AI Expert 프로그램 실습

7/9 Graphical Models, Gaussian Process, Hawkes Process

Overview

This tutorial is three-fold as follows:

1.	Graphical Models	- 80 min.
2.	Gaussian Process (GP)	- 80 min.
3.	Hawkes Process (HP)	- 80 min.

* 10 minutes break between each part.

Environments

1. Python 3.6
2. virtualenv
3. Gpy
4. Matplotlib
5. Scipy
6. Image
7. Tqdm
8. Ipykernel
9. Tick

   Module for statistical learning, with a particular emphasis on time-dependent modelling

Part 0. Environment Setting

Download the source code

```
cogito@digits-1:~$ git clone https://github.com/cogito288/samsung-ds-kaist.git
```

Part 0. Environment Setting

Create the virtual environment and activate it

```
cogito@digits-1:~$ virtualenv -p python3 samdung-ds-0710-env
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in /home/cogito/samdung-ds-0710-env/bin/python3
Also creating executable in /home/cogito/samdung-ds-0710-env/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
cogito@digits-1:~$ source samdung-ds-0710-env/bin/activate
(samdung-ds-0710-env) cogito@digits-1:~$
```

Install ipykernel package

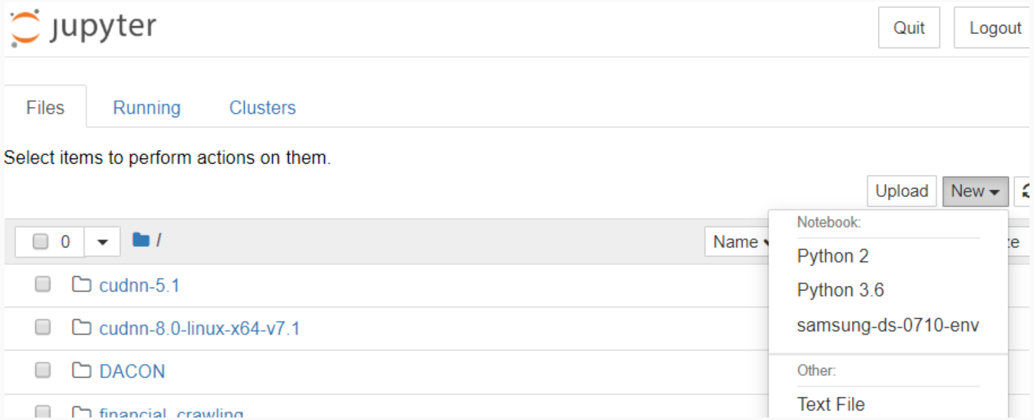
```
(samdung-ds-0710-env) cogito@digits-1:~$ pip install ipykernel
(samdung-ds-0710-env) cogito@digits-1:~$ pip install matplotlib
(samdung-ds-0710-env) cogito@digits-1:~$ pip install scipy==1.0.0
(samdung-ds-0710-env) cogito@digits-1:~$ pip install image
(samdung-ds-0710-env) cogito@digits-1:~$ pip install tqdm
```

Part 0. Environment Setting

Add virtualenv to Jupyter kernel

```
cogito@digits-1:~$ python3 -m ipykernel install --user  \
--name samsung-ds-0710-env --display-name "samsung-ds-0710-env“
Installed kernelspec samsung-ds-0710-env in /home/cogito/.local/share/jupyter/kernels/samsung-ds-0710-env
```

```
cogito@digits-1:~$ jupyter notebook
```

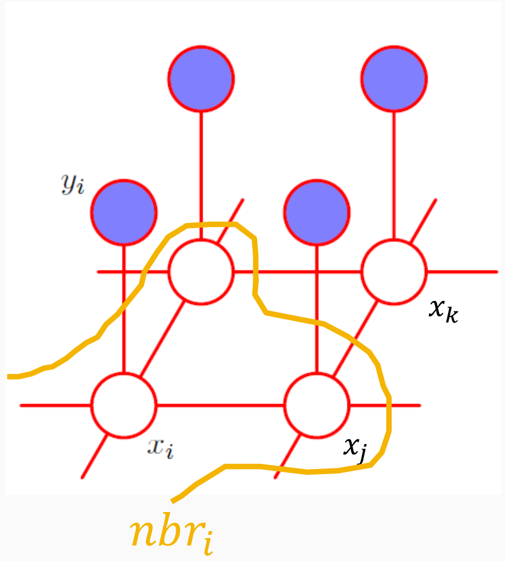


Select items to perform actions on them.

cudnn-5.1
cudnn-8.0-linux-x64-v7.1
DACON
financial_crawling

Part 1. Graphical Models (recap)

- Markov Random Field

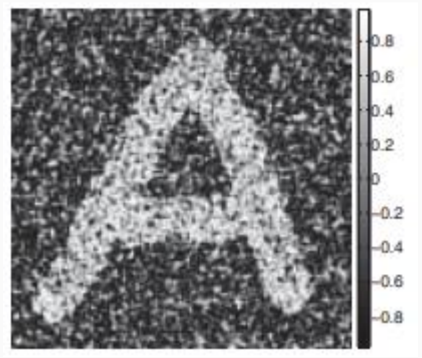


- Markov Property
  - $p(x_i, x_j) \neq p(x_i)p(x_j)$ for $x_j \in nbr_i$
  - $p(x_i, x_k) = p(x_i)p(x_k)$ for $x_k \notin nbr_i$

- Energy Based Model
  - $p(x) = \frac{1}{Z}\exp(-E(x))$

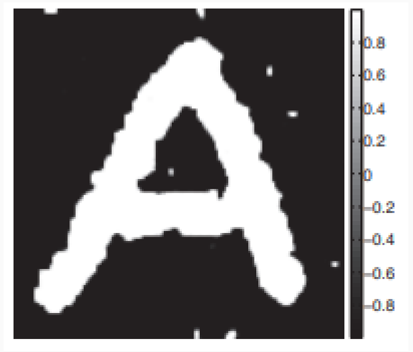
Part 1. Graphical Models (recap)

- Image Denoising using Ising Model



Noisy Image

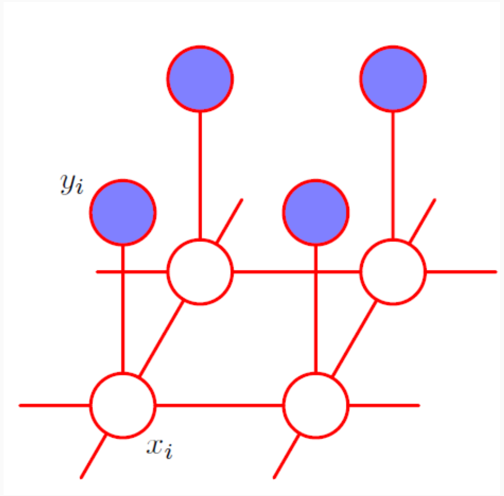
Denoising



Clean Image

Part 1. Graphical Models (recap)

- Ising Model



- y_i: noisy pixel value for i^{th} pixel
- x_i: binary state value for i^{th} pixel $\in \{-1, 1\}$
- $nbr_i = \{x_{i\leftarrow}, x_{i\rightarrow}, x_{i\uparrow}, x_{i\downarrow}\}$

- $p(x) = \frac{1}{Z_0}\exp\left(-E_0(x)\right)$
- $E_0(x) = -\sum_{i=1}^{D}\sum_{j\in nbr_i} W_{ij}x_ix_j$

Part 1. Graphical Models (recap)

- Ising Model
  - $p(x) = \frac{1}{Z_0} \exp\left(-E_0(x)\right)$
  - $E_0(x) = -\sum_{i=1}^{D} \sum_{j \in nbr_i} W_{ij} x_i x_j$

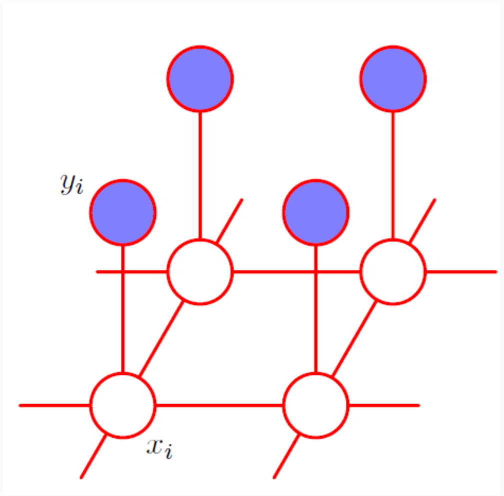
$x_j \backslash x_i$	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

- $W_{ij} > 0$
  - When $x_i = x_j$, $p(x)$ is high.
  - When $x_i \neq x_j$, $p(x)$ is small
- $W_{ij} < 0$
  - When $x_i \neq x_j$, $p(x)$ is high.
  - When $x_i = x_j$, $p(x)$ is small

Part 1. Graphical Models (recap)

- Ising Model



- y_i: noisy pixel value for i^{th} pixel
- x_i: binary state value for i^{th} pixel $\in \{-1, 1\}$
- $nbr_i = \{x_{i\leftarrow}, x_{i\rightarrow}, x_{i\uparrow}, x_{i\downarrow}\}$

- $p(x) = \frac{1}{Z_0} \exp\left(-E_0(x)\right)$
- $E_0(x) = -\sum_{i=1}^{D} \sum_{j \in nbr_i} W_{ij} x_i x_j \quad (\text{set } W_{ij} = 1)$
  $= -\sum_{i=1}^{D} \sum_{j \in nbr_i} x_i x_j$
- $p(y|x) = \prod_i p(y_i|x_i) \qquad (\text{Markov property})$
  $= \prod_i N(y_i|\, x_i)$

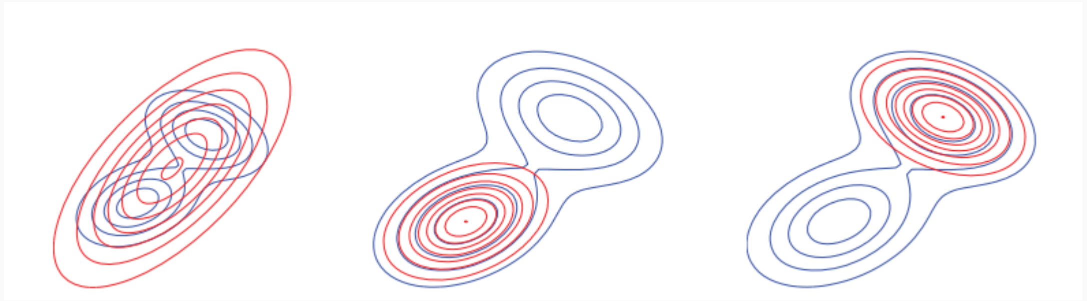
Part 1. Graphical Models (recap)

- $p(x) = \frac{1}{Z_0} \exp\left(-E_0(x)\right)$
- $E_0(x) = -\sum_{i=1}^{D} \sum_{j \in nbr_i} x_i x_j$
- $p(y|x) = \prod_i N(y_i|\, x_i)$

- $p(x|y) = \frac{1}{Z} p(y|x)p(x) = \frac{1}{Z} \exp(-\mathrm{E}_0(x) + \log \prod_i N(y_i|\, x_i))$
  $= \frac{1}{Z} \exp\left(\sum_{i=1}^{D} \sum_{j \in nbr_i} x_i x_j + \log \prod_i N(y_i|\, x_i)\right)$

Part 1. Graphical Models (recap)

- Variational Inference
  - Approximate $\underbrace{\text{intractable distribution } p(x)}_{\text{Target distribution}}$ using $\underbrace{\text{tractable distribution } q(x)}_{\text{Proposal distribution}}$.
- Mean field approximation
  - $q(x) = \prod_i q_i(x_i)$
- Example:
  - $p(x)$: Unknown distribution
  - $q(x)$: Normal distribution



Part 1. Graphical Models (recap)

- Variational inference for Ising model
  - Target distribution: $p(x|y)$
  - Proposal distribution: $q(x)$
- Mean field approximation
  - $q(x) = \prod_i q(x_i, \mu_i)$ where μ_i is mean value for x_i
- $q_i(x_i) = \frac{1}{Z_i} \exp\left(\mathrm{E}_{-q_i}[\log p(x|y)]\right)$
  - $\log\left(p(x|y)\right) = \sum_{i=1}^{D} \sum_{j \in nbr_i} x_i x_j + \log \prod_i N(y_i|\, x_i) + const$
  - $\mathrm{E}_{-q_i}[\log p(x|y)] = \mathrm{E}_{-q_i}\left[\sum_{i=1}^{D} \sum_{j \in nbr_i} x_i x_j + \log \prod_i N(y_i|\, x_i) + const\right]$

$$= x_i \sum_{j \in nbr_i} \mathrm{E}_{\mathrm{q_j}}\left[x_j\right] + \log N(y_i|x_i) + const = x_i \sum_{j \in nbr_i} \mu_j + \log N(y_i|x_i) + const$$

Part 1. Graphical Models (recap)

- $q_i(x_i) = \frac{1}{Z_i} \exp\left(\mathrm{E}_{-q_i}[\log p(x|y)]\right)$
  - $\mathrm{E}_{-q_i}[\log p(x|y)] = x_i \sum_{j \in nbr_i} \mu_j + \log N(y_i|x_i) + const$
- $q_i(x_i) \propto \exp\left(x_i \sum_{j \in nbr_i} \mu_j + \log N(y_i|x_i)\right)$
- $q_I(x_i = 1) = \frac{\exp\left(\sum_{j \in nbr_i} \mu_j + \log N(y_i|1)\right)}{\exp\left(\sum_{j \in nbr_i} \mu_j + \log N(y_i|1)\right) + \exp\left(-\sum_{j \in nbr_i} \mu_j + \log N(y_i|-1)\right)}$

$$= \frac{1}{1 + \exp\left(-2\sum_{j \in nbr_i} \mu_j + \log N(y_i|-1) - \log N(y_i|1)\right)} = \mathrm{sigmoid}(2\mathrm{a_i})$$

$$\mathrm{a_i} = \sum_{j \in nbr_i} \mu_j + 0.5 * (\log N(y_i|1) \; - \log N(y_i|-1))$$

Part 1. Graphical Models (recap)

- $q_i(x_i = 1) = \text{sigmoid}(2\mathrm{a_i})$
  - $\mathrm{a_i} = \sum_{j \in nbr_i} \mu_j + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))$
- $q_i(x_i = -1) = \text{sigmoid}(-2\mathrm{a_i})$
- $\mu_i = \mathrm{E}_{q_i}[x_i] = (+1) \cdot q_i(x_i = 1) + (-1) \cdot q_i(x_i = -1)$

$$= \frac{1}{1+\exp(-2a_i)} - \frac{1}{1+\exp(2a_i)} = \frac{\exp(a_i)}{\exp(a_i)+\exp(-a_i)} - \frac{\exp(-a_i)}{\exp(-a_i)+\exp(a_i)}$$

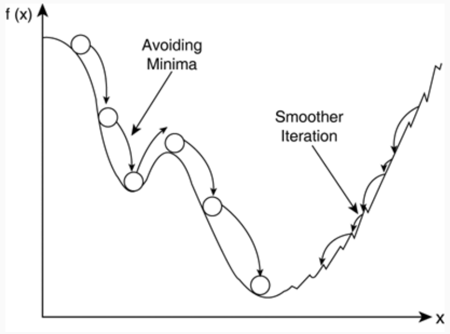
$$= \frac{\exp(a_i)-\exp(-a_i)}{\exp(a_i)+\exp(-a_i)} = \tanh a_i$$

Part 1. Graphical Models (recap)

- Update variational parameter μ_i
  - $\mu_i = \tanh(a_i) \; = \tanh\left(\sum_{j \in nbr_i} \mu_j + 0.5 * (\log N(y_i|1) \; - \log N(y_i|-1))\right)$
  - $\mu_i^t = \tanh\left(\sum_{j \in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) \; - \log N(y_i|-1))\right)$

(fixed point algorithm)

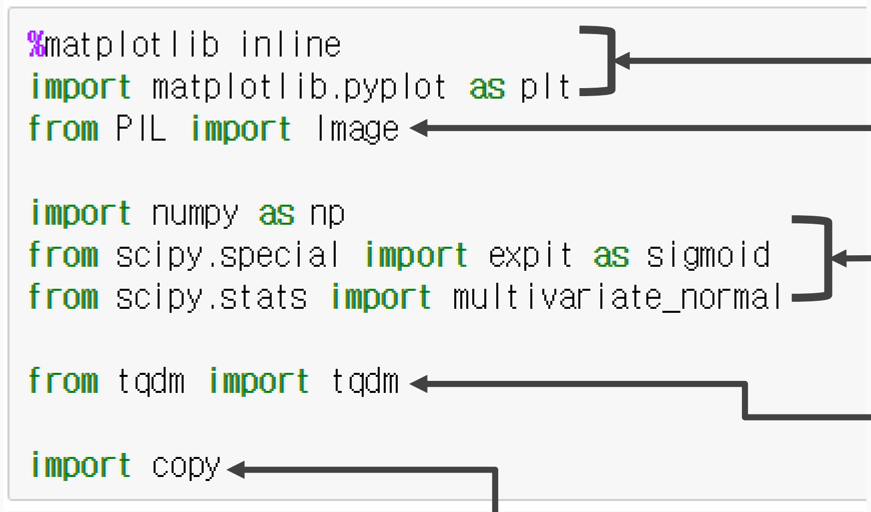
  - $\mu_i^t = (1-\lambda)\mu_j^{t-1} + \lambda \tanh\left(\sum_{j \in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) \; - \log N(y_i|-1))\right)$



(damped update)

Part 1. Graphical Models (code)

- Import library



```
%matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image

import numpy as np
from scipy.special import expit as sigmoid
from scipy.stats import multivariate_normal

from tqdm import tqdm

import copy
```

To visualize data (e.g. plot)

To handle image data (e.g. open)

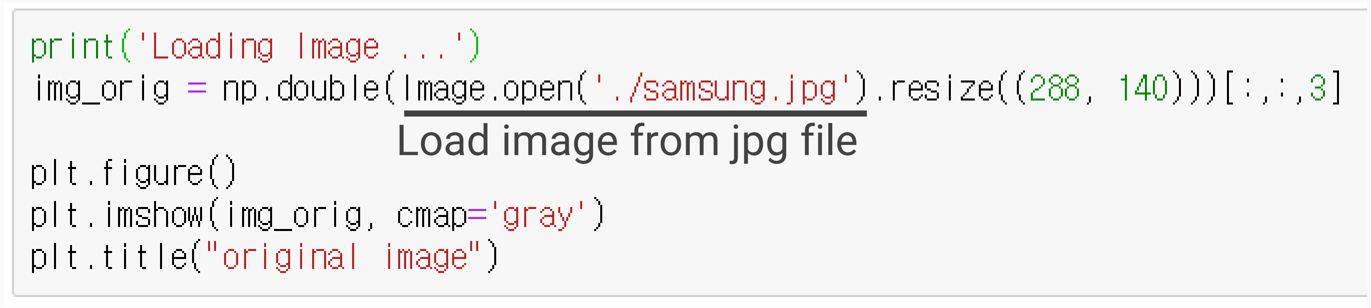
Mathematical library/function/class

To visualize progress (e.g. iteration of loop)

For deep copy of array

Part 1. Graphical Models (code)

- Load image file as array

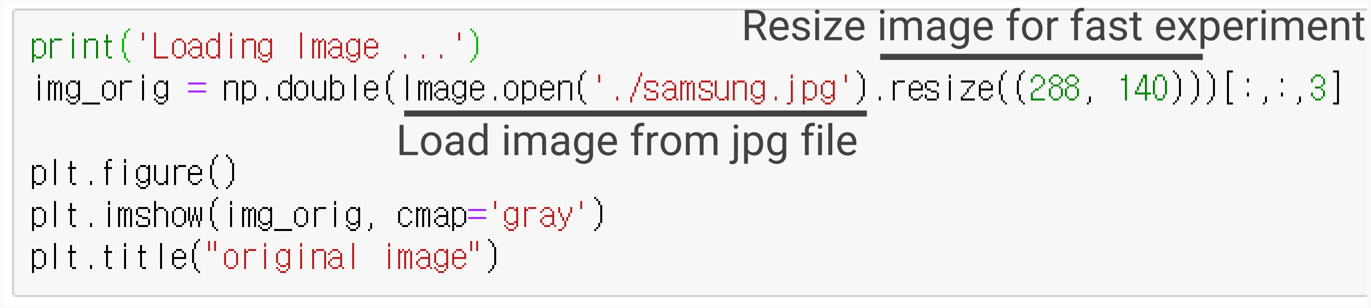


```
print('Loading Image ...')
img_orig = np.double(Image.open('./samsung.jpg').resize((288, 140)))[:,:,3]

plt.figure()
plt.imshow(img_orig, cmap='gray')
plt.title("original image")
```

Load image from jpg file

Part 1. Graphical Models (code)

- Load image file as array



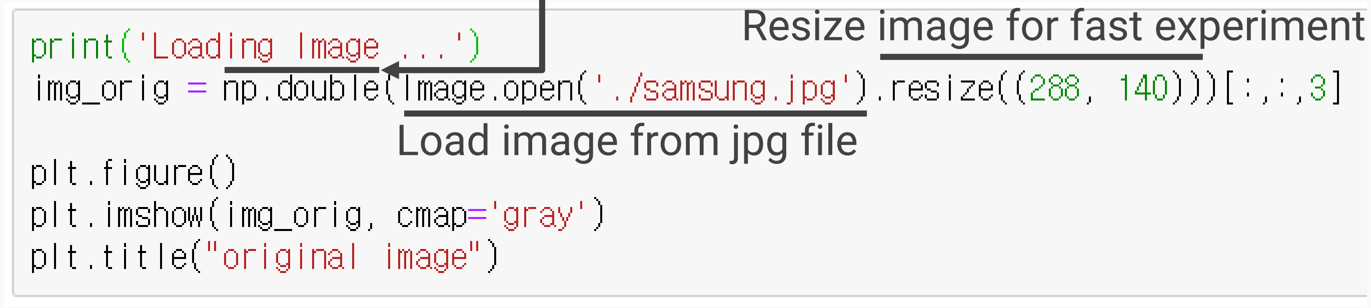
```
print('Loading Image ...')
img_orig = np.double(Image.open('./samsung.jpg').resize((288, 140)))[:,:,3]

plt.figure()
plt.imshow(img_orig, cmap='gray')
plt.title("original image")
```

Part 1. Graphical Models (code)

- Load image file as array

Type cast from image to double array

Resize image for fast experiment



```
print('Loading Image ...')
img_orig = np.double(Image.open('./samsung.jpg').resize((288, 140)))[:,:,3]

plt.figure()
plt.imshow(img_orig, cmap='gray')
plt.title("original image")
```

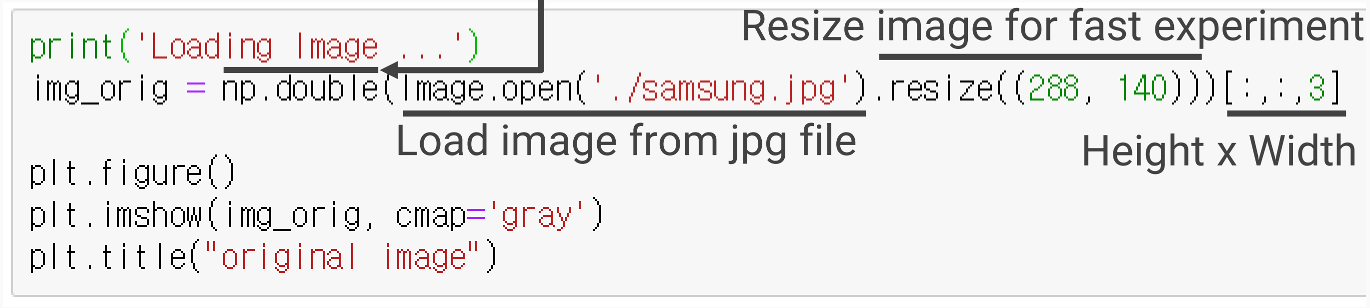
Load image from jpg file

Part 1. Graphical Models (code)

- Load image file as array

Type cast from image to double array

Resize image for fast experiment



```
print('Loading Image ...')
img_orig = np.double(Image.open('./samsung.jpg').resize((288, 140)))[:,:,3]

plt.figure()
plt.imshow(img_orig, cmap='gray')
plt.title("original image")
```

Load image from jpg file

Height x Width X RGBA

Part 1. Graphical Models (code)

- Load image file as array

Type cast from image to double array

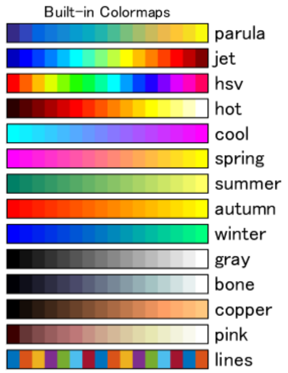
Resize image for fast experiment

```
print('Loading Image ...')
img_orig = np.double(Image.open('./samsung.jpg').resize((288, 140)))[:,:,3]

plt.figure()
plt.imshow(img_orig, cmap='gray')
plt.title("original image")
```

Load image from jpg file

Height x Width X RGBA

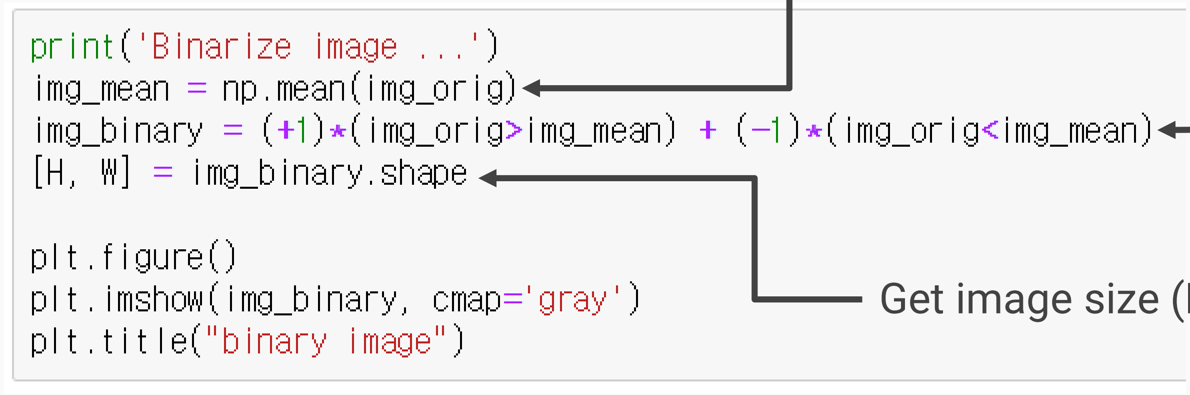
1. Open new window
2. Plot image gray color map
3. Set title to plot



Part 1. Graphical Models (code)

- Binarize pixel value of image

Compute mean pixel value of image



```
print('Binarize image ...')
img_mean = np.mean(img_orig)
img_binary = (+1)*(img_orig>img_mean) + (-1)*(img_orig<img_mean)
[H, W] = img_binary.shape

plt.figure()
plt.imshow(img_binary, cmap='gray')
plt.title("binary image")
```

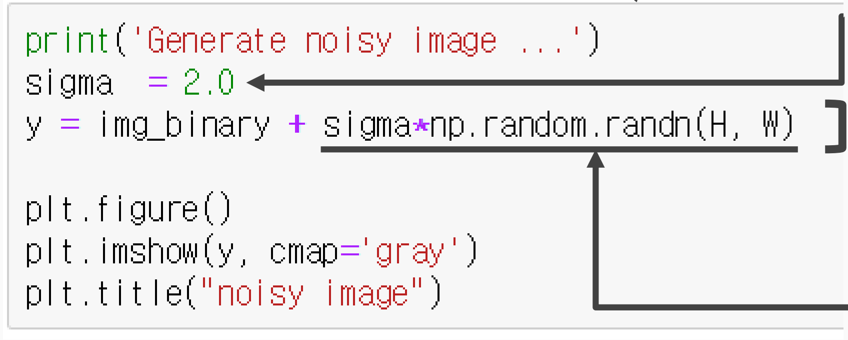
Binarize image based on mean pixel value

Get image size (Height and Width)

Part 1. Graphical Models (code)

- Generate noisy image

Set the noise level
(standard deviation=2.0)



```
print('Generate noisy image ...')
sigma  = 2.0
y = img_binary + sigma*np.random.randn(H, W)

plt.figure()
plt.imshow(y, cmap='gray')
plt.title("noisy image")
```

Add noise ($N(0, \sigma^2)$)
to binarized image

$$N(0, \sigma^2) = \sigma N(0, 1)$$

Part 1. Graphical Models (code)

- Build Ising model

$x_j \backslash x_i$	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

```
W_ij = 1.0
dampling_factor = 0.5
max_iter = 15

normal_pos = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1, cov=sigma**2), (H, W))
normal_neg = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1, cov=sigma**2), (H, W))
normal_all = normal_pos - normal_neg

mu = y

a = mu + 0.5*normal_all
prob_x_pos = sigmoid(+2*a)
prob_x_neg = sigmoid(-2*a)
```

- $\mu_i^t = \tanh\left(\sum_{j\in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))\right)$

(fixed point algorithm)

- $\mu_i^t = (1-\underline{\lambda})\mu_j^{t-1} + \underline{\lambda}\tanh\left(\sum_{j\in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))\right)$

(damped update)

Part 1. Graphical Models (code)

- Build Ising model

x_j \ x_i	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

```
W_ij = 1.0
dampling_factor = 0.5
max_iter = 15

normal_pos = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1, cov=sigma**2), (H, W))
normal_neg = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1, cov=sigma**2), (H, W))
normal_all = normal_pos - normal_neg

mu = y

a = mu + 0.5*normal_all
prob_x_pos = sigmoid(+2*a)
prob_x_neg = sigmoid(-2*a)
```

- $\overline{\mu_i^t} = \tanh\left(\sum_{j \in nbr_i} \overline{\mu_j^{t-1}} + 0.5 * (\log N(y_i|1) \; - \log N(y_i|-1))\right)$

(fixed point algorithm)

- $\overline{\mu_i^t} = (1-\lambda)\overline{\mu_j^{t-1}} + \lambda \tanh\left(\sum_{j \in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) \; - \log N(y_i|-1))\right)$

(damped update)

Part 1. Graphical Models (code)

- Build Ising model

x_j \ x_i	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

```
W_ij = 1.0
dampling_factor = 0.5
max_iter = 15

normal_pos = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1, cov=sigma**2), (H, W))
normal_neg = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1, cov=sigma**2), (H, W))
normal_all = normal_pos - normal_neg

mu = y

a = mu + 0.5*normal_all
prob_x_pos = sigmoid(+2*a)
prob_x_neg = sigmoid(-2*a)
```

1. $\log N(y_i|1)$
2. $\log N(y_i|-1)$
3. $\log N(y_i|1) - \log N(y_i|-1)$

Part 1. Graphical Models (code)

- Build Ising model

$x_j \backslash x_i$	-1	1
-1	$-W_{ij}$	W_{ij}
1	W_{ij}	$-W_{ij}$

$E_0(x)$

```
W_ij = 1.0
dampling_factor = 0.5
max_iter = 15

normal_pos = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1, cov=sigma**2), (H, W))
normal_neg = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1, cov=sigma**2), (H, W))
normal_all = normal_pos - normal_neg

mu = y

a = mu + 0.5*normal_all
prob_x_pos = sigmoid(+2*a)
prob_x_neg = sigmoid(-2*a)
```

Initial μ = y

1. $\log N(y_i|1)$
2. $\log N(y_i|-1)$
3. $\log N(y_i|1) - \log N(y_i|-1)$

Part 1. Graphical Models (code)

- Build Ising model

$$\begin{array}{c|cc} x_j \backslash x_i & -1 & 1 \\ \hline -1 & -W_{ij} & W_{ij} \\ 1 & W_{ij} & -W_{ij} \end{array}$$

$E_0(x)$

```
W_ij = 1.0
dampling_factor = 0.5
max_iter = 15

normal_pos = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1, cov=sigma**2), (H, W))
normal_neg = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1, cov=sigma**2), (H, W))
normal_all = normal_pos - normal_neg

mu = y

a = mu + 0.5*normal_all
prob_x_pos = sigmoid(+2*a)
prob_x_neg = sigmoid(-2*a)
```

Initial $\mu = y$

1. $\log N(y_i|1)$
2. $\log N(y_i|-1)$
3. $\log N(y_i|1) - \log N(y_i|-1)$

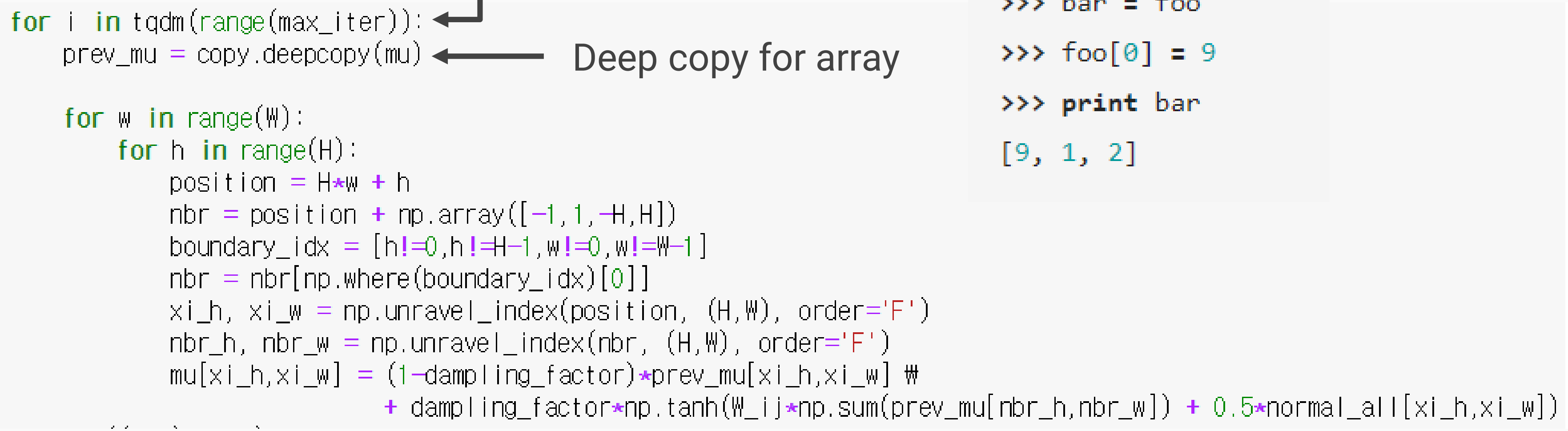
- $q_i(x_i = 1) = \mathrm{sigmoid}(2a_i)$
  - $a_i = \sum_{j \in nbr_i} \mu_j + 0.5 * (\log N(y_i|1) - \log N(y_i|-1))$
- $q_i(x_i = -1) = \mathrm{sigmoid}(-2a_i)$

Part 1. Graphical Models (code)

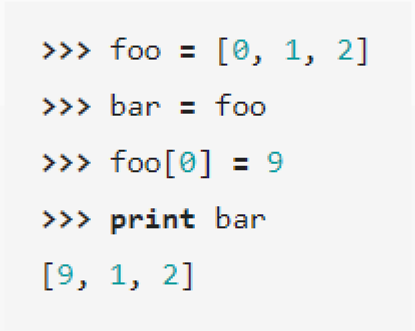
- Mean field variational inference

Visualize loop progress
(e.g. time per loop)

Deep copy for array



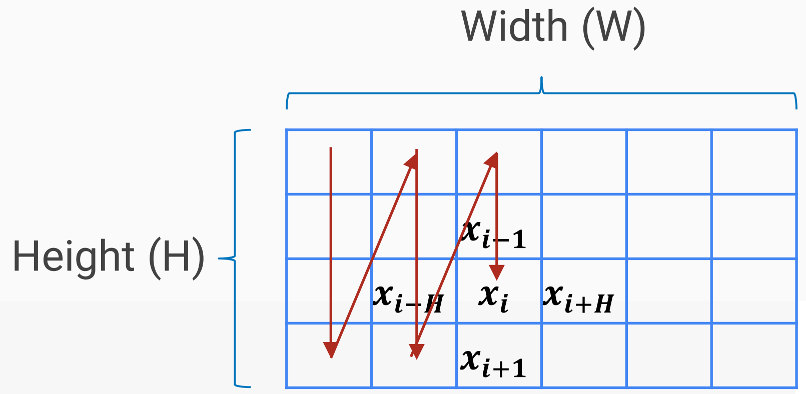
```
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] ₩
                            + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

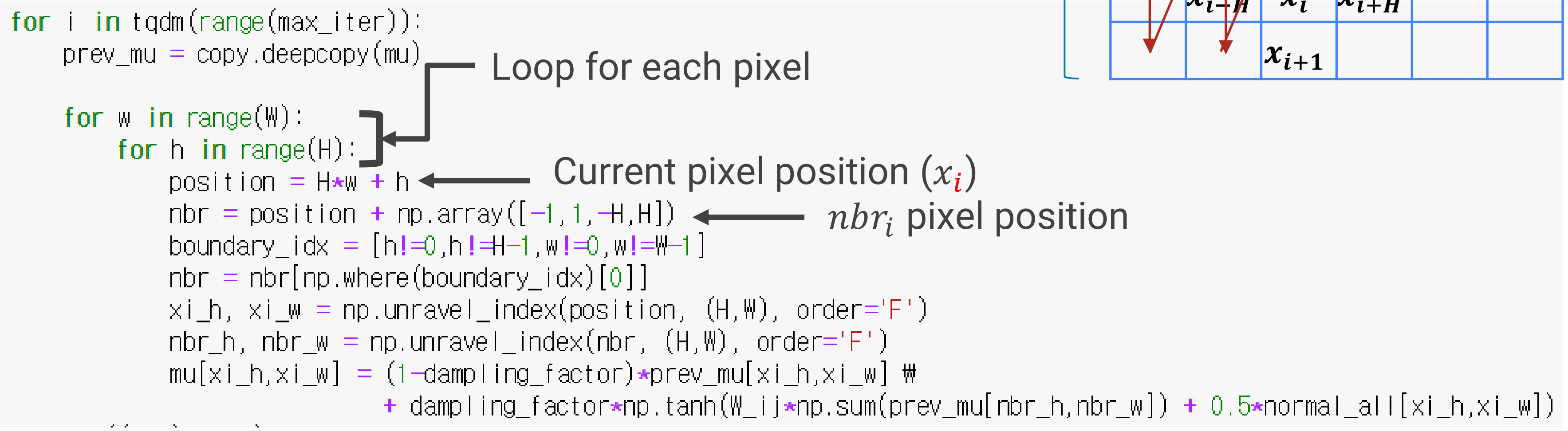


```
>>> foo = [0, 1, 2]
>>> bar = foo
>>> foo[0] = 9
>>> print bar
[9, 1, 2]
```

Part 1. Graphical Models (code)

- Mean field variational inference





```
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] ₩
                        + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

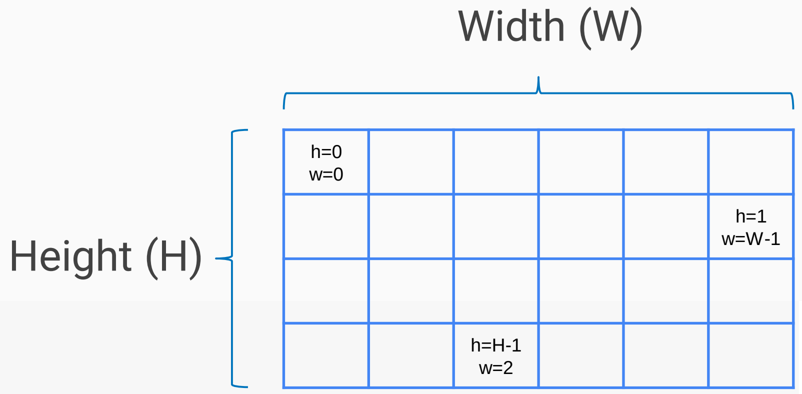
Loop for each pixel

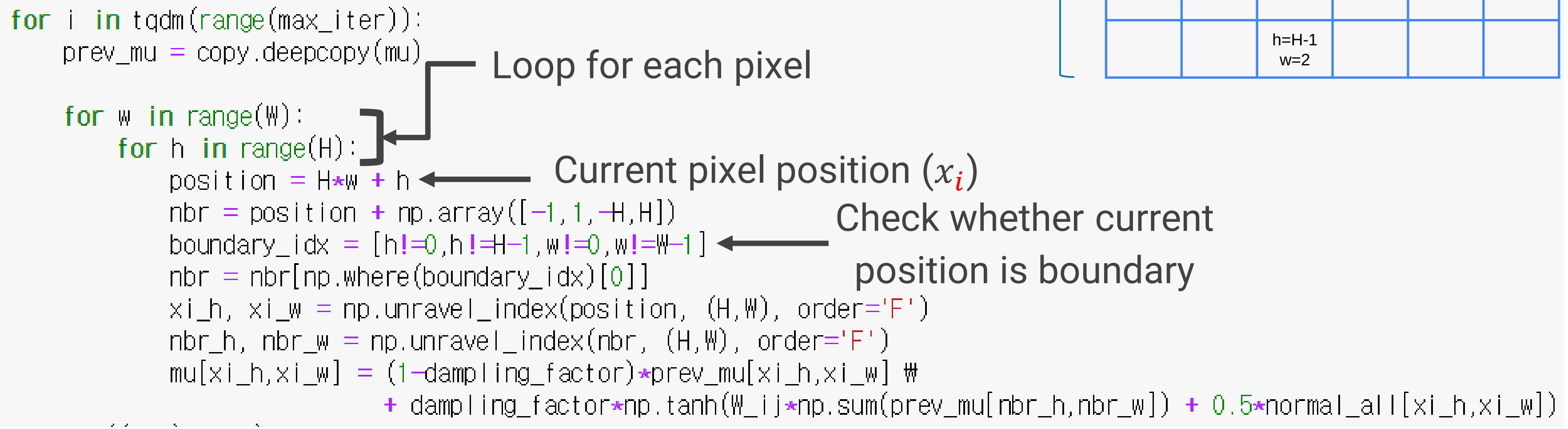
Current pixel position (x_i)

nbr_i pixel position

Part 1. Graphical Models (code)

- Mean field variational inference





```
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] ₩
                          + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

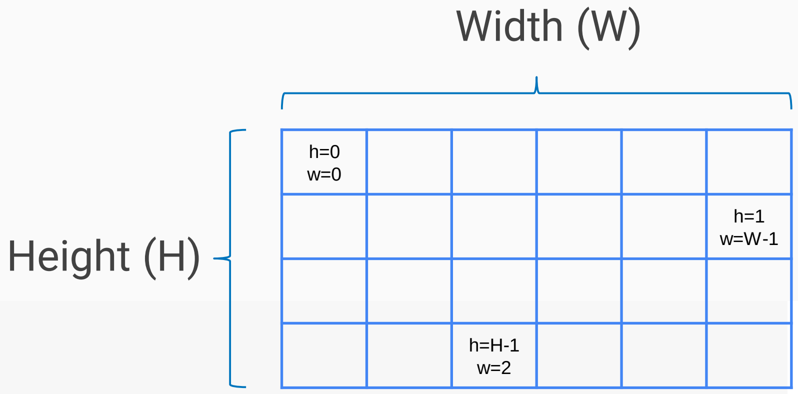
Loop for each pixel

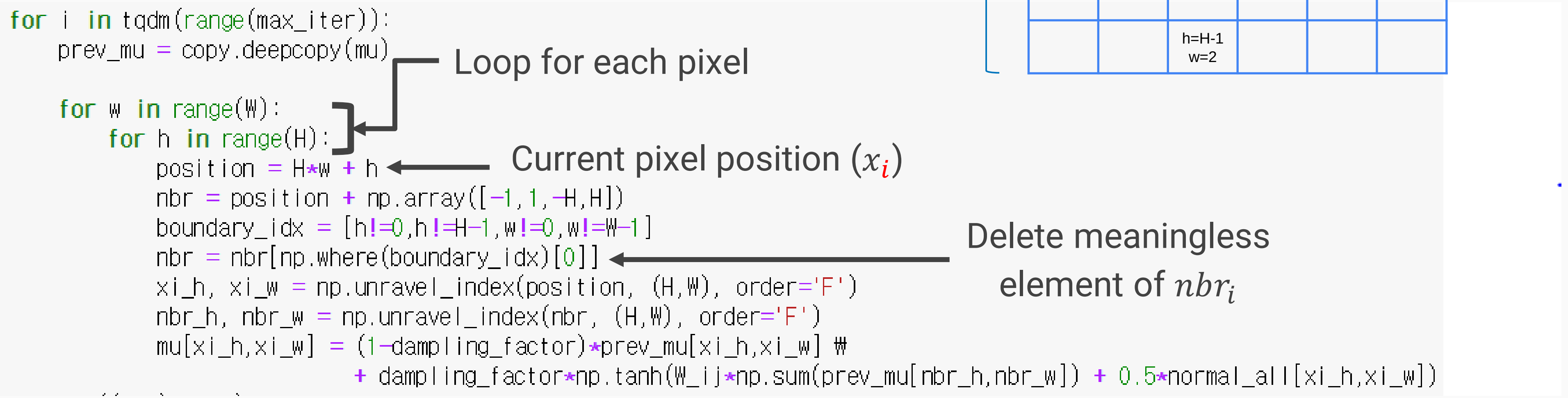
Current pixel position (x_i)

Check whether current position is boundary

Part 1. Graphical Models (code)

- Mean field variational inference





```
for i  in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] ₩
                            + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

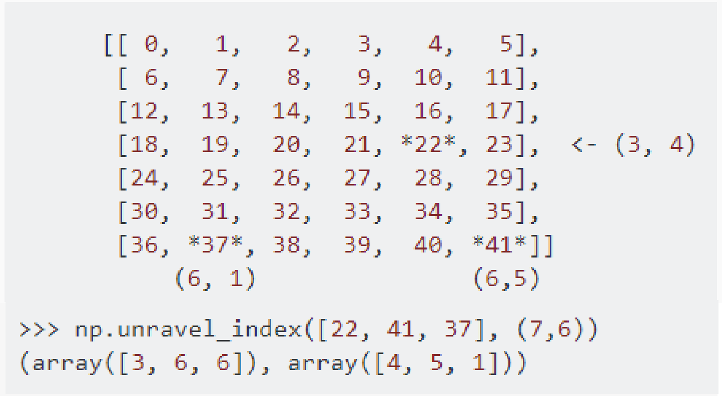
Loop for each pixel

Current pixel position (x_i)

Delete meaningless element of nbr_i

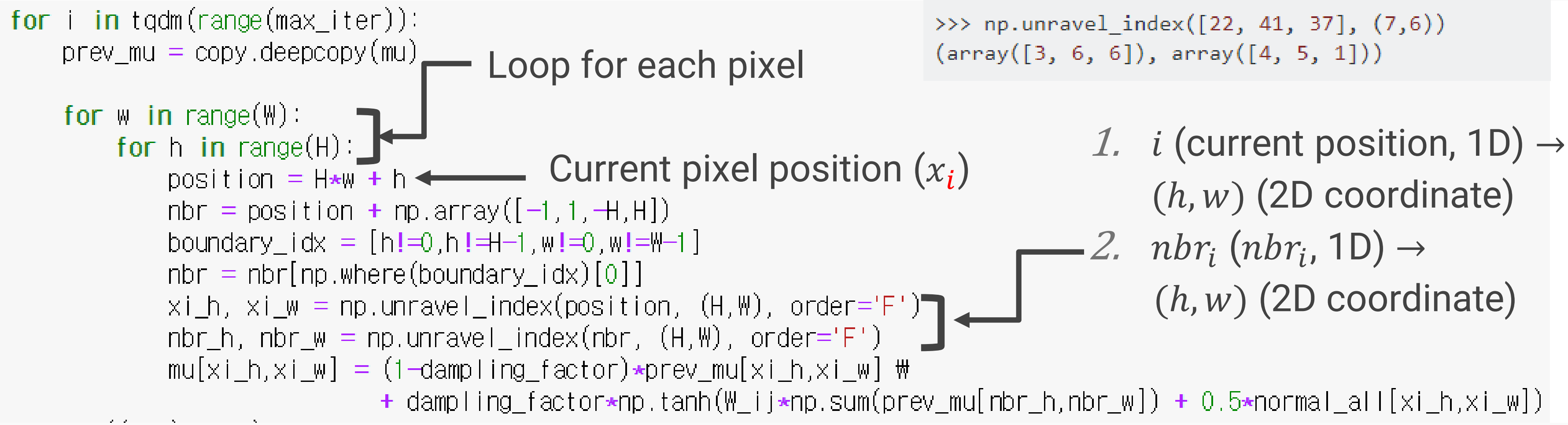
Part 1. Graphical Models (code)

- Mean field variational inference



```
[[ 0,    1,    2,    3,    4,    5],
 [ 6,    7,    8,    9,   10,   11],
 [12,   13,   14,   15,   16,   17],
 [18,   19,   20,   21, *22*,   23],   <- (3, 4)
 [24,   25,   26,   27,   28,   29],
 [30,   31,   32,   33,   34,   35],
 [36, *37*,   38,   39,   40, *41*]]
     (6, 1)                   (6,5)
```

```
>>> np.unravel_index([22, 41, 37], (7,6))
(array([3, 6, 6]), array([4, 5, 1]))
```



```
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] ₩
                          + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

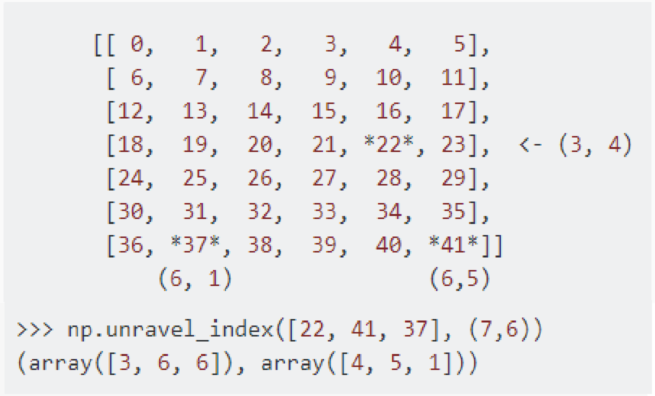
Loop for each pixel

Current pixel position (x_i)

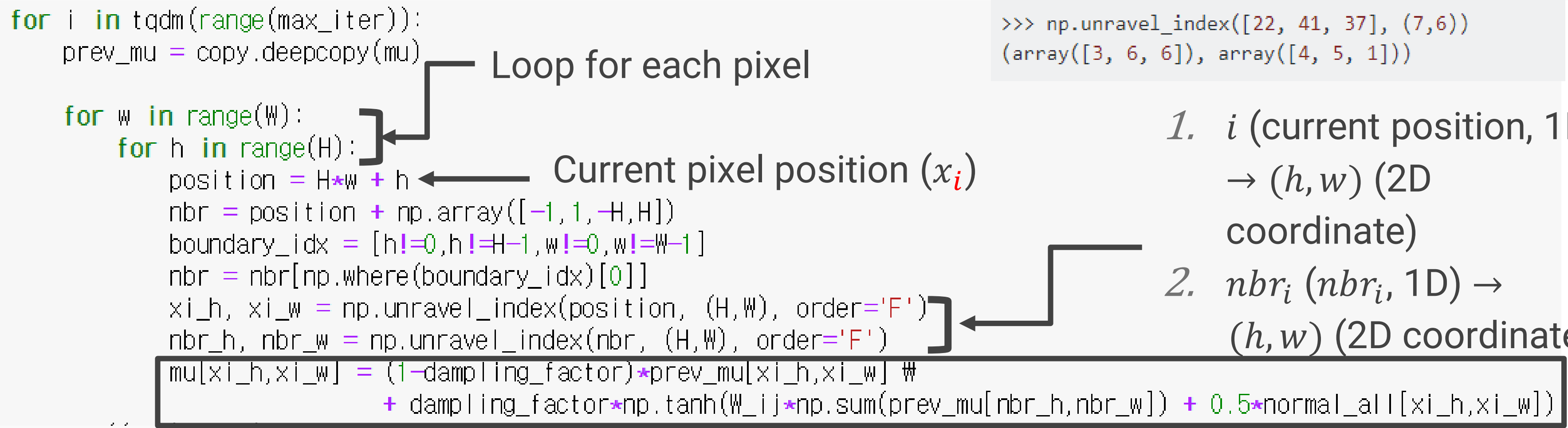
1. i (current position, 1D) $\rightarrow$ (h, w) (2D coordinate)
2. nbr_i (nbr_i, 1D) $\rightarrow$ (h, w) (2D coordinate)

Part 1. Graphical Models (code)

- Mean field variational inference



```
[[ 0,   1,   2,   3,   4,   5],
 [ 6,   7,   8,   9,  10,  11],
 [12,  13,  14,  15,  16,  17],
 [18,  19,  20,  21, *22*, 23],  <- (3, 4)
 [24,  25,  26,  27,  28,  29],
 [30,  31,  32,  33,  34,  35],
 [36, *37*, 38,  39,  40, *41*]]
     (6, 1)              (6,5)

>>> np.unravel_index([22, 41, 37], (7,6))
(array([3, 6, 6]), array([4, 5, 1]))
```



```
for i in tqdm(range(max_iter)):
    prev_mu = copy.deepcopy(mu)

    for w in range(W):
        for h in range(H):
            position = H*w + h
            nbr = position + np.array([-1,1,-H,H])
            boundary_idx = [h!=0,h!=H-1,w!=0,w!=W-1]
            nbr = nbr[np.where(boundary_idx)[0]]
            xi_h, xi_w = np.unravel_index(position, (H,W), order='F')
            nbr_h, nbr_w = np.unravel_index(nbr, (H,W), order='F')
            mu[xi_h,xi_w] = (1-dampling_factor)*prev_mu[xi_h,xi_w] ₩
                            + dampling_factor*np.tanh(W_ij*np.sum(prev_mu[nbr_h,nbr_w]) + 0.5*normal_all[xi_h,xi_w])
```

Loop for each pixel

Current pixel position (x_i)

1. i (current position, 1D) $\rightarrow (h, w)$ (2D coordinate)
2. nbr_i (nbr_i, 1D) $\rightarrow$ (h, w) (2D coordinate)

$$\mu_i^t = (1-\lambda)\mu_j^{t-1} + \lambda \tanh\left(\sum_{j\in nbr_i} \mu_j^{t-1} + 0.5 * (\log N(y_i|1) \; - \log N(y_i|-1))\right)$$

(damped update)

Reference

[1] K. Murphy, “Machine Learning: A Probabilistic Perspective”, The MIT Press, 2012
[2] https://towardsdatascience.com/variational-inference-ising-model-6820d3d13f6a
[3] https://github.com/vsmolyakov/experiments_with_python/blob/master/chp02/mean_field_mrf.ipynb

Part 2. Gaussian Process

Recall that in the simple linear regression setting.

In linear regression, we assume the outputs are a linear function of the inputs with additional noise:

$$y_t = f(x_t) + \epsilon_i \qquad \epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2)$$

$$= \beta_0 + \beta_1 x_t + \epsilon_t,$$

$$y_t = \mathbf{x}_t^\top \mathbf{w} + \epsilon_i$$

defining the vectors

$$\mathbf{x}_t = \begin{bmatrix} 1 \\ x_t \end{bmatrix}, \qquad \mathbf{w} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

$$\mathbf{X}_t = \begin{bmatrix} 1 & 0.9 \\ 1 & 3.8 \\ \vdots & \vdots \\ 1 & 9.6 \end{bmatrix}, \qquad \mathbf{y}_t = \begin{bmatrix} 0.1 \\ 1.2 \\ \vdots \\ 1.2 \end{bmatrix}$$

To predict the output for x* , we need to estimate the weights from the previous observations.

If we use a Gaussian prior over the weights *p*(w) = $\mathcal{N}$(0, Σ) and the Gaussian likelihood,

$$p(\mathbf{y}_t | \mathbf{X}_t, \mathbf{w}) = \mathcal{N}(\mathbf{X}_t^\top \mathbf{w}, \sigma_\epsilon^2 \mathbf{I})$$

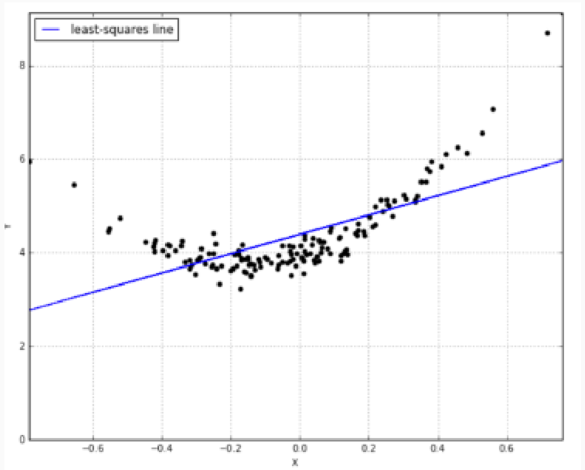
$$p(\mathbf{w} | \mathbf{y}_t, \mathbf{X}_t) \propto p(\mathbf{y}_t | \mathbf{X}_t, \mathbf{w}) p(\mathbf{w})$$

$$= \mathcal{N}\left(\frac{1}{\sigma_\epsilon^2} \mathbf{A}_t^{-1} \mathbf{X}_t \mathbf{y}_t, \mathbf{A}_t^{-1}\right)$$

$$p(f_\star | \mathbf{x}_\star, \mathbf{X}_t, \mathbf{y}_t) = \int p(f_\star | \mathbf{x}_\star, \mathbf{w}) p(\mathbf{w} | \mathbf{X}_t, \mathbf{y}_t) d\mathbf{w}$$

$$= \mathcal{N}\left(\frac{1}{\sigma_\epsilon^2} \mathbf{x}_\star^\top \mathbf{A}_t^{-1} \mathbf{X}_t \mathbf{y}_t, \mathbf{x}_\star^\top \mathbf{A}_t^{-1} \mathbf{x}_\star\right)$$

Part 2. Gaussian Process



$$y_t = f(x_t) + \epsilon_i \qquad \epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2)$$

$$= \beta_0 + \beta_1 x_t + \epsilon_t,$$

But what if we don’t want to specify upfront how many parameters are involved?

We’d like to consider every possible function that matches our data, with however many parameters are involved.

That’s what non-parametric means: it’s not that there aren’t parameters, it’s that there are infinitely many parameters.

$$y_t = f(x_t) + \epsilon_i \qquad \epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2)$$

$$= \beta_0 + \beta_1 x_t + \epsilon_t,$$

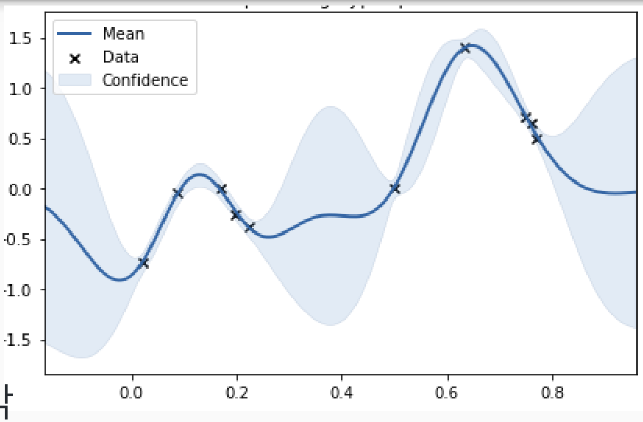
$$f(x) \sim GP(m(x), k(x, x'))$$

http://katbailey.github.io/post/gaussian-processes-for-dummies/

Part 2. Gaussian Process

1. Bayesian Inference + Gaussian Process

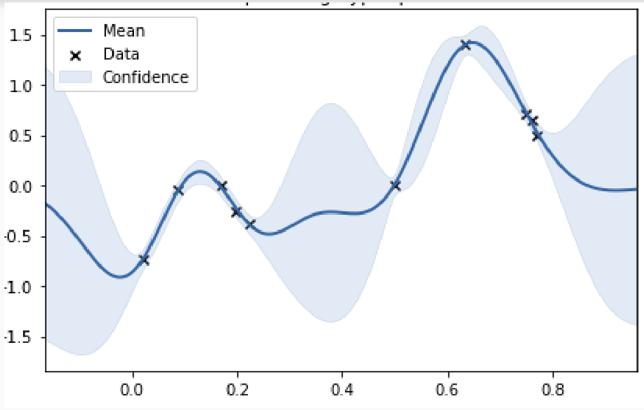
Bayesian inference를 통해 불확실성을 알 수 있다.
+ Gaussian Process는 함수 자체에 대한 Prior를 정의한다
= 함수 자체에 대한 불확실성을 볼 수 있다.



2. Gaussian distribution에서 뽑아 낸 랜덤 변수는 벡터이지만,
Gaussian process에서는 랜덤 함수 f(x)가 뽑혀져 나오며 이 함수는 제각각
각자의 평균과 분산을 갖습니다.

$$f(x) \sim GP(m(x), k(x, x'))$$

Part 2. Gaussian Process



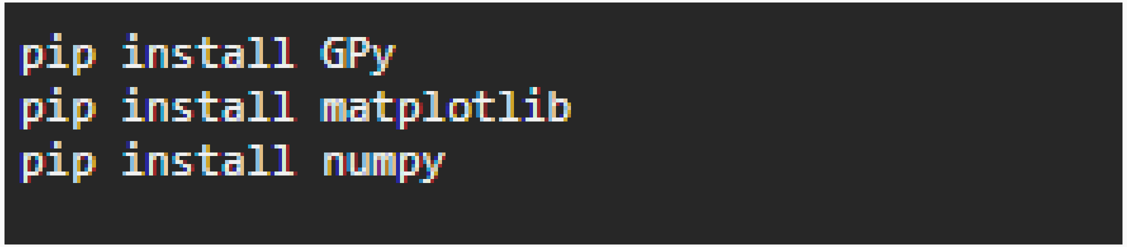
$$exp(-\frac{1}{2}|x_p - x_q|^2)$$

$$f(x) \sim GP(m(x), k(x, x'))$$

3. Regression에서 **Training data 간의 linear correlation이 높으면, training output 역시 높은 correlation을 갖는다.** Kernel을 통해 이를 고려한 gaussian process를 만들 수 있다

4. Covariance function(kernel) 은 아래의 식과 같이 similarity 라고도 볼 수 있는 output 을 뽑아낸다. 즉, 두 점이 가까우면 covariance function 의 값이 높고, 두 점이 멀면 covariance function의 값이 낮다.

Part 2. Gaussian Process



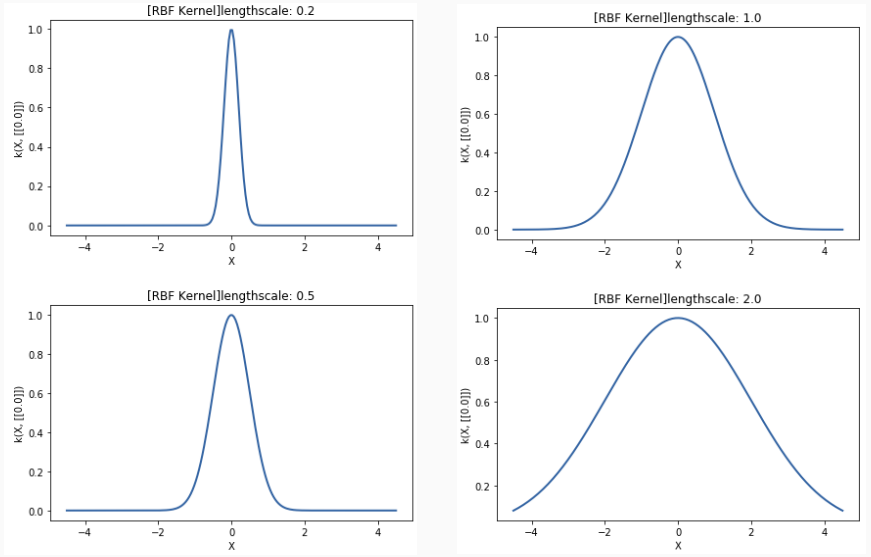
```
pip install GPy
pip install matplotlib
pip install numpy
```

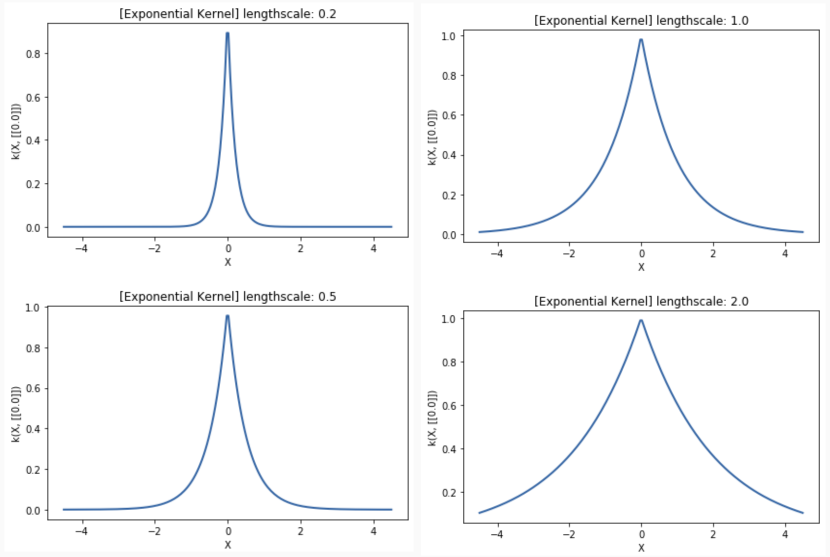
1. Covariance function 의 종류, hyper-parameter 의 영향 overview
2. mean function 과 covariance function 을 정하여 GP로 정의된 Prior를 확인합니다.
3. 관찰된 데이터를 통해 Posterior를 확인합니다.
4. 관찰된 데이터에 최적화된 kernel의 hyper-parameter로 optimization 합니다.

Part 2. Gaussian Process

0. 다양한 커널에 대한 overview 및 intuition 획득

(1) Kernel 종류에 따른 covariance function의 변화를 확인.
(2) Kernel에서의 length scale에 따른 covariance function의 변화를 확인

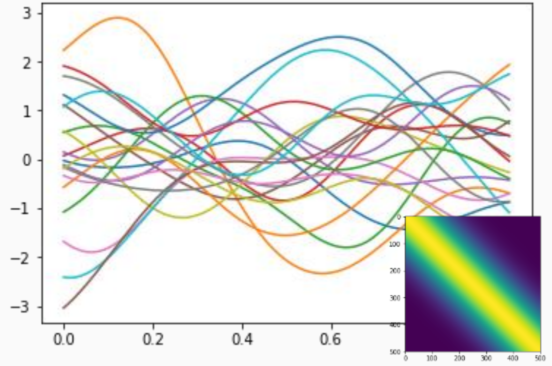
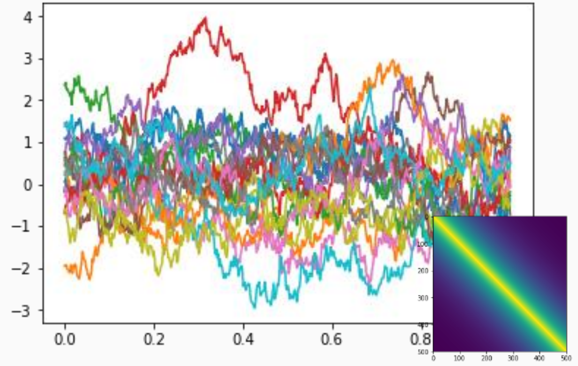
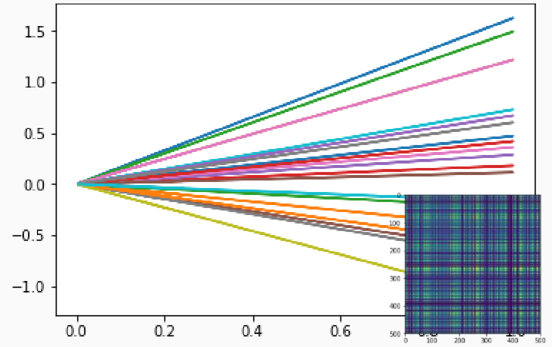
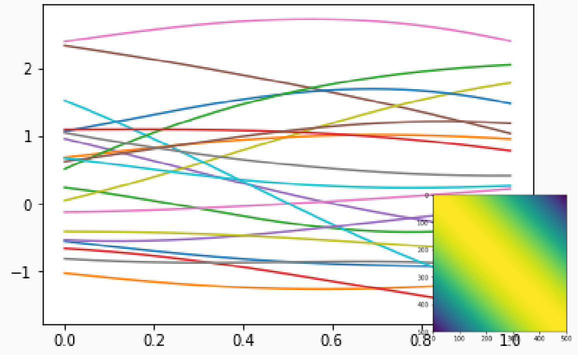




Part 2. Gaussian Process

0. 다양한 커널에 대한 overview 및 intuition 획득

(3) Sample paths from a GP.

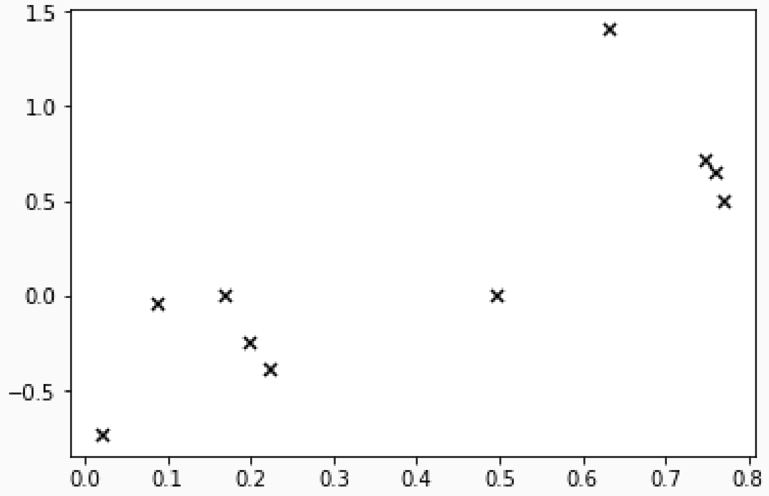





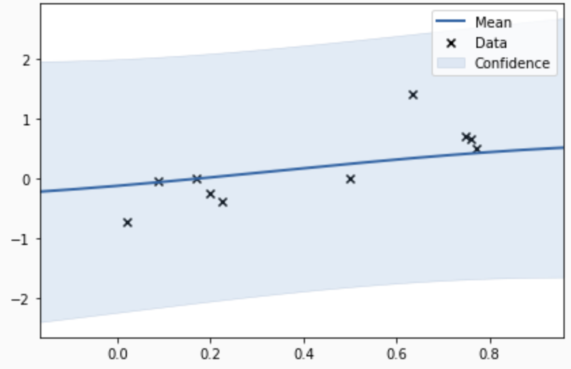
1. RBF kernel
2. Exponential kernel
3. Linear kernel
4. Exponential Quadratic kernel

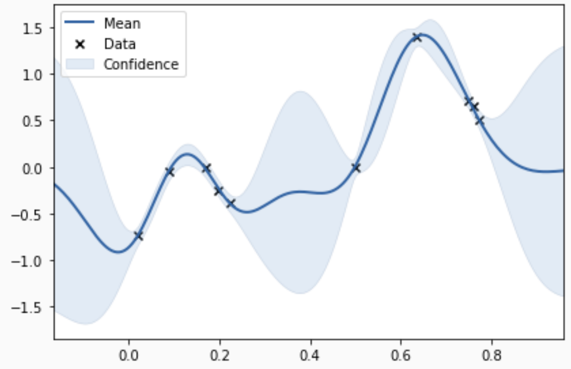
Part 2. Gaussian Process

1. 1D regression에 대하여 여러커널에 적합한 데이터셋을 생성 및 학습.

(1) GP regression model (RBF kernel)



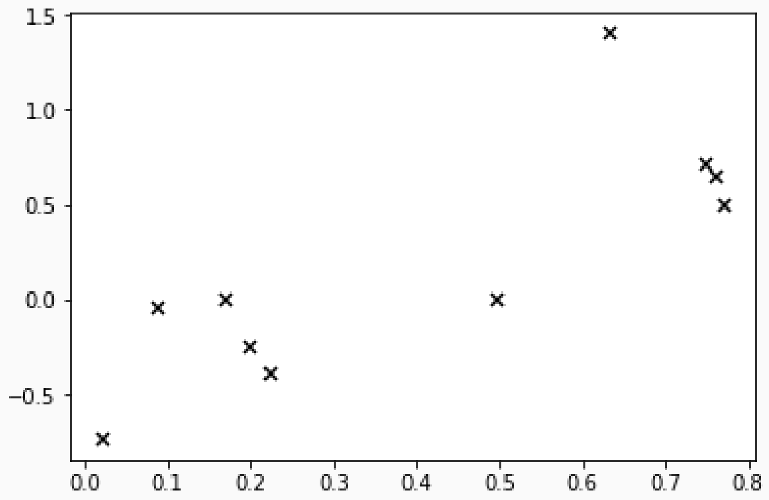


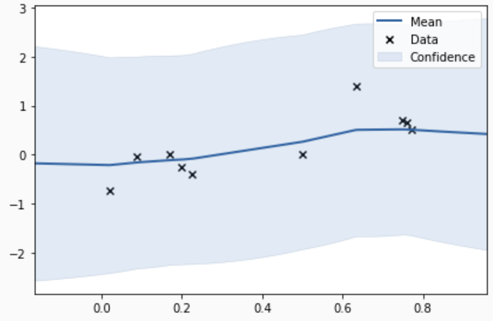


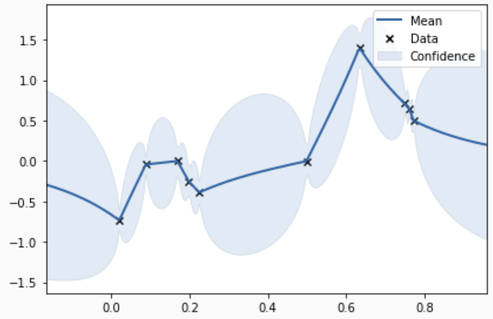
Part 2. Gaussian Process

1. 1D regression에 대하여 여러커널에 적합한 데이터셋을 생성 및 학습.

(2) GP regression model (Exponential kernel)



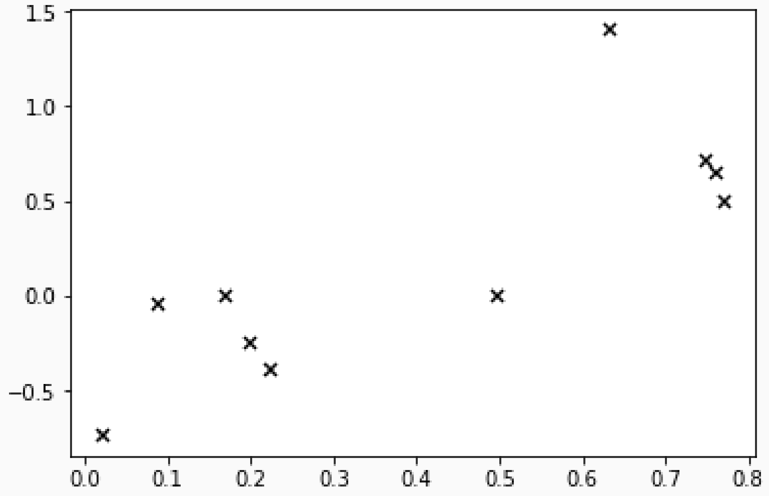


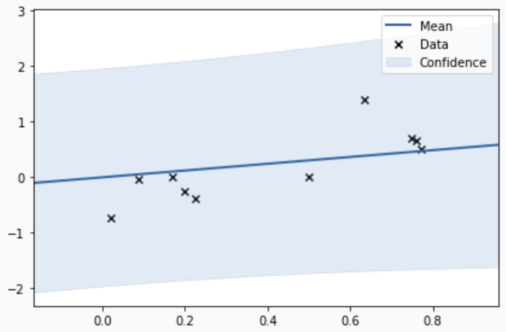


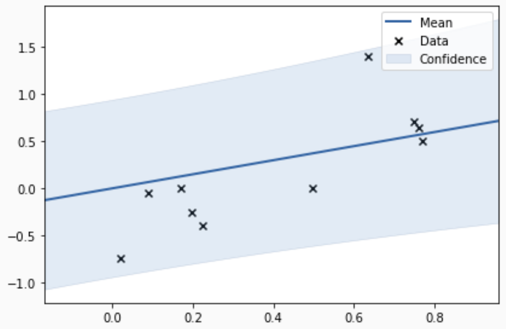
Part 2. Gaussian Process

1. 1D regression에 대하여 여러커널에 적합한 데이터셋을 생성 및 학습.

(3) GP regression model (Linear kernel)



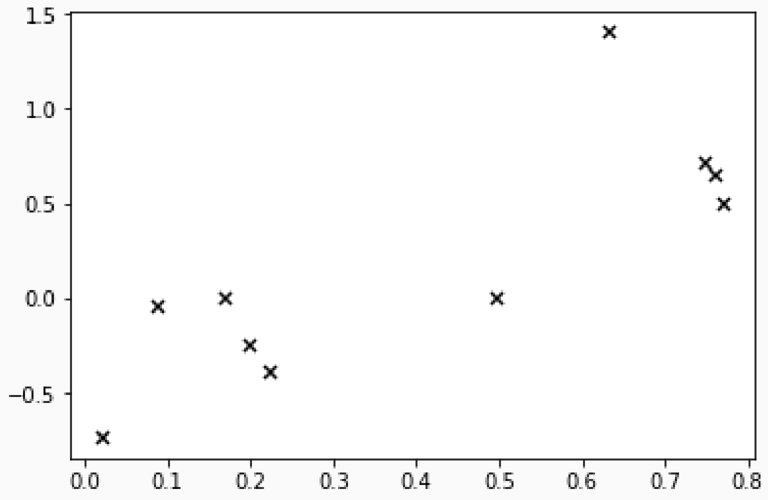


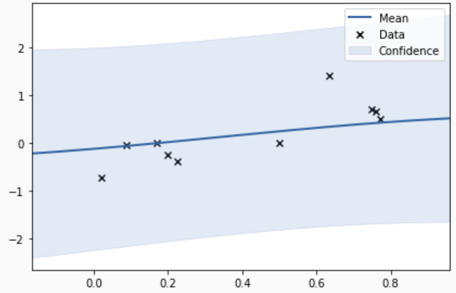


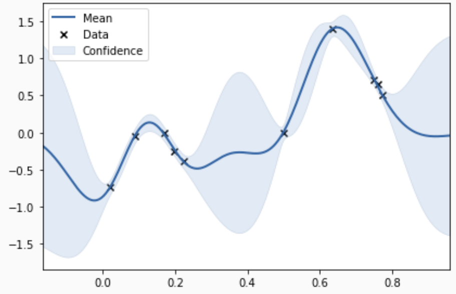
Part 2. Gaussian Process

1. 1D regression에 대하여 여러커널에 적합한 데이터셋을 생성 및 학습.

(4) GP regression model (Exponential Quadratic kernel)







Part 3. Hawkes Process

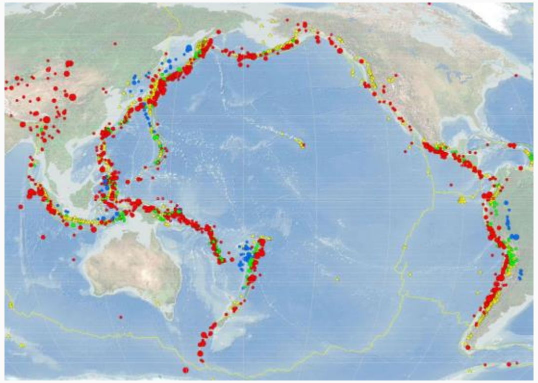
Overview

- Point Process
- Poisson Process
- Hawkes Process
- Predicting Retweet Dynamics

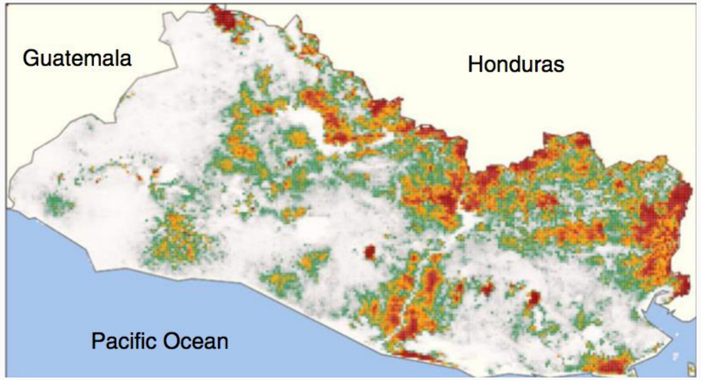
* This part refers to the ICML 2018 tutorial *Learning with Temporal Point Processes* (Rodriguez and Valera, 2018)
* Also, predicting Retweet dynamics refers to *TideH* (Kobayashi and Lambiotte, 2016).

Part 3. Hawkes Process

Many discrete events in continuous time !

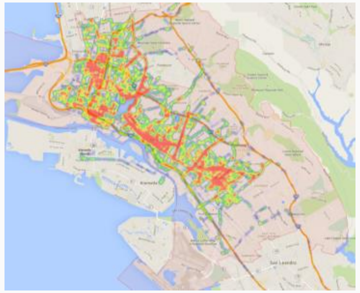

Earthquake

Online actions



Disease dynamics

Financial trading


Mobility dynamics

Part 3. Hawkes Process

Variety of processes behind these events

Stock trading

Flu spreading

Article creation in Wikipedia

News spread in Twitter

Reviews and sales in Amazon

Ride-sharing requests

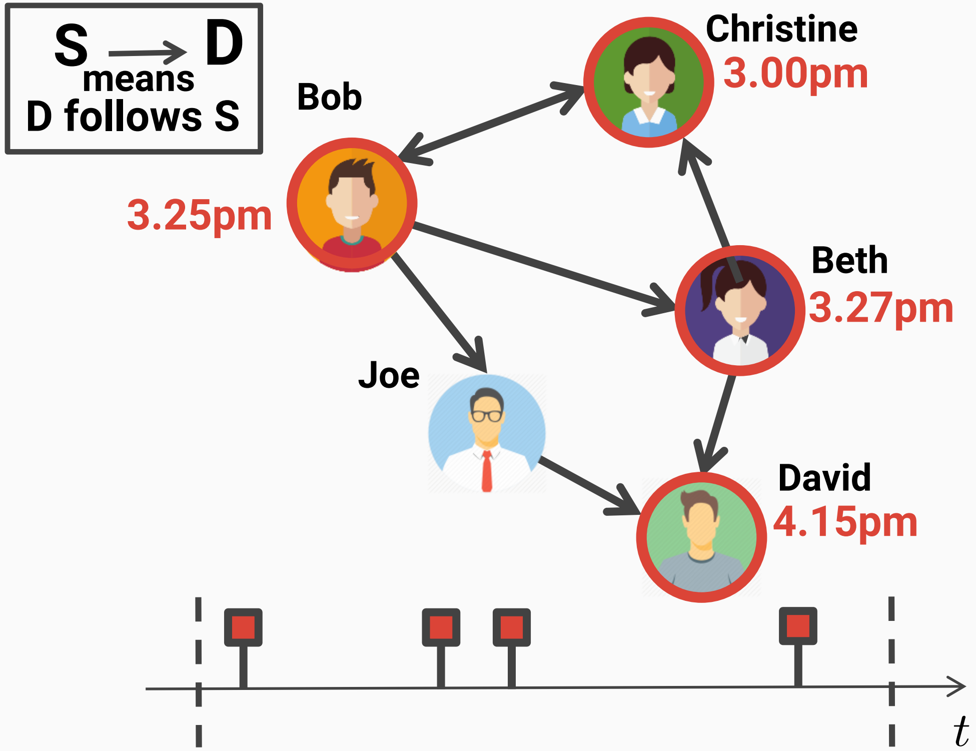
A user's reputation in Quora

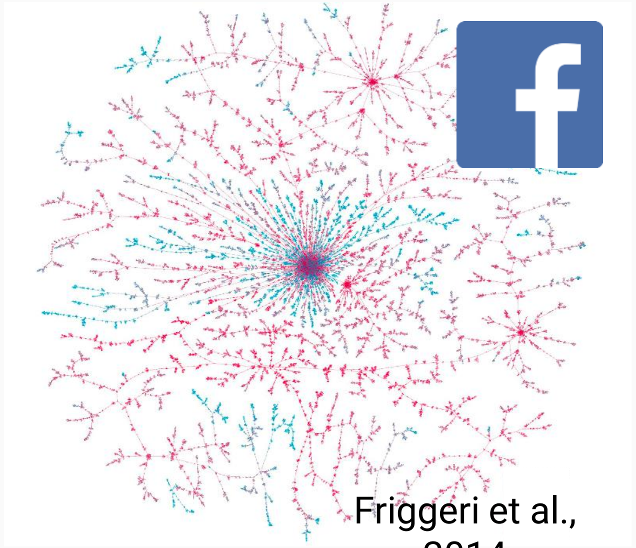
FAST → SLOW

...in a wide range of temporal scales.

Part 3. Hawkes Process

Real World Examples



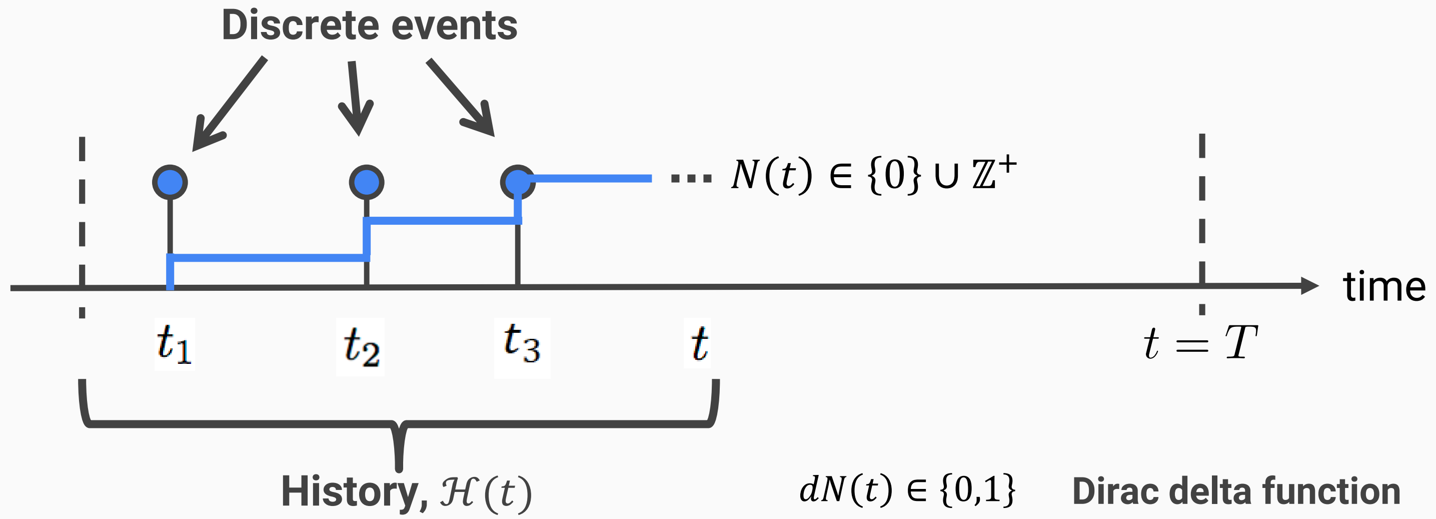


theguardian

Click and elect: how fake news helped Donald Trump win a real election

Part 3. Hawkes Process

Temporal point process is a random process whose realization consists of discrete events localized in time

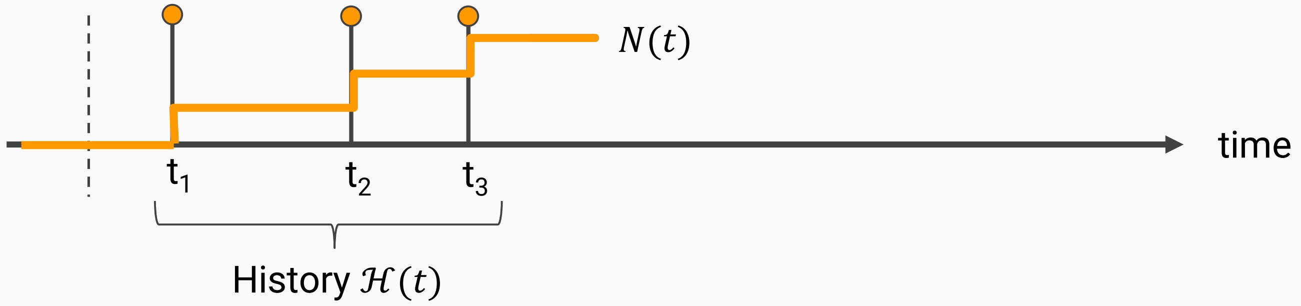

Discrete events
$\cdots N(t) \in \{0\} \cup \mathbb{Z}^+$
time
t_1
t_2
t_3
t
$t = T$
History, $\mathcal{H}(t)$

$dN(t) \in \{0,1\}$ **Dirac delta function**

Formally: $N(t) = \int_0^t dN(s) \Rightarrow dN(t) = \sum_{t_i \in \mathcal{H}(t)} \delta(t - t_i) dt$

Part 3. Hawkes Process

Temporal Point process can be represented as Intensity function



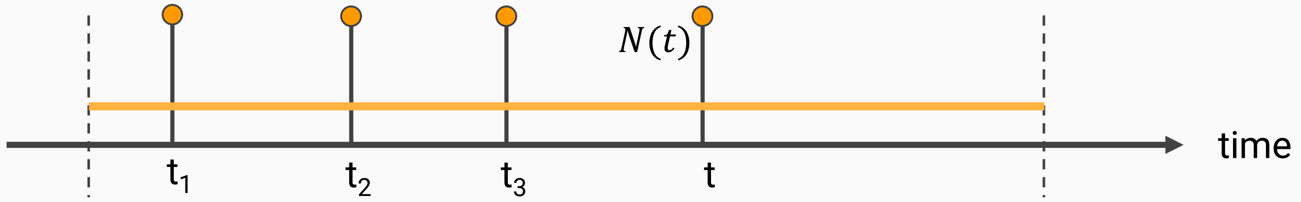
Since it is cumbersome to model event counts over time, we model event **intensity** over time.

$$\underline{\lambda^*(t)}dt = \mathbb{E}[dN(t)|\mathcal{H}(t)]$$

It is a rate = # of events / unit of time

Part 3. Hawkes Process

Poisson process



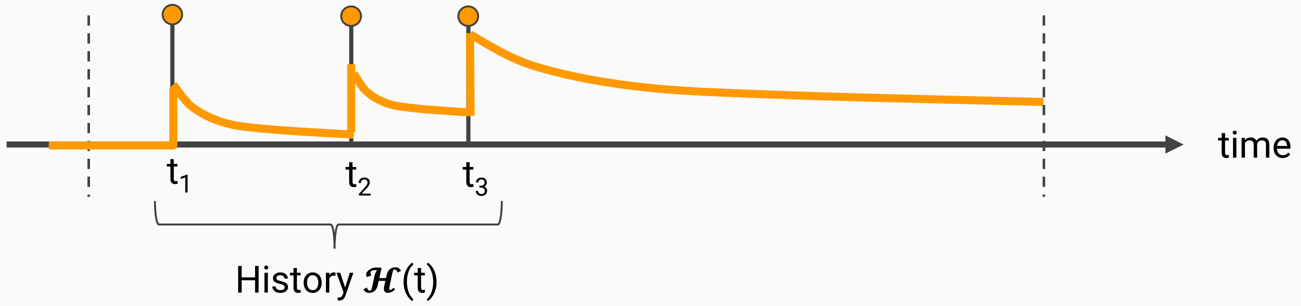
Intensity of a Poisson process

$$\lambda^*(t) = \mu$$

Observations:

1. Intensity independent of history
2. Uniformly random occurrence
3. Time interval follows exponential distribution

Hawkes process: Self-exciting intensity function



Self-exciting: each arrival increase the rate of future arrivals for some period

Intensity of Hawkes process:

$$\lambda^*(t) = \overbrace{\mu}^{\text{Baseline}} + \alpha \sum_{t_i<t} \overbrace{\phi(t-t_i)}^{\text{Triggering kernel}}$$

Observations:

1. Clustered (or bursty) occurrence of events
2. Intensity is stochastic and history dependent

Part 3. Hawkes Process

Tick: Python Library for statistical learning about point processes

How to Install

```
cogito@digits-1:~$ pip install tick
cogito@digits-1:~$ pip install --upgrade scipy==1.0.0 # optional
cogito@digits-1:~$ pip install dill # optional
```

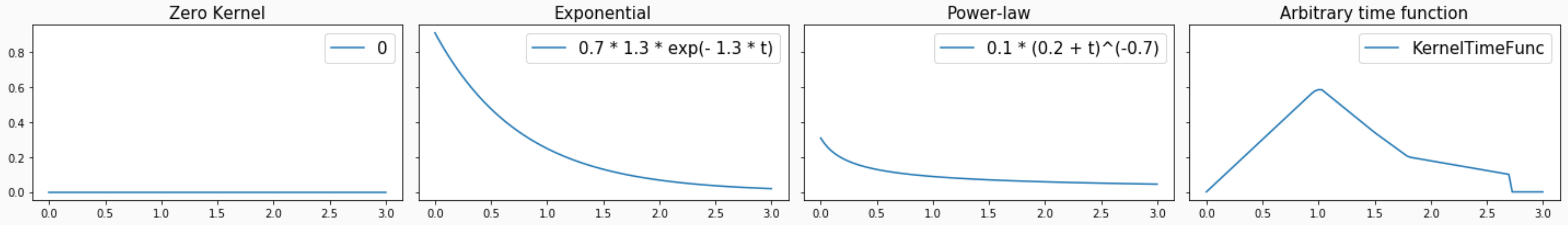
Part 3. Hawkes Process

Tick: Python Library for statistical learning about point processes

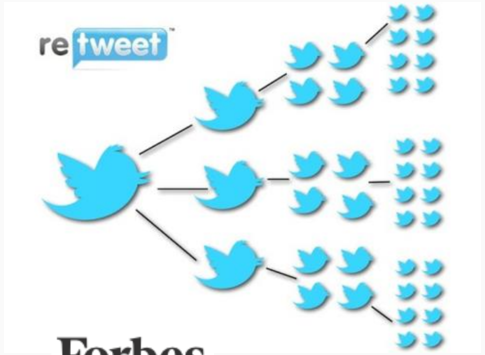
How to Use - Simulation

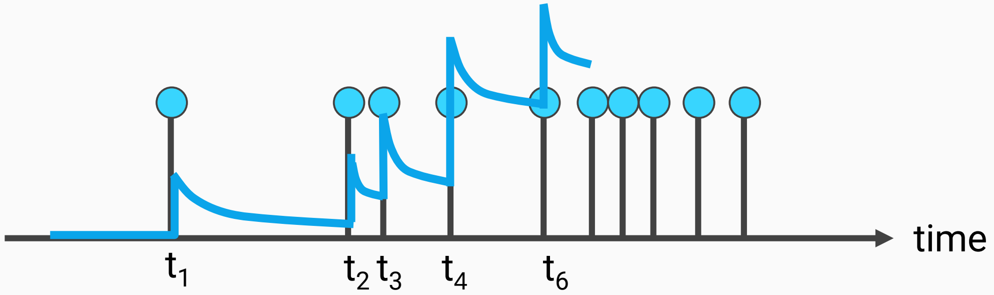
```python
from tick.hawkes import HawkesKernel0, HawkesKernelExp, HawkesKernelPowerLaw, HawkesKernelTimeFunc
...
kernel_exp = HawkesKernelExp(.7, 1.3) # HawkesKernelExp(intensity, decay)
t_values = np.linspace(0, 3, 100)
kernel_exp.get_values(t_values)) # Simulation using get_values([times])
```

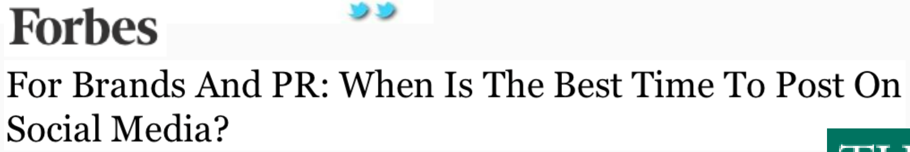
Let’s run the sample code !



Predicting Retweet Dynamics using Hawkes Process






Forbes

For Brands And PR: When Is The Best Time To Post On Social Media?

THE HUFFINGTON POST

THE BLOG

The Best Times to Post on Social Media

Kobayashi, R., & Lambiotte, R. (2016, March). Tideh: Time-dependent hawkes process for predicting retweet dynamics. In *Tenth International AAAI Conference on Web and Social Media*.

Part 3. Predicting Retweet Dynamics using Hawkes Process

Get ready for the codes!

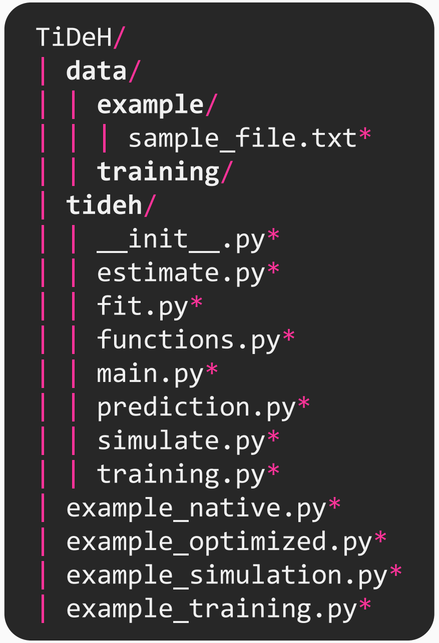
https://github.com/NII-Kobayashi/TiDeH

```
cogito@digits-1:~$ git clone https://github.com/NII-Kobayashi/TiDeH
Cloning into 'TiDeH'...
remote: Enumerating objects: 132, done.
remote: Total 132 (delta 0), reused 0 (delta 0), pack-reused 132
Receiving objects: 100% (132/132), 1.61 MiB | 310.00 KiB/s, done.
Resolving deltas: 100% (9/9), done.
Checking connectivity... done.
cogito@digits-1:~$ cd TiDeH/
```

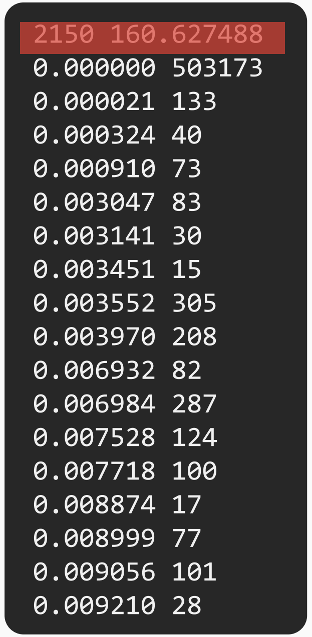
Kobayashi, R., & Lambiotte, R. (2016, March). Tideh: Time-dependent hawkes process for predicting retweet dynamics.
In *Tenth International AAAI Conference on Web and Social Media*.

Part 3. Predicting Retweet Dynamics using Hawkes Process

What you will see is ...



```
TiDeH/
| data/
| | example/
| | | sample_file.txt*
| | training/
| tideh/
| | __init__.py*
| | estimate.py*
| | fit.py*
| | functions.py*
| | main.py*
| | prediction.py*
| | simulate.py*
| | training.py*
| example_native.py*
| example_optimized.py*
| example_simulation.py*
| example_training.py*
```



```
2150 160.627488
0.000000 503173
0.000021 133
0.000324 40
0.000910 73
0.003047 83
0.003141 30
0.003451 15
0.003552 305
0.003970 208
0.006932 82
0.006984 287
0.007528 124
0.007718 100
0.008874 17
0.008999 77
0.009056 101
0.009210 28
```

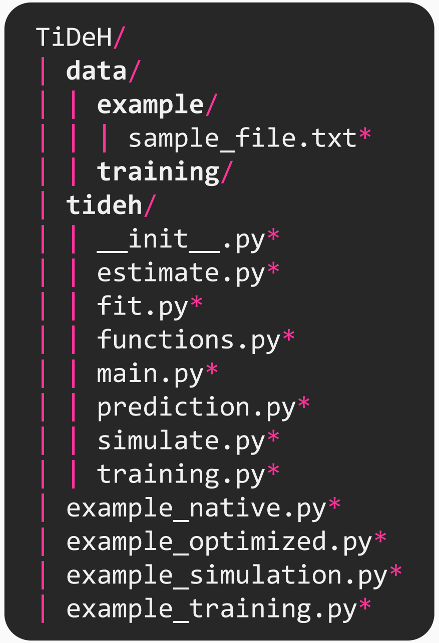
* one file per tweet
* space separated
* only tweets with more than 2000 retweets
were used

First row

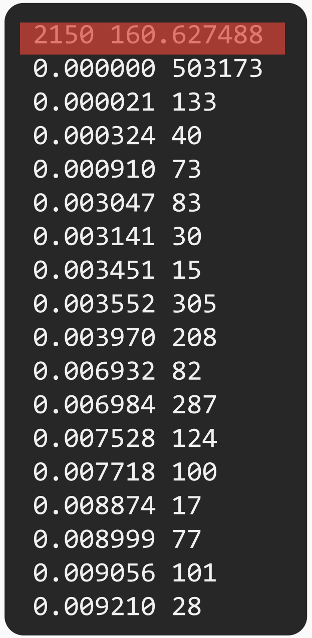
- <number of total retweets>
  <start time of tweet in days>

Part 3. Predicting Retweet Dynamics using Hawkes Process

What you will see is …



```
TiDeH/
| data/
| | example/
| | | sample_file.txt*
| | training/
| tideh/
| | __init__.py*
| | estimate.py*
| | fit.py*
| | functions.py*
| | main.py*
| | prediction.py*
| | simulate.py*
| | training.py*
| example_native.py*
| example_optimized.py*
| example_simulation.py*
| example_training.py*
```



```
2150 160.627488
0.000000 503173
0.000021 133
0.000324 40
0.000910 73
0.003047 83
0.003141 30
0.003451 15
0.003552 305
0.003970 208
0.006932 82
0.006984 287
0.007528 124
0.007718 100
0.008874 17
0.008999 77
0.009056 101
0.009210 28
```

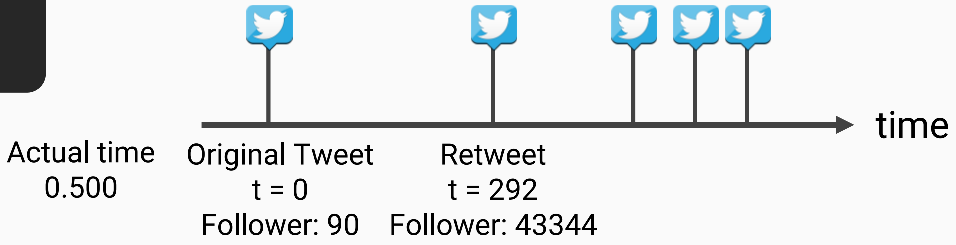
* one file per tweet
* space separated
* only tweets with more than 2000 retweets were used

First row

- <number of total retweets> <start time of tweet in days>

Every other row

- <relative tweeted/retweeted time in seconds> <number of followers of retweeting person>


time
Actual time 0.500
Original Tweet t = 0 Follower: 90
Retweet t = 292 Follower: 43344

Part 3. Predicting Retweet Dynamics using Hawkes Process

How to run the example code

```
cogito@digits-1:~/TiDeH$ python3 example_native.py
```

Example of predicting future retweet activity. Given an observation time, the parameters of the infectious rate are estimated and then, the number of retweets before a given time is predicted.

Then, you will see the results

```
Estimated parameters are:
p0: 0.001
r0: 0.414
phi0: 0.140
tm: 1.822
Average % error (estimated to fitted): 12.33
Predicted number of retweets from 48 to 168 hours: 37
Predicted number of retweets at hour 168: 2170
Prediction error (absolute): 20
```

Part 3. Predicting Retweet Dynamics using Hawkes Process

How to run the example code

Let’s look at the code *example_native.py*

```python
# Module import
from tideh import estimate_parameters
from tideh import load_events
from tideh import predict
# Read dataset
filename = 'data/example/sample_file.txt'
(_, start_time), events = load_events(filename)
```

Part 3. Predicting Retweet Dynamics using Hawkes Process

How to run the example code

Let's look at the code *example_native.py*

```
# additional parameters passed to infectious rate function
add_params = {'t0': start_time, 'bounds': [(-1, 0.5), (1, 20.)]}
obs_time = 48 # observation time of 2 days
pred_time = 168 # predict for one week
params, err, _ = estimate_parameters(events=events, obs_time=obs_time, **add_params)
```

The probability for getting a retweet in a small-time interval $[t, t + \Delta t]$ is $\lambda(t)\Delta t$.

$$\lambda(t) = \underbrace{p(t)}_{\text{Infectious rate}} \sum_{i:t_i<t} \underbrace{d_i}_{\text{\# Followers}} \overbrace{\phi(t - t_i)}^{\text{Exciting kernel}}$$

Part 3. Predicting Retweet Dynamics using Hawkes Process

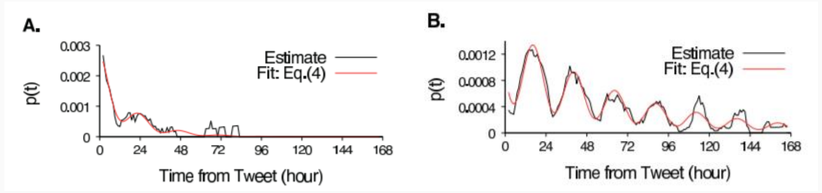
How to run the example code

Let's look at the code *example_native.py*

```
# additional parameters passed to infectious rate function, bounds for r0 and taum
add_params = {'t0': start_time, 'bounds': [(-1, 0.5), (1, 20.)]}
obs_time = 48 # observation time of 2 days
pred_time = 168 # predict for one week
params, err, _ = estimate_parameters(events=events, obs_time=obs_time, **add_params)
```

Infectious rate is

$$p(t) = p_0 \left\{1 - r_0 \sin\left(\frac{2\pi}{T_m}(t + \phi_0)\right)\right\} e^{-(t-t_0)/\tau_m}$$



- t_0: time of the original tweet
- $T_m = 1$ day: period of oscillation
- p_0: intensity
- r_0 : the relative amplitude of the oscillation
- ϕ_0: its phase
- τ_m: characteristic time of popularity decay

Part 3. Predicting Retweet Dynamics using Hawkes Process

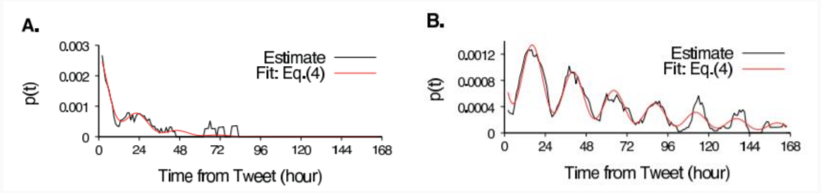
How to run the example code

Let’s look at the code *example_native.py*

```python
# additional parameters passed to infectious rate function, bounds for r0 and taum
add_params = {'t0': start_time, 'bounds': [(-1, 0.5), (1, 20.)]}
obs_time = 48 # observation time of 2 days
pred_time = 168 # predict for one week
params, err, _ = estimate_parameters(events=events, obs_time=obs_time, **add_params)
```

Infectious rate is

$$p(t) = p_0 \left\{ 1 - r_0 \sin\left(\frac{2\pi}{T_m} (t + \phi_0) \right) \right\} e^{-(t-t_0)/\tau_m}$$



- t_0: time of the original tweet
- $T_m = 1$ day: period of oscillation
- p_0: intensity
- r_0 : the relative amplitude of the oscillation
- ϕ_0: its phase
- τ_m: characteristic time of popularity decay

Part 3. Predicting Retweet Dynamics using Hawkes Process

How to run the example code

Let's look at the code *example_native.py*

```
# tideh/main.py - Estimate of model parameters of TiDeH (p_0, r_0, phi_0, t_m).
def estimate_parameters(events, obs_time=None, window_size=4, window_stride=1,
                        kernel_int=functions.integral_zhao,
                        p=functions.infectious_rate_tweets, values=None, **p_params):
    obs_time = int(min(obs_time, events[len(events) - 1][0]))
    events = [e for e in events if e[0] < obs_time] # e[0] holds time of the event
    estimations, window_event_count, window_middle = \
        estimate.estimate_infectious_rate(events, kernel_int, obs_time, window_size, window_stride)
    fitted = fit.fit_parameter(estimates=estimations, fun=lambda x, *args: p(x, *args, **p_params),
    start_values=values, xval=window_middle)
```

```
# tideh/estimate.py - Esitmate base rate p0
def estimate_infectious_rate(events, kernel_integral=functions.integral_zhao, obs_time=24, window_size=4,
                             window_stride=1):
for start in range(0, obs_time - window_size + window_stride, window_stride):
    ...
    est = estimate_infectious_rate_constant(events=events_tmp, t_start=start, t_end=end,
                                            kernel_integral=kernel_integral, count_events=count_current)
    ...
```

Part 3. Predicting Retweet Dynamics using Hawkes Process

How to run the example code

Let’s look at the code *example_native.py*

```
# tideh/main.py
def estimate_parameters(events, obs_time=None, window_size=4, window_stride=1,
                        kernel_int=functions.integral_zhao,
                        p=functions.infectious_rate_tweets, values=None, **p_params):
    obs_time = int(min(obs_time, events[len(events) - 1][0]))
    events = [e for e in events if e[0] < obs_time] # e[0] holds time of the event
    estimations, window_event_count, window_middle = \
        estimate.estimate_infectious_rate(events, kernel_int, obs_time, window_size, window_stride)
    fitted = fit.fit_parameter(estimates=estimations, fun=lambda x, *args: p(x, *args, **p_params),
    start_values=values, xval=window_middle)
```

```
# tideh/fit.py - Estimate of model parameters r_0, phi_0, t_m
from scipy.optimize import leastsq
def fit_parameter(estimates, fun, start_values, xval):
    if start_values is None:
        start_values = np.array([0, 0, 0, 1.])
    return leastsq(func=loss_function, x0=start_values, args=(estimates, fun, xval))[0]
```

Part 3. Predicting Retweet Dynamics using Hawkes Process

How to run the example code

Then, where is the exciting kernel?

```
# additional parameters passed to infectious rate function
add_params = {'t0': start_time, 'bounds': [(-1, 0.5), (1, 20.)]}
obs_time = 48 # observation time of 2 days
pred_time = 168 # predict for one week
params, err, _ = estimate_parameters(events=events, obs_time=obs_time, **add_params)
```

The probability for getting a retweet in a small-time interval $[t, t+\Delta t]$ is $\lambda(t)\Delta t$.

$$\lambda(t) = p(t) \sum_{i:t_i<t} d_i \overline{\phi(t-t_i)} \longleftarrow \text{Exciting kernel}$$

Exciting kernel is fitted to the empirical data by the function where is the heavily tailed in a variety of social networks (Crane, et al. 2008, Masuda, et al. 2013)

$$\phi(s) = \begin{cases} 0 & (s<0) \\ c_0 & (0 \le s \le s_0), \\ c_0(s/s_0)^{-(1+\theta)} & (\text{Otherwise}) \end{cases}$$

- $c_0 = 6.49 \times 10^{-4}$ seconds
- $s_0 = 300$ seconds
- $\theta = 0.242$

Part 3. Predicting Retweet Dynamics using Hawkes Process

We are going to estimate the exciting kernel from the dataset.

Then, where is the exciting kernel?

$$\phi(s) = \begin{cases} 0 & (s < 0) \\ c_0 & (0 \leq s \leq s_0) \\ c_0 (s/s_0)^{-(1+\theta)} & (\text{Otherwise}) \end{cases},$$

- $c_0 = 6.49 \times 10^{-4}$ seconds
- $s_0 = 300$ seconds
- $\theta = 0.242$

Now, let's estimate the kernel !