Sparsity Detection

Beyond sparse matrices and Principle Component Analysis (PCA)!

```
1 using PlutoUI
```

Table of Contents


```
TableOfContents()
```

Information

A measure of randomness, usually measured by the entropy

$$S = -\sum_k p_k \log p_k.$$

Quiz: Which knowledge bellow removes more information?

1. When I toss a coin, its head side will be up,
2. Tomorrow will rain,
3. Today's lecture will be a successful one.

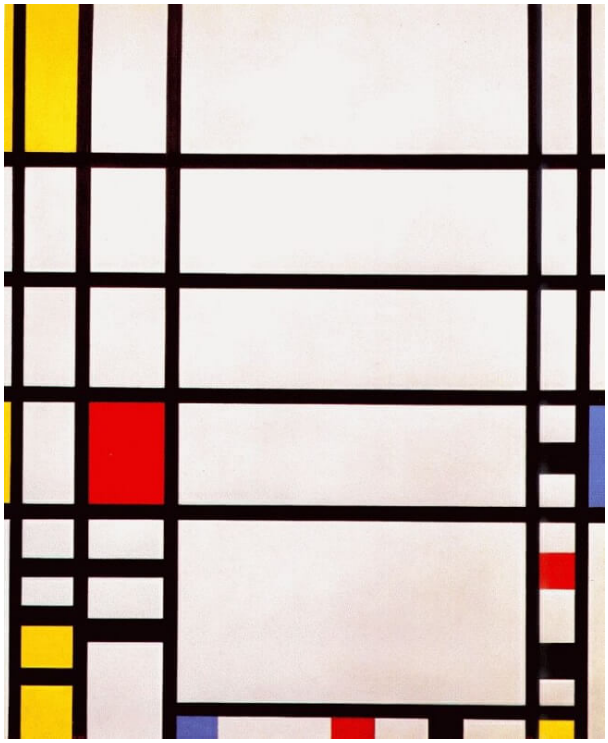
Huffman coding

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding or using such a code proceeds by means of Huffman coding.

Ref: https://www.programiz.com/dsa/huffman-coding

Task: describe the following image in computer.

Mondrian - Trafalgar Square, 1939-43 - a picture having little information from various perspective.



The naive approach

- R: 000
- Y: 001
- B: 010
- K: 011
- W: 100

We need $3mn$ bits to store this image. Can we do better?

Observation

Calculate the frequency of each color in the image.

- R: 3%
- Y: 7%
- B: 1%
- K: 10%
- W: 79%

Formalized description

Input

Alphabet $A = (a_1, a_2, \ldots, a_n)$, which is the symbol alphabet of size n. Tuple $W = (w_1, w_2, \ldots, w_n)$, which is the tuple of the (positive) symbol weights (usually proportional to probabilities), i.e. $w_i = \mathbf{weight}\,(a_i)$, $i \in \{1, 2, \ldots, n\}$.

Output

Code $C\,(W) = (c_1, c_2, \ldots, c_n)$, which is the tuple of (binary) codewords, where c_i is the codeword for a_i, $i \in \{1, 2, \ldots, n\}$.

Goal

Let $L\,(C\,(W)) = \sum_{i=1}^{n} w_i\,\mathbf{length}\,(c_i)$ be the weighted path length of code C. Condition: $L\,(C\,(W)) \leq L\,(T\,(W))$ for any code $T\,(W)$.

Algorithm

The simplest construction algorithm uses a priority queue where the node with lowest probability is given highest priority:

1. Create a leaf node for each symbol and add it to the priority queue.
2. While there is more than one node in the queue:
    1. Remove the two nodes of highest priority (lowest probability) from the queue
    2. Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
    3. Add the new node to the queue.
3. The remaining node is the root node and the tree is complete.

Since efficient priority queue data structures require $O(\log n)$ time per insertion, and a tree with n leaves has $2n - 1$ nodes, this algorithm operates in $O(n \log n)$ time, where n is the number of symbols.

Implementation

Build a huffman tree

```
struct Node{VT, PT}
    value::Union{VT,Nothing}
        prob::PT
    left::Union{Node{VT,PT}, Nothing}
    right::Union{Node{VT,PT}, Nothing}
end
```

```
using DataStructures
```

```
huffman_tree (generic function with 1 method)
```

```
function huffman_tree(symbols, probs)
        isempty(symbols) && error("empty input!")
        # priority queue can keep the items ordered with log(# of items) effort.
        nodes = PriorityQueue(Base.Order.Forward,
            [Node(c, f, nothing, nothing)=>f for (c, f) in zip(symbols, probs)])
    while length(nodes) > 1
        left = dequeue!(nodes)
        right = dequeue!(nodes)
        parent = Node(nothing, left.prob + right.prob, left, right)
        enqueue!(nodes, parent=>left.prob + right.prob)
    end
        return dequeue!(nodes)
end
```

```
ht =
  Node(nothing, 1.0, Node(nothing, 0.21000000000000002, Node('K': ASCII/Unicode U+004B (
```

```
ht = huffman_tree("RYBKW", [0.03, 0.07, 0.01, 0.1, 0.79])
```

From the tree, we generate the binary code.

```
decent! (generic function with 3 methods)
```

```
function decent!(tree::Node{VT}, prefix::String="", d::Dict = Dict{VT,String}())
where VT
        if tree.left === nothing # leaft
            d[tree.value] = prefix
        else   # non-leaf
            decent!(tree.left, prefix*"0", d)
            decent!(tree.right, prefix*"1", d)
        end
        return d
end
```

```
code_dict =
  Dict('K' => "00", 'Y' => "011", 'R' => "0101", 'W' => "1", 'B' => "0100")
```

```
code_dict = decent!(ht)
```

```
mean_code_length = 1.36
```

```
mean_code_length = let
        code_length = 0.0
        for (symbol, prob) in zip("RYBKW", [0.03, 0.07, 0.01, 0.1, 0.79])
            code_length += length(code_dict[symbol]) * prob
        end
        code_length
end
```

We only need $1.36mn$ bits to represent the Mondrian's Trafalgar Square!

The optimality

Lemma: Huffman Encoding produces an optimal tree.

The compressed text has a minimum size of Sn, where

$$S = -\sum_k p_k \log p_k.$$

It is reached when all non-leaf nodes in the tree are ballanced, i.e. having the same weight for left and right children.

```
S_trafalgar = 1.0876128696604102
```

```
S_trafalgar = StatsBase.entropy([0.03, 0.07, 0.01, 0.1, 0.79], 2)
```

Matrix Product State/Tensor Train

Calculate the compression ratio.

Compressed Sensing

Reference: https://www.pyrunner.com/weblog/B/index.html

Example 1: Two sinusoids

```
import FFTW
```

Let us define a function of adding two sinusoids.

```
n = 5000
```

```
# sum of two sinusoids
n = 5000
```

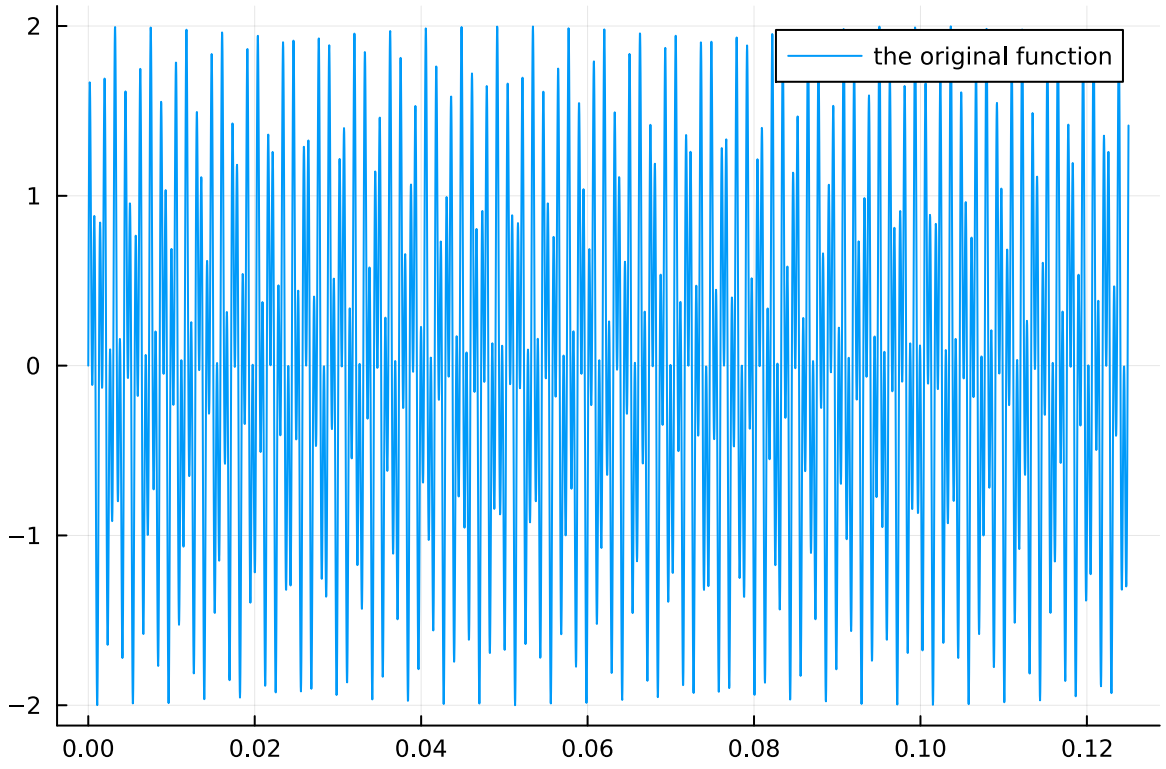
```
t =
5000-element LinRange{Float64, Int64}:
 0.0,2.5005e-5,5.001e-5,7.5015e-5,0.00010002,…,0.1249,0.124925,0.12495,0.124975,0.125
```

```
# time sequence
t = LinRange(0, 1/8, n)
```

```
y =
  [0.0, 0.363045, 0.708168, 1.01855, 1.27951, 1.4794, 1.61028, 1.66841, 1.65442, 1.57322,
```

```
# the function
y = sin.(1394π .* t) + sin.(3266π .* t)
```

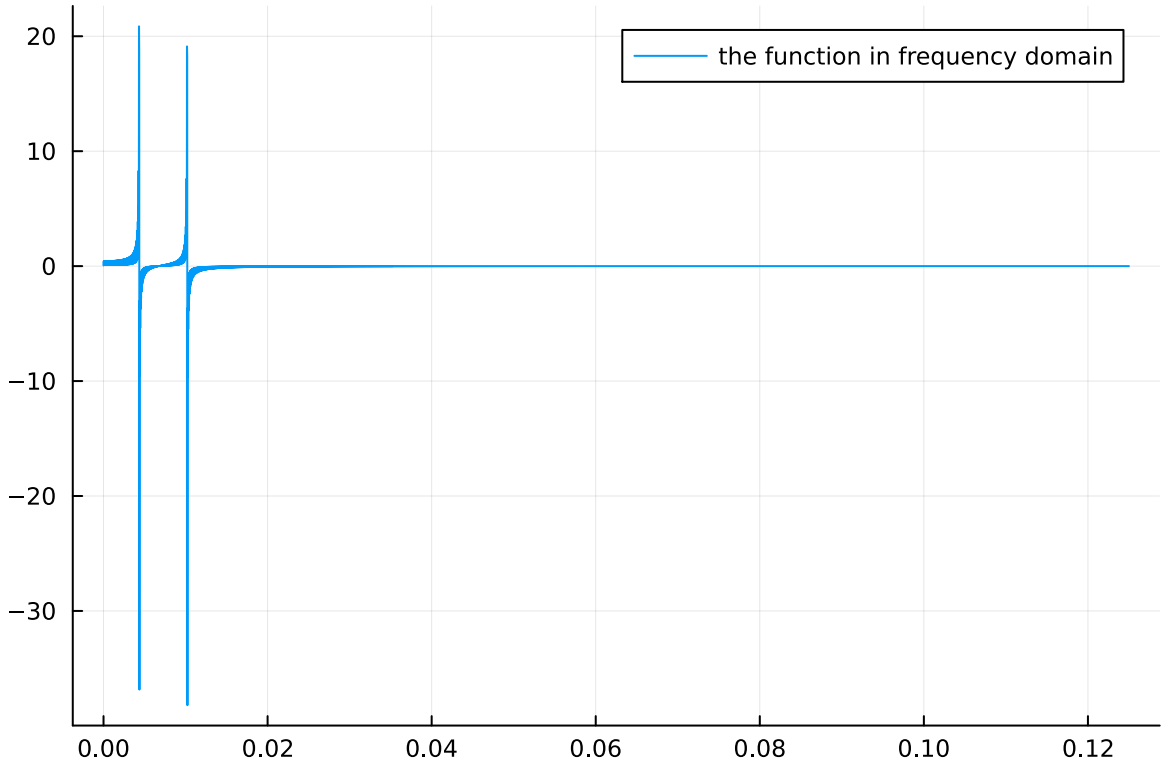
```
using Plots
```




```
plot(t, y; label="the original function")
```

```
yt =
  [0.0638438, 0.429684, 0.0902974, 0.42977, 0.0903235, 0.429942, 0.0903669, 0.430201, 0.0
```

```
# the function in the spectrum domain
yt = FFTW.dct(y)
```




```
plot(t, yt; label="the function in frequency domain")
```

Let us extract 10% samples from it.

```
m = 500
```

```
m = 500
```

```
using StatsBase
```

```
samples =
 [2, 5, 24, 30, 41, 75, 83, 88, 127, 140, 141, 181, 205, 211, 217, 224, 227, 235, 237, 249
```

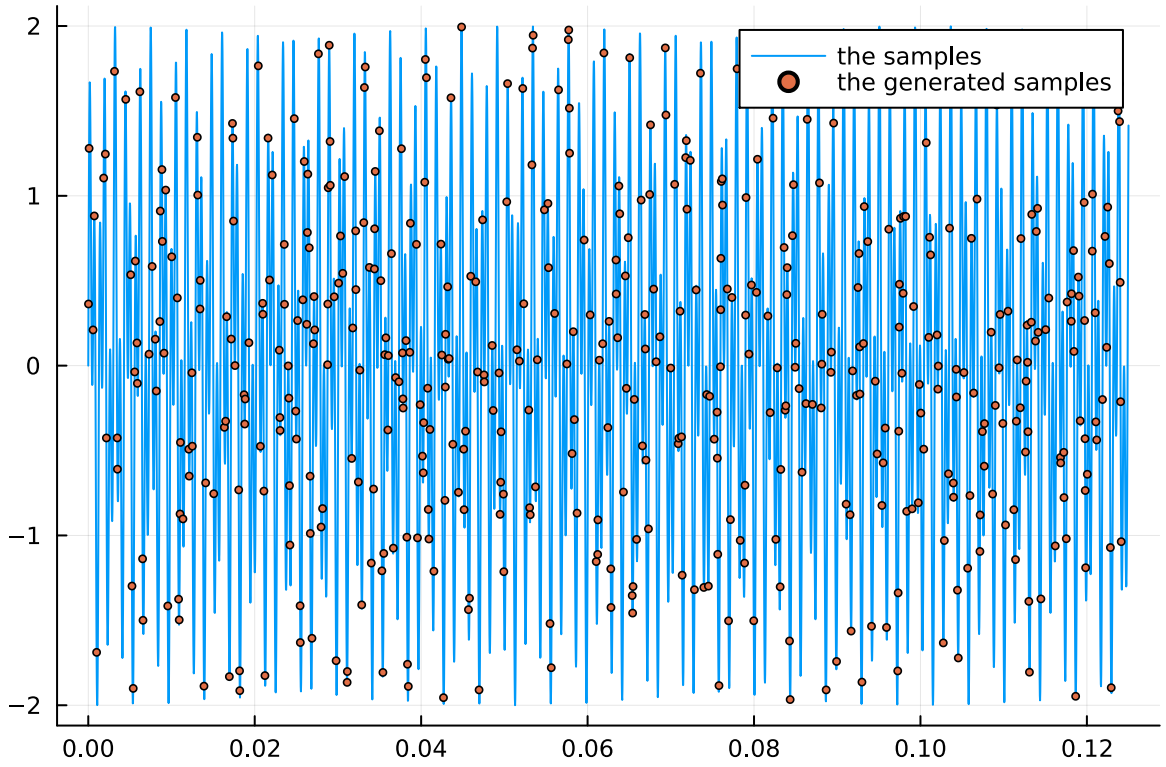
```
# not allowing repeated indices
samples = sort!(StatsBase.sample(1:n, m, replace=false))
```

```
t2 =
 [2.5005e-5, 0.00010002, 0.000575115, 0.000725145, 0.0010002, 0.00185037, 0.00205041, 0
```

```
t2 = t[samples]
```

```
y2 =
 [0.363045, 1.27951, 0.21042, 0.881564, -1.68847, 1.10468, 1.24586, -0.425932, 1.73304,
```

```
y2 = y[samples]
```



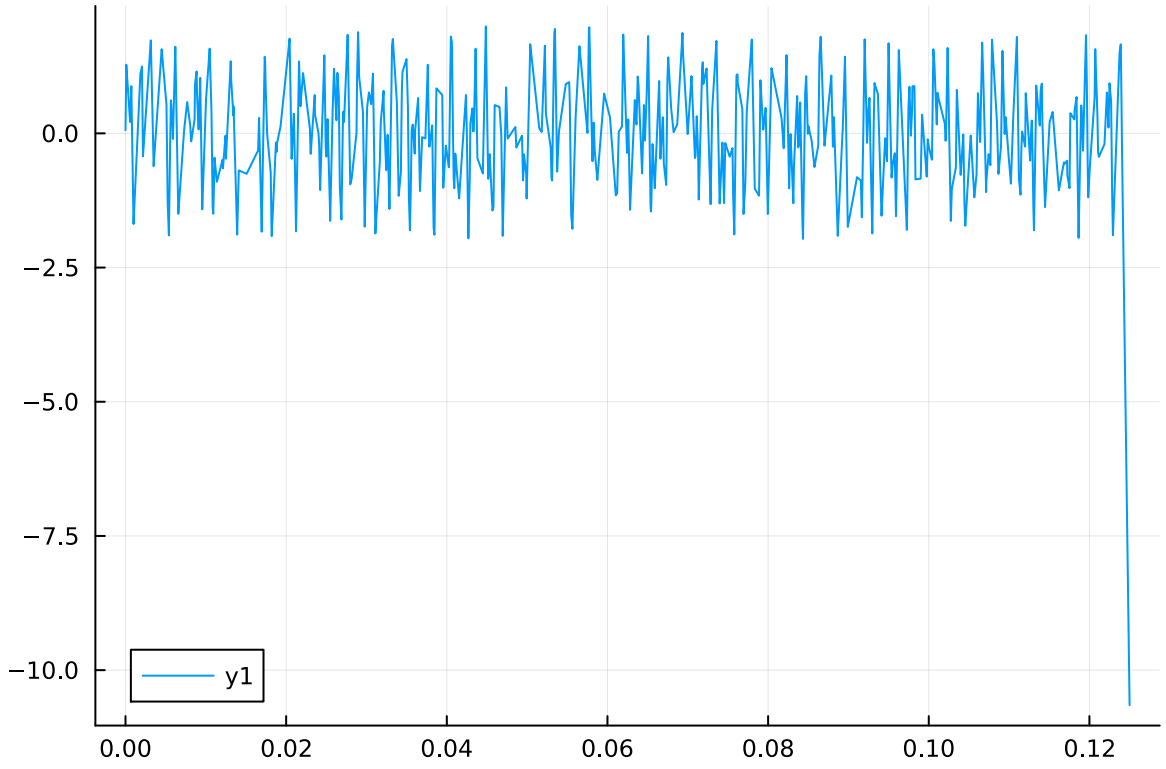

```
let
        plt = plot(t, y; label="the samples")
        scatter!(plt, t2, y2; label="the generated samples", markersize=2)
end
```

If we plot it directly, it looks not so good

```
using Interpolations
```

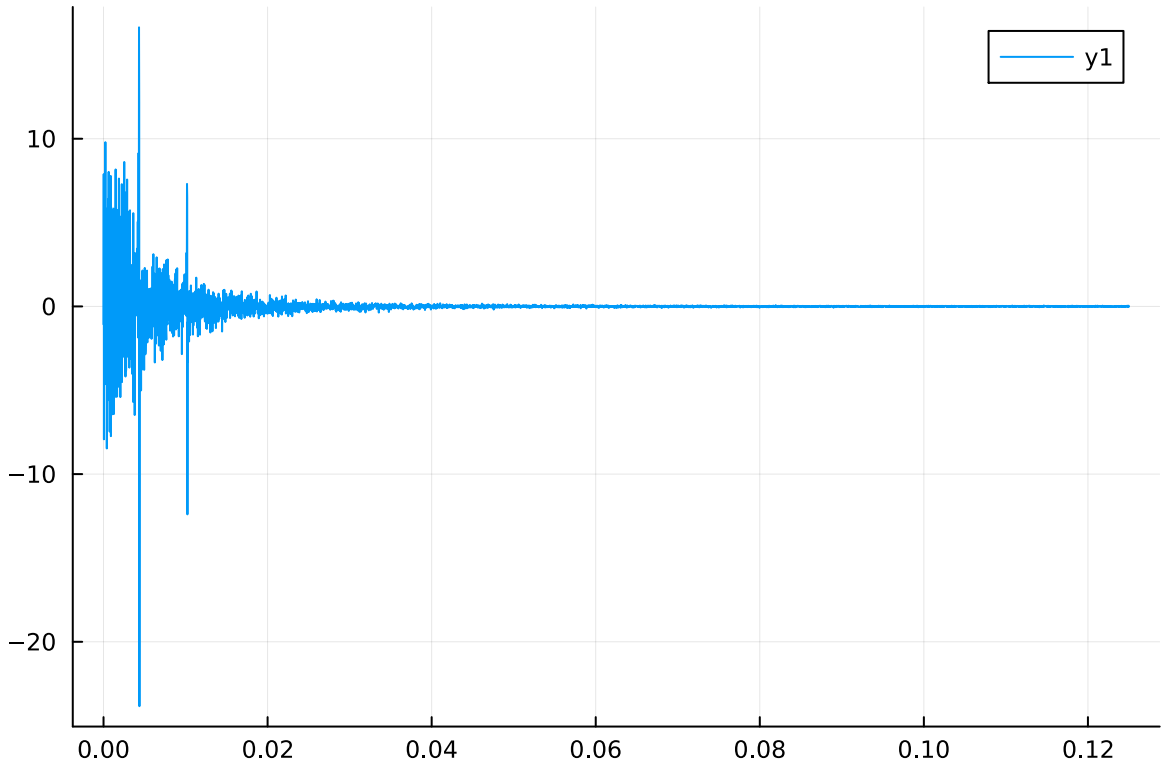
```
interp_linear =
  extrapolate(interpolate((::Vector{Float64},), ::Vector{Float64}, Gridded(Linear())),
```

```
interp_linear = linear_interpolation(t2, y2; extrapolation_bc=Line())
```




```
1 plot(t, interp_linear.(t))
```

Why? Because we haven't used a prior that it is sparse in the frequency domain.




```
1 plot(t, FFTW.dct(interp_linear.(t)))
```

Instead, we rephrase the problem as the following convex optimization problem

$$\min \sum_i |x_i|$$
$$s.t.\ Ax = b$$

```
using JuMP, SCS
```

```
model = A JuMP Model
        Feasibility problem with:
        Variables: 0
        Model mode: AUTOMATIC
        CachingOptimizer state: EMPTY_OPTIMIZER
        Solver name: SCS
```

```
model = Model(SCS.Optimizer)
```

```
using LinearAlgebra
```

```
A = 500×5000 Matrix{Float64}:
     0.0141421   0.02       0.02        0.0199999  …  -3.76991e-5   -1.88496e-5
     0.0141421   0.0199999  0.0199997   0.0199993      0.000113097   5.65486e-5
     0.0141421   0.0199978  0.0199913   0.0199804     -0.000590534  -0.000295299
     0.0141421   0.0199966  0.0199863   0.0199691     -0.000741246  -0.000370687
     0.0141421   0.0199935  0.0199741   0.0199417      0.00101744    0.000508883
     0.0141421   0.0199781  0.0199124   0.0198031  …   0.00186966    0.000935853
     0.0141421   0.0199731  0.0198926   0.0197587      0.00206974    0.00103626
     ⋮                                              ⋱
     0.0141421  -0.0199899  0.0199597  -0.0199095      0.00126835   -0.000634495
     0.0141421  -0.0199922  0.0199687  -0.0199297  …   0.00111782   -0.000559131
     0.0141421  -0.0199925  0.0199701  -0.0199328     -0.00109273    0.000546569
     0.0141421  -0.0199935  0.0199741  -0.0199417      0.00101744   -0.000508883
     0.0141421  -0.0199941  0.0199766  -0.0199474      0.000967233  -0.000483758
     0.0141421  -0.019995   0.0199801  -0.0199552     -0.000891916   0.000446069
```

```
A = FFTW.idct(Matrix{Float64}(I, n, n), 1)[samples, :]
```

```
# do L1 optimization
@variable model x[1:n];
```

```
@variable(model, norm1);
```

```
@constraint model A * x .== y2;
```

We use l_1 norm because l_0 is very hard to optimize.

```
@constraint(model, [norm1; x] in MOI.NormOneCone(1 + length(x)));
```

```
@objective(model, Min, norm1);
```

☐

```
@bind run_optimize CheckBox()
```

```
if run_optimize
        optimize!(model)
        plot([JuMP.value.(x), FFTW.idct(JuMP.value.(x))]; layout=(2, 1), xlim=(0, 1000), labels=["spectrum", "recovered"])
end
```

```
# A*JuMP.value.(x) - y2
```

```
[-8.88178e-16, 1.11022e-15, 1.47105e-15, -3.33067e-16, -3.33067e-15, -4.44089e-16, 1.3
```

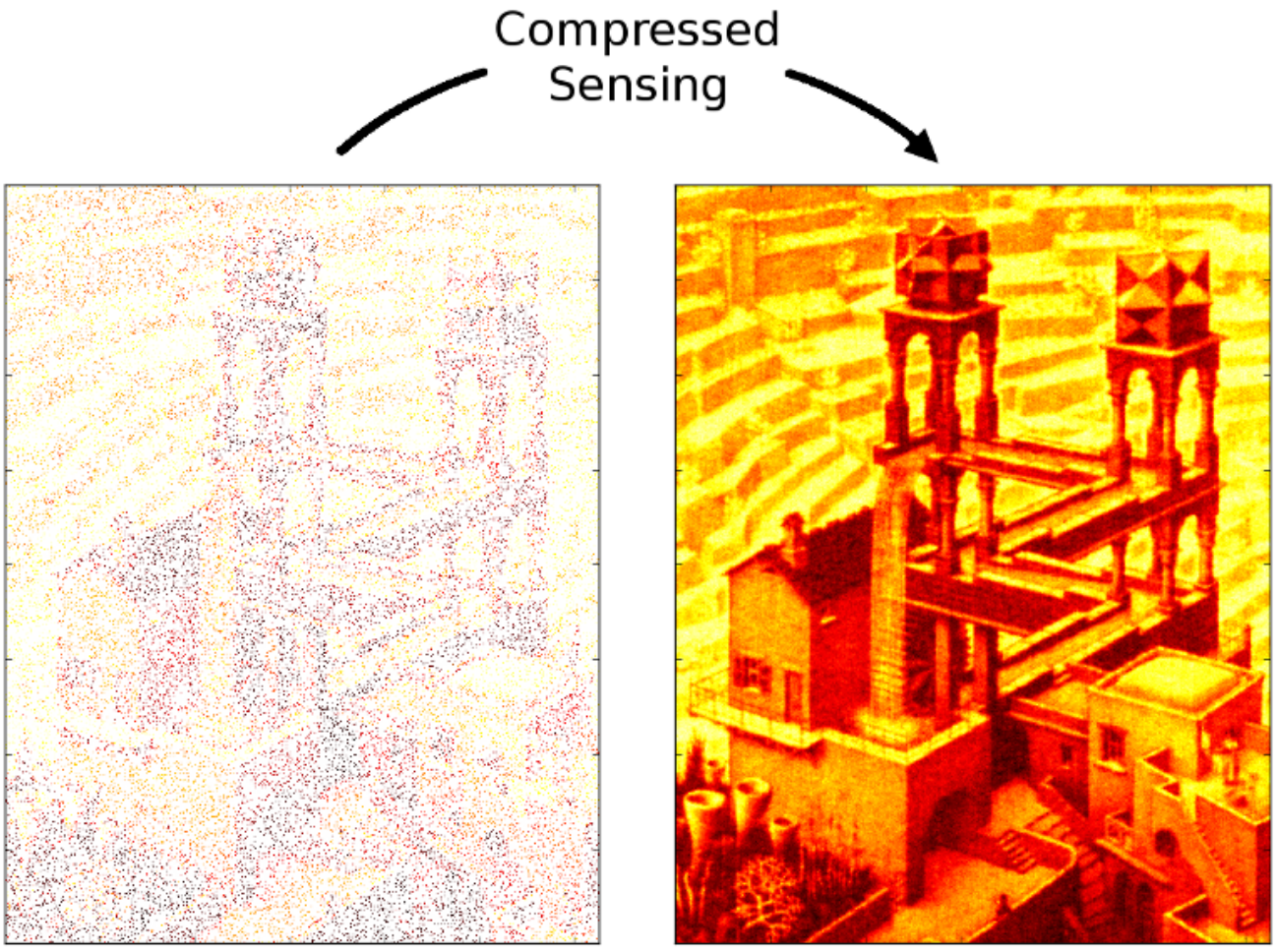
```
A * FFTW.dct(interp_linear.(t)) - y2
```

```
# norm(JuMP.value.(x), 1)
```

```
1252.8136456312393
```

```
norm(FFTW.dct(interp_linear.(t)), 1)
```

Example 2: Recovering an image



Creating a Julia Package

1. Go to the folder for package development,

```
cd path/to/julia/dev/folder
```

2. Type the following command

```
julia> using PkgTemplates

julia> tpl = Template(; user="GiggleLiu", plugins=[
           GitHubActions(; extra_versions=["nightly"]),
           Git(),
           Codecov()
    ], dir=pwd())

julia> tpl("CompressedSensingTutorial")
```

> **Note**
>
> Please replace `GiggleLiu` with your own user name, the `CompressedSensingTutorial` with your own package name! Please check the document of PkgTemplates. Now you should see a new package in your current folder.

3. develop your project with, e.g., VSCode.

```
cd CompressedSensingTutorial

code .
```

In the VSCode, please use your project environment as your julia project environment, check here.

4. Please configure your project dependency by typing in the `pkg>` mode.

```
(CompressedSensingTutorial) pkg> add FFTW FiniteDifferences Images Optim ...
```

https://github.com/timholy/Revise.jl

```
1 using Images
```

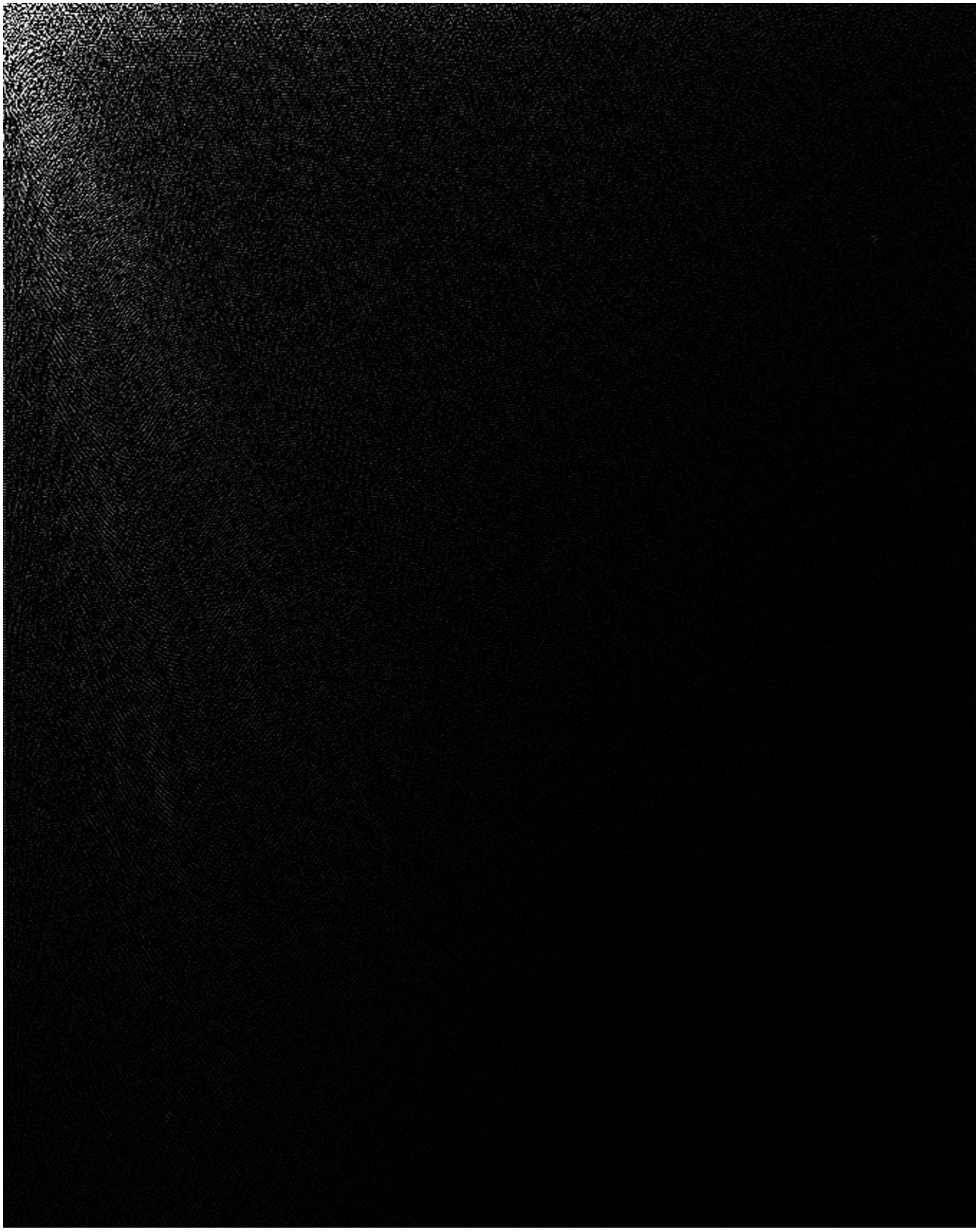
source_img =

```
source_img = Gray.(Images.load(joinpath(@__DIR__, "images/waterfall.jpeg")))
```

```
img = Float64.(source_img);
```

```
(852, 677)
```

```
1 size(img)
```



```
1 Gray.(FFTW.dct(img))
```

```
using NLSolversBase
```

```
mod = Main.var"CompressedSensingTutorial.jl"
```

```
# We have to use the Pluto ingredients for loading a local project
# Please check the issue:
https://github.com/fonsp/Pluto.jl/issues/115#issuecomment-661722426
mod =
ingredients("../lib/CompressedSensingTutorial/src/CompressedSensingTutorial.jl")
```

```
CT = Main.var"CompressedSensingTutorial.jl".CompressedSensingTutorial
```

```
CT = mod.CompressedSensingTutorial
```

Let us check the project!

```
pixels =
```

```
ImageSamples((852, 677), [CartesianIndex(237, 132), CartesianIndex(566, 168), Cartesia
```

```
pixels = CT.sample_image_pixels(img, 0.1)
```

The objective function.

```
Gray.(CT.zero_padded(pixels, pixels.values))
```

```
using Optim
```

```
using FiniteDifferences
```

☐

```
@bind do_compressed_sensing CheckBox()
```

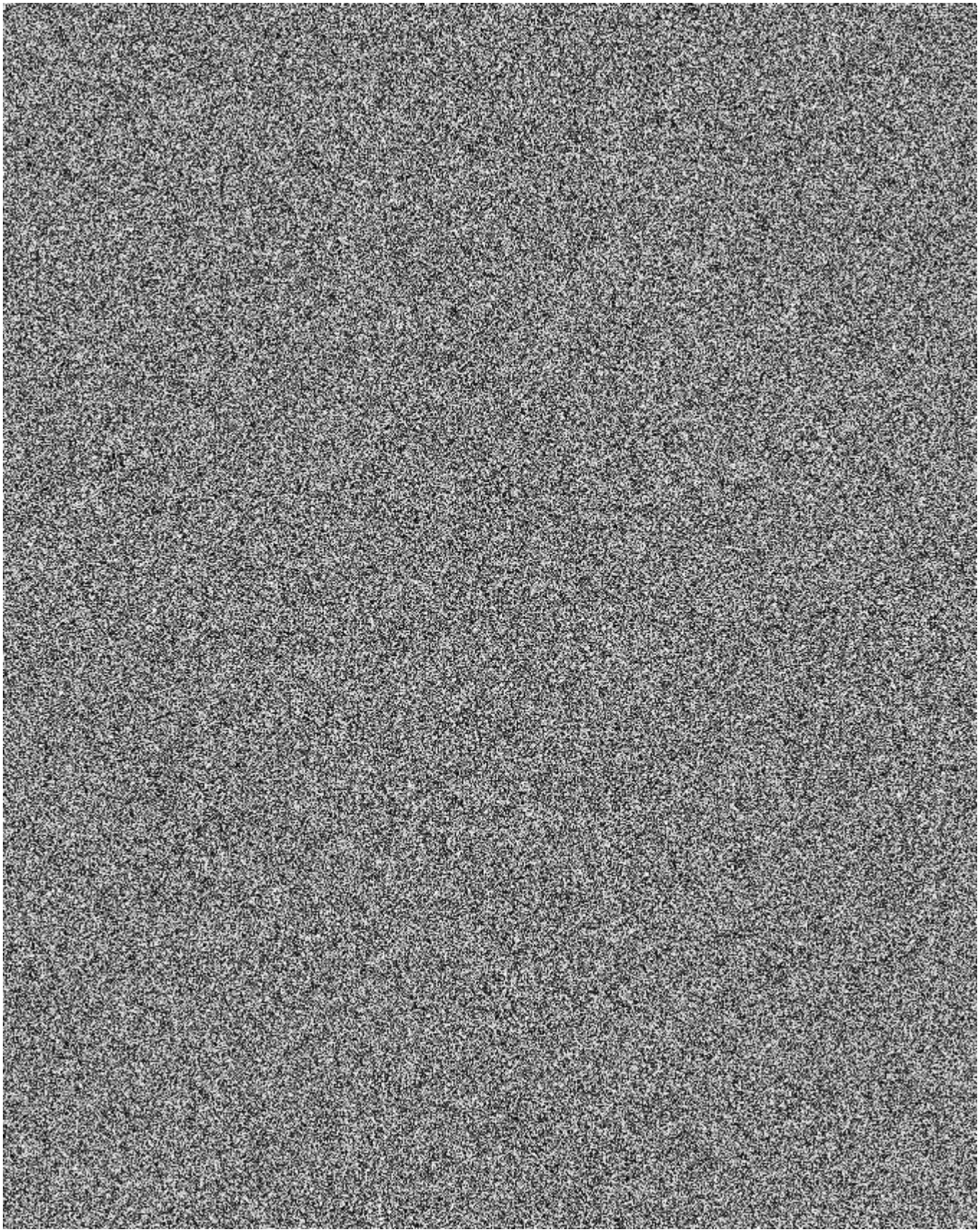
No documentation found.

`Main.var"CompressedSensingTutorial.jl".CompressedSensingTutorial.sensing_image` is a `Function`.

```
# 1 method for generic function "sensing_image":
[1] sensing_image(samples::Main.var"CompressedSensingTutorial.jl".CompressedSensingTutorial.ImageSamples; img0, C, iterations, g_tol, show_trace, optimizer, linesearch) in Main.var"CompressedSensingTutorial.jl".CompressedSensingTutorial at /home/user1/ModernScientificComputing/lib/CompressedSensingTutorial/src/compressed_sensing_2d.jl:82
```

```
@doc CT.sensing_image
```

```
newimg = if do_compressed_sensing
        CT.sensing_image(pixels; C=0.005, optimizer=:LBFGS, show_trace=false, linesearch=Optim.HagerZhang())
else
        rand(size(img)...)
end;
```



```
Gray.(newimg)
```

```
852×677 Matrix{Float64}:
 379.712      -0.151358     0.331452   …  -0.149366   -0.565087   -0.167036
  -0.295914   -0.13364     -0.0971303      -0.0136837   0.397504   -0.143805
   0.167813    0.151115     0.361371       -0.257618   -0.484249    0.224002
  -0.42675     0.387339    -0.628964       -0.25449     0.0391268  -0.0583232
   0.0731325   0.129793     0.642018        0.267562   -0.242472    0.447718
   0.0735419  -0.168379     0.0547628  …    0.357009    0.281323    0.130422
  -0.101056   -0.499675     0.480157        0.0811993  -0.026564   -0.248805
   ⋮                                   ⋱               ⋮
   0.0470563   0.196965     0.142223       -0.161866    0.267023    0.11059
   0.31407     0.322496    -0.0743662      0.285801    0.0569936  -0.0469573
  -0.0234353  -0.281623     0.0975859      -0.234342   -0.0931713   0.374223
  -0.35144     0.113908     0.0846986      0.225797    0.0520964   0.0302883
   0.310478    0.00363086   0.101769   …  -0.636723    0.264145   -0.328049
   0.178374    0.549348    -0.215101       -0.302296   -0.176181   -0.409846
```

```
1 FFTW.dct(newimg)
```

```
[-0.243377, 0.108125, -0.521312, -0.0422847, 0.0563089, -0.123375, 0.375204, -0.112139
```

```
1 newimg[pixels.indices] .- pixels.values
```

Related research works

- Quantum State Tomography via Compressed Sensing. David Gross, Yi-Kai Liu, Steven T. Flammia, Stephen Becker, and Jens Eisert. Phys. Rev. Lett. 105, 150401 – Published 4 October 2010

Kernel PCA

References:

- Universal Kernels, Charles A. Micchelli, Yuesheng Xu, Haizhang Zhang; 2006.
- Kernel Principal Component Analysis, Bernhard Scholkopf, Alexander Smola, Klaus Robert Muller, 1997
- Universality, Characteristic Kernels and RKHS Embedding of Measures Sriperumbudur, B. K., Fukumizu, K. & Lanckriet, G. R. G. Journal of Machine Learning Research 12, 2389–2410 (2011).

Kernel Method

From dot product to distance

Let $x, y \in R^n$ be two vectors, their distance is defined by

$$\text{dist}(x, y) = \|x - y\|^2 = x^T x + y^T y - 2y^T x$$

If we can defined an inner product between two vectors, we can defined a measure of distance.

Kernel functions

By extending the dot product by an arbitrary symmetric positive definite kernel funciton.

$$x^T y \to \kappa(x, y)$$

We have a new measure of distance as

$$\text{dist}_\kappa(x, y) = \kappa(x, x) + \kappa(y, y) - 2\kappa(x, y)$$

Universality of a kernel

A kernel κ is universal if and only if the following equation is a universal function approximator.

$$f = \sum_{j=1}^{n} c_j \kappa(\cdot, x_j),$$

where $c_j \in \mathbb{R}$ and x_j can be either a number of a vector.

As noted in Micchelli et al. (2006), one can ask whether the function, f in the above euqation approximates any real-valued target function arbitrarily well as the number of summands increases without bound. This is an important question to consider because if the answer is affirmative, then the kernel-based learning algorithm can be consistent in the sense that for any target function, $f^\star$, the discrepancy between f (which is learned from the training data) and $f^\star$ goes to zero (in some appropriate sense) as the sample size goes to infinity.

Homework

1. Autodiff

Given $A \in \mathbb{R}^{n \times n}$ and $x, b \in \mathbb{R}^n$. Please derive the backward rule of $\mathcal{L} = \|Ax - b\|_2$ either using the chain rules or the perturbative approach (from the last lecture).

2. Sparsity detection

Choose one.

(a). Text compression

Given a text to be compressed:

```
Compressed sensing (also known as compressive sensing, compressive sampling, or s
parse sampling) is a signal processing technique for efficiently acquiring and re
constructing a signal, by finding solutions to underdetermined linear systems. Th
is is based on the principle that, through optimization, the sparsity of a signal
can be exploited to recover it from far fewer samples than required by the Nyquis
t–Shannon sampling theorem. There are two conditions under which recovery is poss
ible. The first one is sparsity, which requires the signal to be sparse in some d
omain. The second one is incoherence, which is applied through the isometric prop
erty, which is sufficient for sparse signals.
```

Please

1. Analyse the frequency of each char
2. Create an optimal Huffman coding for each char
3. Encode the text and count the length of total coding (not including the deliminators).

(b). Compressed Sensing

Go through the video clip Compressed Sensing: When It Works

Please summarize this video clip, and explain when does compressed sensing work and when not.

Note

If you are interested in knowing more about compressed sensing, please do not miss this youtube video paylist: https://youtube.com/playlist?list=PLMrJAkhIeNNRHP5UA-gIimsXLQyHXxRty