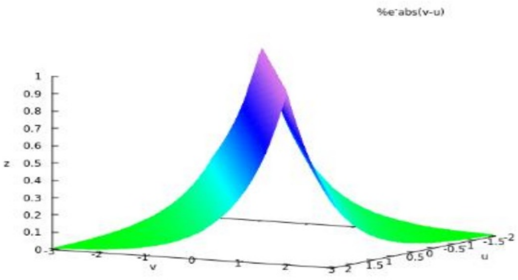
test

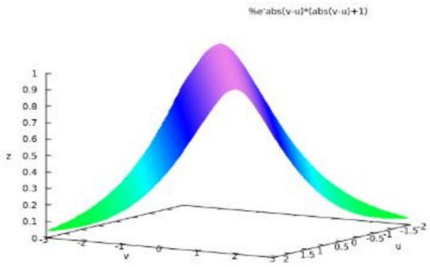


Optimized Gaussian Processes

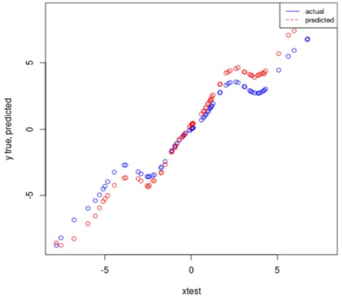
Different covariance ‘kernels’ with associated hyperparameters appropriate per different noise settings.

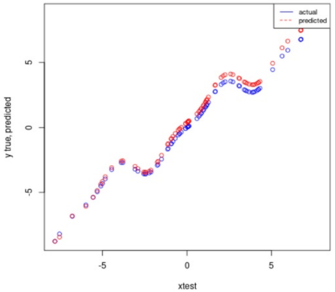






e.g. Sigma[i,j] = a*exp(-b*(x1[i] – x2[j] -c)^d); {a,b,c,d} are the hyperparameters.





Optimization of kernel type and parameters often yields better prediction results.

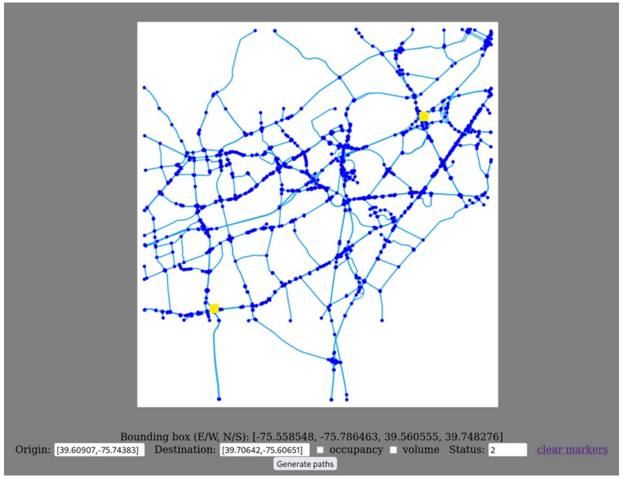
```
for ( k in 1:ncdescent_iters ) {

# pick random num in 1-4 range
coord_to_opt = round(runif(1, 1, 4));

printf(">>>>>>>>>>>>>>>>>>>>>>> iter = %d -> coord_to_opt = %d\n", k, coord_to_opt);
svals1 = c();
svals2 = c();
svals3 = c();
loglikevals = c();

for ( vind in 1:length(as) ) {

if (coord_to_opt == 1){
    a = as[vind]; b = bsave; c = csave; d = dsave;
} else if (coord_to_opt == 2){
    a = asave; b = bs[vind]; c = csave; d = dsave;
} else if (coord_to_opt == 3){
    a = asave; b = bsave; c = cs[vind]; d = dsave;
} else if (coord_to_opt == 4){
    a = asave; b = bsave; c = csave; d = ds[vind];
}

printf("test kernel %d\n", k);
gp = gp_solve( x.train_nt , y.noisy_nt , x.test_nt , kernels[[opt_kernel]] , sigma2e, a, b, c, d )

# compute var sum
var_sums = colMeans(gp[['var']]);
sval1 = sum(var_sums);
print("sum of variances 1:");
sval1 = sum(var_sums);
print(sval1);
svals1 = c(svals1,sval1);

inds = which(x.test>-5 & x.test<5)
var_sums2 = abs(var_sums[inds]);
print("sum of variances 2:");
sval2 = sum(var_sums2)
print(sval2)
svals2 = c(svals2,sval2);

diffs = gp[['mu']] - ytest_true;
diffs = abs(diffs[inds]);
sval3 = sum(diffs);
svals3 = c(svals3,sval3);

loglikeval = get_log_likelihood( x.train_nt , y.noisy_nt , x.test_nt , kernels[[opt_kernel]] , sigma2e = 0, a, b, c, d );
loglikevals = c(loglikevals,loglikeval);

}
```

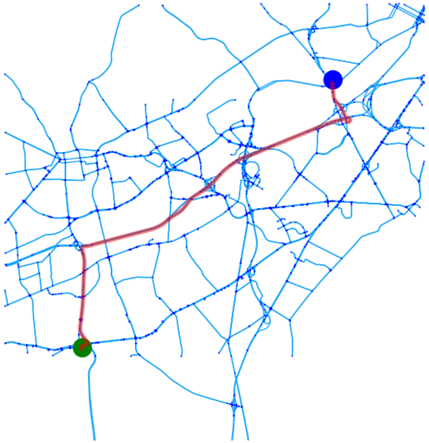
Developed optimized GP code performs several level of optimization to pick the optimal kernel and tune parameters for that kernel based on the train data. Basic approach based on randomized coordinate descent method.

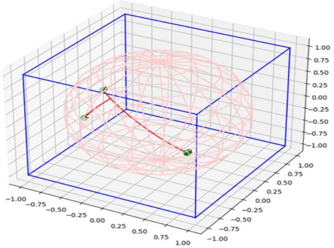
```
## Function to evaluate Log-Likelihood
get_log_likelihood = function( x.train , y.train , x.pred , kernel , sigma2e = 0, a, b, c, d ) {
    k.xx = kernel(x.train,x.train,a,b,c)
    dims = dim(k.xx);
    k.xx = k.xx + d*matrix(runif(dims[1]*dims[2]),dims[1],dims[2]);

    Vinv = solve(k.xx + sigma2e * diag(1, ncol(k.xx)))
    printf("size y.train:\n");
    print(dim(y.train))
    printf("size Vinv:\n");
    print(dim(Vinv))
    tyt = t(y.train);
    printf("size tyt:\n");
    print(dim(tyt))
    val = -0.5*t(y.train)%*%Vinv%*%(y.train) - 0.5*log(norm(k.xx)) - length(y.train)/2*log(2*pi);
    return (val);
}
```

Evaluate model fit based on log likelihood and similar metrics.

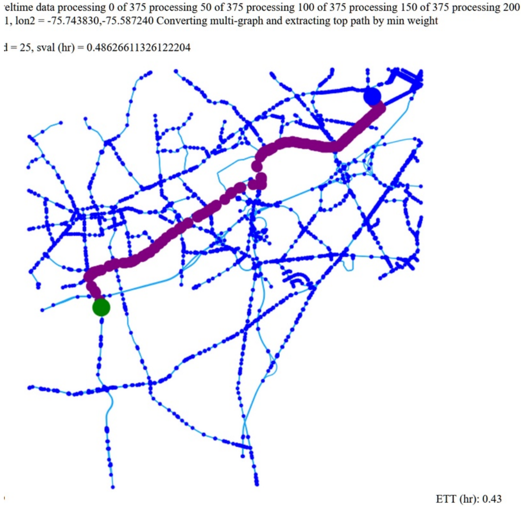
Traffic routing system with weighted graph

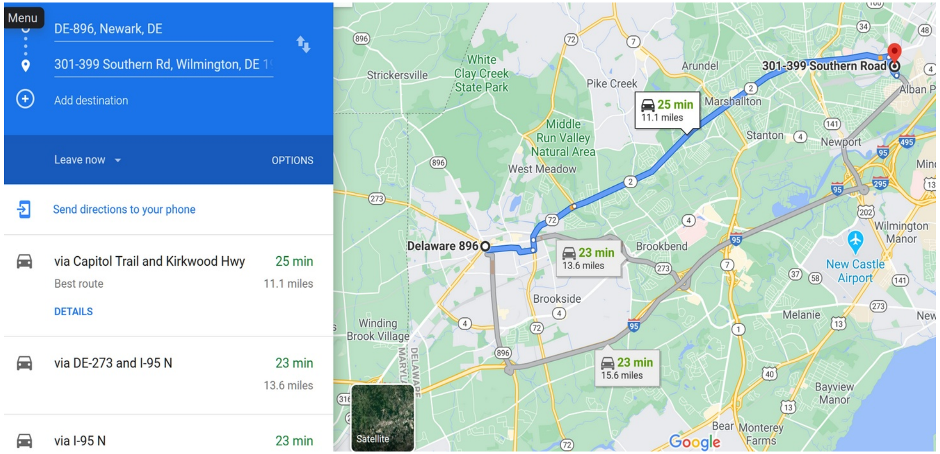






```
M = nx.Graph();
for u,v,data in G.edges(data=True):
    w = data['weight'] if 'weight' in data else 1.0
    if M.has_edge(u,v):
        M[u][v]['weight'] += w;
    else:
        M.add_edge(u,v,weight=w);

paths_unw_shortest = nx.shortest_simple_paths(M, source=orig_node, target=dest_node);
```





Weighted graph based routing approach using near real time road speed data (Delaware DOT).

Accelerated implementation

Goal is to accelerate critical sub-parts of algorithmic implementation to enable application to bigger problem sizes and for faster parameter optimization.

```
// CUDA exp kernel
__global__ void kfunc_exp_kernel(double *x1, double *x2, double *Sigma,
                                 const int M,
                                 const int N,
                                 const double a, const double b, const double c)
{
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  int j = threadIdx.y + blockIdx.y * blockDim.y;
  if(i<M && j<N){
      Sigma[j*M+i] = a*exp(-b*pow(fabs(x1[i]-x2[j]),c));
  }
  return;
}
```

```
// launch threads, one per byte array
pthread_t threadIds[4];
myargs = (struct arg *)malloc(4*sizeof(struct arg)); //array of structs, one per byte array

for(nb=0; nb<4; nb++){
  myargs[nb].byte_num = nb+1;
  myargs[nb].byte_arr = (unsigned char*)malloc(nints_in_file * sizeof(unsigned char));
  if(nb == 0){memcpy(myargs[nb].byte_arr, barr1, nints_in_file);} // or set addr
  if(nb == 1){memcpy(myargs[nb].byte_arr, barr2, nints_in_file);}
  if(nb == 2){memcpy(myargs[nb].byte_arr, barr3, nints_in_file);}
  if(nb == 3){memcpy(myargs[nb].byte_arr, barr4, nints_in_file);}
  ret = pthread_create( &threadIds[nb], NULL, processByteArray, (void *)(&myargs[nb]));
  printf("after calling pthread create with id = %lu..\n", threadIds[nb]);
  if(ret != 0){
    printf( "Error creating thread %lu\n", threadIds[nb] );
  }
}
```

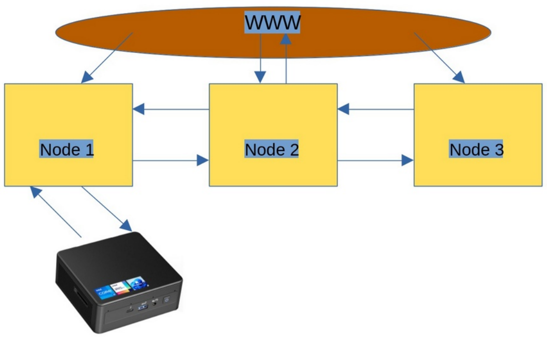
```
void get_idct(double **DCTMatrix, int M, int N){
    int i,j;
    #pragma omp for num_threads(4) collapse(2)
    for (i = 0; i < M; i++) {
        for (j = 0; j < N; j++) {
            if(i==0){
                DCTMatrix[j][i] = (double)(1.0/sqrt((double)N));
            } else{
                DCTMatrix[j][i] = (double)sqrt(2.0/(double)N)*cos((2*j+1)*i*M_PI/(2.0*N));
            }
        }
    }
```

```
automodel = pm.auto_arima(tsdata,
                          start_p=1,
                          start_q=1,
                          test="adf",
                          seasonal=False,
                          trace=True)

ndays_ahead = 14;
tsdata_vals = tsdata.values;
tsdata_preds = automodel.predict(ndays_ahead);
tsdata_combined = np.concatenate((tsdata_vals,tsdata_preds));
date_list = list(df['Datetime'].map(lambda ss:ss.date()))+list((pd.timedelta_range(start='1 day',periods=ndays_ahead)+df.iloc[-1]['Datetime']).map(lambda ss:ss.date()));
```

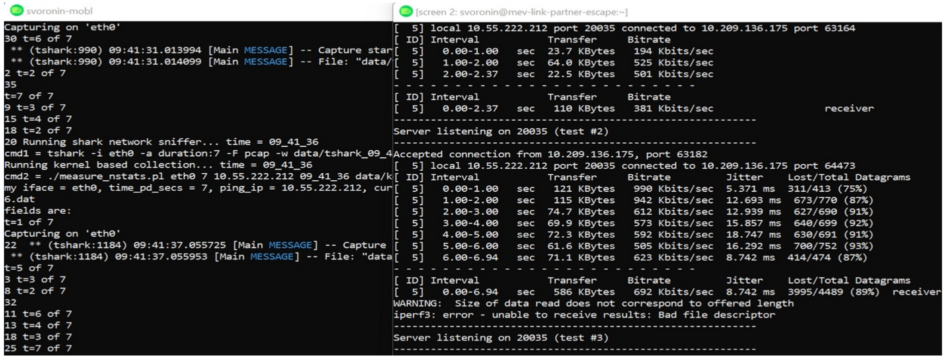
Techniques with P-threads, OpenMP, CUDA, and time series based methods. Example: parallel BWT.

Network data analysis / anomaly detection

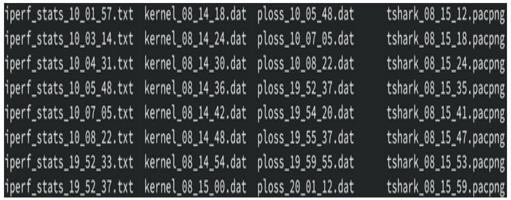
A web or device-based service takes several rounds of network data which can be collected with a simple Linux based device, and text-based instructions, performs analysis on each uploaded data segment and creates outputs based on the user supplied instructions. This service can detect anomalies and changes in usage on a network.



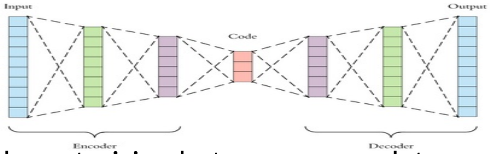
Sample three node setup with network data collection and processing device.



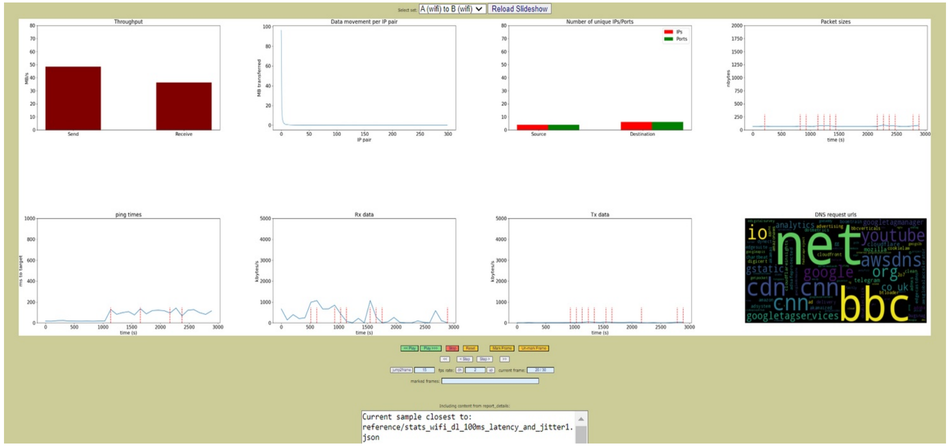
Server or mobile processing unit performs analysis on each data batch, following supplied instructions.



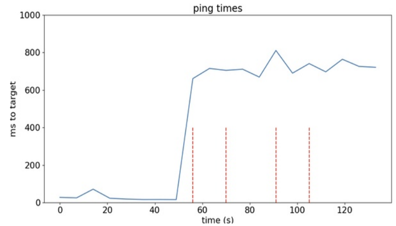
Heterogeneous data bundle from small analysis interval.



Feature mining for characterizing heterogeneous data bundles with statistics and autoencoder.



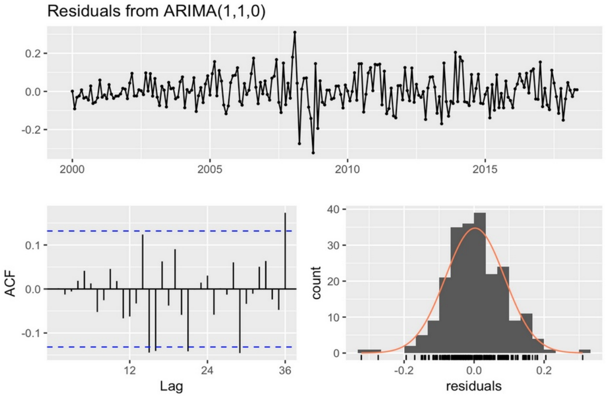
Based on the analysis, the processing node generates graphical output for each batch and for a collective multi-batch view, to allow domain experts to view results for outlier batches.



Change point / breakpoint detection in time series data (e.g. latency).

Time series analysis / prediction

Many available methods for interpolation and prediction. Libraries in Python and R. Examples are ARIMA based codes and machine learning models (e.g. LSTM).



Autoregression, differencing, moving average.

$$AR(p) : Y_t = c + \sum_{i=1}^{p} \phi_i Y_{t-i} + \epsilon_t$$

Autoregression defines current value of series in terms of previous p lags.

$$MA(q) : Y_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

Moving average captures patterns in residual terms.

$$ARMA(p,q) : Y_t = c + \sum_{i=1}^{p} \phi_i Y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t$$

Combination of AR and MA processes.

$$ARIMA(p,d,q) : Y_d = c + \sum_{i=1}^{p} \phi_i Y_{d-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t$$

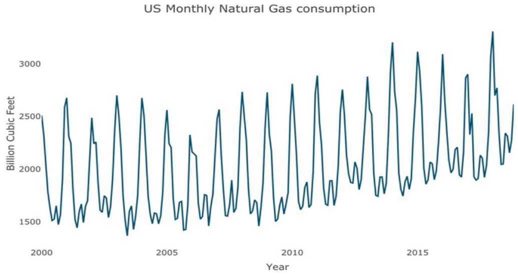
Many statistical choices for modeling univarate series.

ARIMA: A p-order AR process, d-degrees of differencing, and q-order MA process. No simple extension to multivariate case. Can use relatively simple VAR model instead:

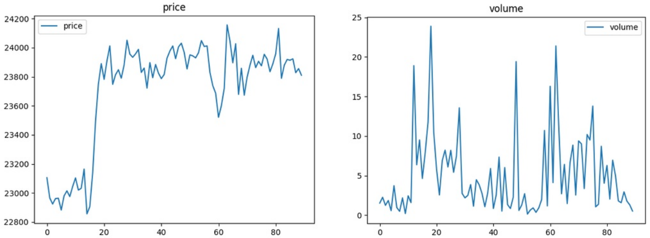
$$Y_{1,t} = \alpha_1 + \beta_{11,1} Y_{1,t-1} + \beta_{12,1} Y_{2,t-1} + \epsilon_{1,t}$$
$$Y_{2,t} = \alpha_2 + \beta_{21,1} Y_{1,t-1} + \beta_{22,1} Y_{2,t-1} + \epsilon_{2,t}$$

$$Y_{1,t} = \alpha_1 + \beta_{11,1} Y_{1,t-1} + \beta_{12,1} Y_{2,t-1} + \beta_{11,2} Y_{1,t-2} + \beta_{12,2} Y_{2,t-2} + \epsilon_{1,t}$$
$$Y_{2,t} = \alpha_2 + \beta_{21,1} Y_{1,t-1} + \beta_{22,1} Y_{2,t-1} + \beta_{21,2} Y_{1,t-2} + \beta_{22,2} Y_{2,t-2} + \epsilon_{2,t}$$

```
model = VAR(price_and_vol)
model.information
model.neqs
model_fit = model.fit()
model_fit.coefs
pred = model_fit.forecast(model_fit.endog, steps=20)
```



Time series with seasonal and trend patterns, can be handled with decompositions.



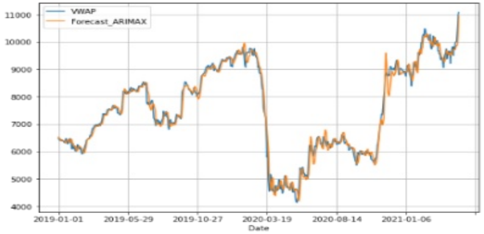

```
X, y = create_sequences(scaled_data, seq_length, steps_ahead, feature_index)
print(f"Shapes after creating sequences - X: {X.shape}, y: {y.shape}")

split = int(0.8 * len(X))
X_train, X_val = X[:split], X[split:]
y_train, y_val = y[:split], y[split:]

input_layer = Input(shape=(seq_length, X.shape[2]))
lstm_units = hp.Int('lstm_units', min_value=32, max_value=128, step=16)
lstm_out = LSTM(lstm_units, return_sequences=True)(input_layer)
lstm_out = Dropout(hp.Float('dropout_rate', min_value=0.1, max_value=0.5, step=0.1))(lstm_out)
lstm_out = LSTM(lstm_units, return_sequences=True)(lstm_out)
lstm_out = Dropout(hp.Float('dropout_rate', min_value=0.1, max_value=0.5, step=0.1))(lstm_out)

# Attention mechanism
attention = Attention()([lstm_out, lstm_out])
attention_out = Concatenate()([lstm_out, attention])

# Flatten the output before dense layers
flat = Flatten()(attention_out)

# Ensure the dense layer output size matches steps_ahead
dense_units = hp.Int('dense_units', min_value=32, max_value=128, step=16)  # Dynamically set dense units
dense_out = Dense(dense_units, activation='relu')(flat)

# Reshape the output to match steps_ahead and features
output = Dense(steps_ahead)(dense_out)
output = Reshape((steps_ahead, 1))(output)  # Predicting 1 feature (log_returns) for steps_ahead

model = Model(inputs=input_layer, outputs=output)
```

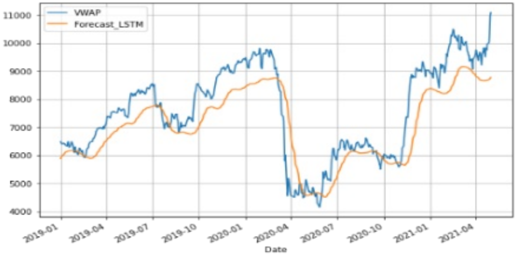
Limited statistical methods for multivariate case drive interest towards machine learning approaches (e.g. LSTM), but there are challenges with optimal data formatting, model parameter optimization, and mechanism for multi-step ahead prediction.

Multivariate time series predictions

Summary: For multivariate cases (where there are two or more series; for instance, price and volume data in finance, or medical pulse and oxygen saturation data), there are fewer available tools. Variance autoregressive models (VAR) are most common from statistics. There is a need for more advanced and efficient methods for different applications.

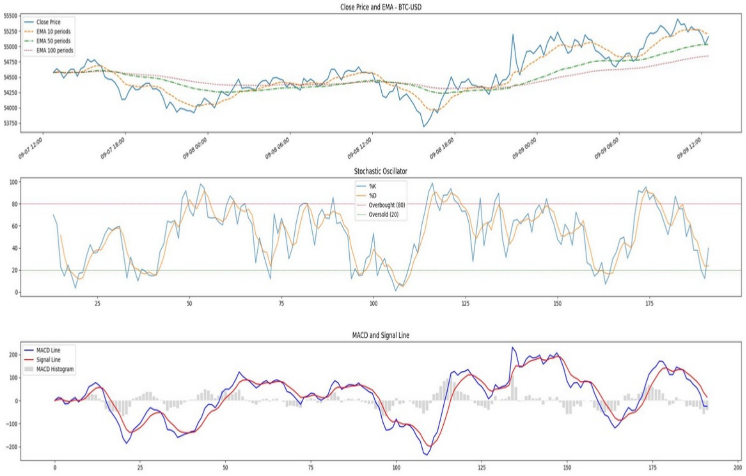
```
from sklearnex import patch_sklearn
patch_sklearn()
```






```
[[1.035e+02 1.300e+06]
 [1.028e+02 1.100e+06]
 [1.051e+02 1.400e+06]]

[[1.028e+02 1.100e+06]
 [1.051e+02 1.400e+06]
 [1.043e+02 1.500e+06]]

[[102.8 105.1 104.3]
[105.1 104.3 106. ]
```

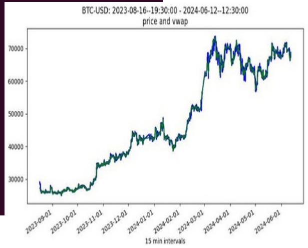
Multi-dim data bundles

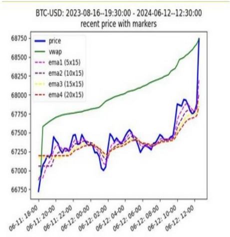


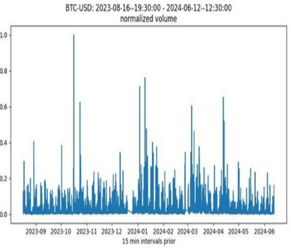
Statistical indicators and AI-based interpretation.

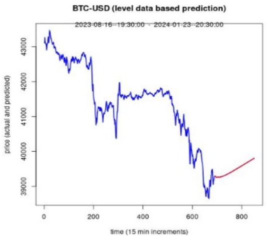
Statistical (many steps ahead) and machine learning based prediction approaches (smaller number of steps ahead, and re-train, re-run approach).

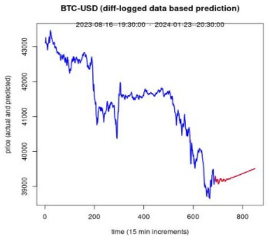
```
loglikeval = get_log_likelihood( x.train , y.train , x.test , kfunc_exp, sigma2e, a, b, c, d );
loglikevals = c(loglikevals,loglikeval);

}


if (coord_to_opt == 1){
    asave = as[which.max(loglikevals)]; b = bsave; c = csave; d = dsave;
} else if (coord_to_opt == 2){
    a = asave; bsave = bs[which.max(loglikevals)]; c = csave; d = dsave;
} else if (coord_to_opt == 3){
    a = asave; b = bsave; csave = cs[which.max(loglikevals)]; d = dsave;
}

# end descent loop
```

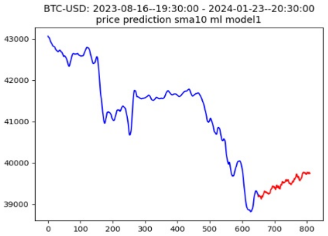













```
input_layer = Input(shape=(seq_length, X.shape[2]))
lstm_units = hp.Int('lstm_units', min_value=32, max_value=128, step=16)
lstm_out = LSTM(lstm_units, return_sequences=True)(input_layer)
lstm_out = Dropout(hp.Float('dropout_rate', min_value=0.1, max_value=0.5, step=0.1))(lstm_out)
lstm_out = LSTM(lstm_units, return_sequences=True)(lstm_out)
lstm_out = Dropout(hp.Float('dropout_rate', min_value=0.1, max_value=0.5, step=0.1))(lstm_out)

# Attention mechanism
attention = Attention()([lstm_out, lstm_out])
attention_out = Concatenate()([lstm_out, attention])
```

LSTM with Bayesian parameter optimization.

Select References

Voronin, et. al., Compression approaches for the regularized solutions of linear systems from large-scale inverse problems, 2015.

Martinsson, Voronin. A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices, 2016.

Thomison, et. al. A Model Reification approach to Fusing Information from Multifidelity Information Sources, 2017.

Stankovic, et. al. On a Gradient-Based Algorithm for Sparse Signal Reconstruction in the Signal/Measurements Domain, 2016.

Voronin, et. al., Multi-resolution classification techniques for PTSD detection, 2018.

Voronin, Mutlti-Channel similarity based compression, 2020.

Voronin et. al., Clustering and presorting for Burrows Wheeler based compression, 2021.

Voronin, SAR image compression with int-int transforms, dimension reduction, 2022.