Signal

CH6

Event and Feature-Finding

6.1) Find Simple Features

to acces single event : logical indexing / np.where and np.nonzero

3 example are given in the pages 106-112

1) Find large Signal values in 1D data 2) ○ start and end of a movement

3) ○ bright pixels in Grayscale image

6.2) Cross Correlation

6.2.1) Comparing Signals

To find similarities we need similarity function. → max when similar, min when not

Dot product satisfies this. signal (dot) feature.

6.2.2) Auto correlation

If two signals being compared are the same the result is called auto-correlation function. it is not used to find events. It is used for finding the peridicity. After accounting for mean offset, the auto-correlation is also used to detect the energy in the signal since the energy in a harmonic oscillator is proportional to the square of the amplitude.

6.2.3) Normalization

Normalization has to account for 3 aspects of the signal:

Offset: To eliminate effects from constant offset we can substract mean of the signal / smallest value of the signal.

Duration: to ensure that 2 signals have same length we can interpolate them.

Amplitude: $$sig_{normalized} = \frac{sig_{raw}}{\sqrt{max(autocorr(sig_{raw}))}}$$

6.2.4) Math

Cross correlation funct of x and y can be obtaind by:

$$R_{xy}(m) = \begin{cases} \sum_{i=0}^{n-m-1} x_{m+i} * y_i^* & , 0 \leqslant m < n-1 \\ R^*_{yx}(-m) & , 1-n < m < 0 \end{cases}$$

to optimize speed cross corr is implemented using FFT not the function above.

6.2.5) Features of cross corr funct.

length of cross corr funct: if signal has n points and pattern has m, length is n+m-1.

Maximum cross corr funct: if signal × a and pattern × b, max of cross corr funct increases by a factor a·b. so for auto corr, → $a \times a = a^2$

6.2.6) Cross correlation and Convolution

There is a strong relationship between cross corr, convolution and FIR-filters.

- convolution of a signal x w/ a kernel w is equivalent to applying an FIR filter w/ weights w to a signal x.
- Apart from a trivial shift in the index, the commands np.correlate(x, y, "full") and np.convolve(x, y[::-1]) produces the same output.

6.3) interpolation

6.3.1) Linear interpolation

not very accurate, 1st derivative not continuous at data points.

6.3.2) Cubic Spline interpolation

CH7) Statistics

CH8) Parameter Fitting

8.1) Correlation

8.1.1) Corr Coef

→ measure of linear corr (or dependence) between 2 variable x and y.

For sample data x_i and y_i;

$$r = \sum_{i=0}^{n-1} \left(\frac{x_i - \bar{x}}{\sqrt{\sum_{j=0}^{n-1} (x_j - \bar{x})^2}} \times \frac{y_i - \bar{y}}{\sqrt{\sum_{k=0}^{n-1} (y_k - \bar{y})^2}} \right)$$

- Symmetric in x and y
- only quantifies how well points lie on a straight line, and if that line is raising or falling

8.1.2) Coef of Determination

For linear regression r^2 is called coef of determination. it quantifies how well the fitted data account for raw data.

$$SS_{tot} = SS_{model} + SS_{resid}$$

$$SS_{tot} = \sum_i (y_i - \bar{y})^2 \qquad SS_{model} = \sum_i (\hat{y}_i - \bar{y})^2 \qquad SS_{resid} = \sum_i (y_i - \hat{y}_i)^2$$

$$r^2 = \frac{SS_{model}}{SS_{tot}} = 1 - \frac{SS_{resid}}{SS_{total}}$$

As an example $r^2 = 0.7$ would indicate that approximately 70% of the variation in the response can be explained by line fit.

r, sometimes called Pearson's correlation coefficient, can be calculated w/ scipy.stats.pearsonr or scipy.stats.linregress

8.2) Straight Line

if $P_1 = (x_1 / y_1)$ and $P_2 = (x_2 / y_2)$, intercept and slope can be calculated:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \qquad b = \frac{y_1 x_2 - x_1 y_2}{x_2 - x_1}$$

in general case 1 dependent variable y and p+1 indep var, → β_i

$$y = \beta_0 + \sum_{i=1}^{p} \beta_i x_i$$

8.2.1) Normal Form of Line Eqn

line can be charactirized by a point on the line $\vec{x}_1$ and a vector $\vec{n}$ that is ⊥ to line

$$n \cdot x = n \cdot x_1$$

$$ax + by = c \quad w/ \quad \vec{n} = \begin{pmatrix} a \\ b \end{pmatrix} \quad \text{and} \quad c = \vec{n} \cdot \vec{x}_1$$

8.3) Line Fitting

often fitting error is taken to be the least squares estimator.

8.3.1) Residuals → data value - model value

e_k : residual $\quad y_k$: dependent var of data

$$e_k = y_k - \left(\beta_0 + \sum_{i=1}^{p} \beta_i x_{ki}\right)$$

8.3.2) Least Square Estimator

n data points, sum of squared residuals is:

$$E(x, y, \beta) = \sum_{k=0}^{n-1} e_k^2 = \sum_{k=0}^{n-1} \left(y_k - \left(\beta_0 + \sum_{i=1}^{p} \beta_i x_{ki}\right)\right)^2$$

least square estimators are the values β_i that minimise $\mathcal{E}$. To determine the value of the LSE $\hat{\beta}_i$ it is necessary to locate the min of $\mathcal{E}$ by finding where the following partial derivatives are zero :

$$0 = \frac{\partial \mathcal{E}}{\partial \beta_0}\Big|_{\hat{\beta}} = -2 \sum_{k=0}^{n-1} \left(y_k - \left(\beta_0 + \sum_{i=1}^{p} \hat{\beta}_i x_{ki} \right)\right)$$

$$0 = \frac{\partial \mathcal{E}}{\partial \beta_i}\Big|_{\hat{\beta}} = -2 \sum_{k=0}^{n-1} \left(y_k - \left(\beta_0 + \sum_{i=1}^{p} \hat{\beta}_i x_{ki} \right)\right) \cdot x_{ip} \quad \forall i = 1, \dots, p$$

it will be done using python

Ordinary LS

The method of ordinary LS can be used to find an approximate sol to over-determined systems. For the system $\vec{A} \cdot \vec{p} = y$, LS is obtained from the problem

$$\min \| A \cdot p - y \| \rightarrow p = (A^T A)^{-1} A^T \cdot y$$

8.4 | Linear fits w/ Python

8.4.2) w/ intercept :

$$y = X \cdot \beta$$

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ \vdots & \vdots & & \ddots & \\ 1 & x_{n1} & & & x_{np} \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$x = [0, 100]$ $\quad y = \beta_0 + \beta_1 x + \text{noise}$

```
M = np.column_stack([x, np.oneslike(x)]])
p_estimator = np.linalg.lstsq(M, y)[0]
slope, y_intercept = p_estimator
```

8.4.5| Sine Fit

freq: ω, amp: A, phase delay δ, offset α,

$$x = \alpha + A \sin(\omega t + \underbrace{\delta)}_{\text{not linear parameter}}$$

Can be :

$$x = \alpha + a \sin(\omega t) + b \cos(\omega t) \qquad A = \sqrt{a^2 + b^2}$$

$$\delta = \tan^{-1}\left(\frac{b}{a}\right)$$

8.4.6) Circle Fit

$$(x-x_c)^2+(y-y_c)^2=r^2$$

$$x^2-2xx_c+y^2-2yy_c+y_c^2=r^2$$

$$2xx_c+2yy_c+1\cdot(r^2-x_c^2-y_c^2)=x^2+y^2$$ where x_c, y_c - center of circle

this gives $\underbrace{(x^2+y^2)}_{\text{known}}=\underbrace{X\cdot\beta}_{\text{lin relationship}}$ w/ $X=\begin{bmatrix}1 & 2x_1 & 2y_1\\ 1 & 2x_2 & 2y_2\\ \vdots & \vdots & \vdots\end{bmatrix}$

$x_c=\beta_1 \quad y_c=\beta_2$

8.5) Confidence Intervals

8.5.1) Finding Confidence interval

To quantify the uncertanity in the best fit estimators we have to make assumptions about the noiseindata. Some commonly made assumptions are that:

- indep var $x_1 \dots x_l$ are known exactly
- residuals are roughly normally distributed
- noise is only dep var y
- res are indep of values $x_1 \dots x_l$

8.6) Fitting Nonlinear Functions

- Nonlinear fits are more efficient and accurate when user provides a decent estimate of the params as a starting point.
- If you have a choice reduce # of params to be estimated.

CH9) Spectral Signal Analysis

9.1) Transforming Data

9.2) Fourier Integral

$$X(f)=\int_{-\infty}^{\infty} x(t)\,e^{-j2\pi ft}\,dt \qquad x(t)=\int_{-\infty}^{\infty} X(f)\,e^{-j2\pi ft}\,df$$

- Complex exp Notation
  - 1) $osc(t)=(r\times e^{j\phi})\,e^{j2\pi ft}$
  - 2) $osc(t)=r\times\sin(2\pi ft+\phi)$
  - 3) $osc(t)=a\times\cos(2\pi ft)+b\sin(2\pi ft)$

$$\rightarrow e^{j2\pi ft}=\cos(2\pi ft)+j\sin(2\pi ft)$$

$$\cos(2\pi ft)=\tfrac{1}{2}\left(e^{j2\pi ft}+e^{-j2\pi ft}\right)$$

$$\sin(2\pi ft)=\tfrac{1}{2j}\left(e^{j2\pi ft}-e^{-j2\pi ft}\right)$$

$$a \cos(2\pi f t) + b \sin(2\pi f t) = c * \sin(2\pi f t + \phi)$$

$$c = \sqrt{a^2 + b^2} \qquad \phi = \tan^{-1}(b/a)$$

Ex1) Fourier transform of constant

$$x(t) = 1 \qquad X(f) = \int_{-\infty}^{\infty} e^{-j2\pi f t}\, dt = \boxed{\delta(0) = X_{\text{const}}(f)}$$

Ex2) Fourier transform of a pure oscillation

$$x(t) = A * \cos(2\pi f' t) \qquad X(f) = \int_{-\infty}^{\infty} A * \cos(2\pi f' t)\, e^{-j2\pi f t}\, dt$$

$$X(f) = \int_{-\infty}^{\infty} A * \left(\frac{e^{j2\pi f' t} + e^{-j2\pi f' t}}{2} \right) e^{-j2\pi f t}\, dt$$

$$X(f) = \int_{-\infty}^{\infty} \frac{A}{2} \left[e^{-j2\pi (f - f')t} + e^{-j2\pi (f + f')t} \right] dt$$

$$\boxed{\begin{aligned} X_{\cos}(f) &= \frac{A}{2}\left[\delta(f - f') + \delta(f + f')\right] \\ X_{\text{sine}}(f) &= \frac{A}{2j}\left[\delta(f - f')\ \delta(f + f')\right] \end{aligned}}$$

9.3) Fourier Series

$$x(t) = a_0 + \sum_{n=1}^{\infty} \left[a_n * \cos(2\pi n f_p t) + b_n * \sin(2\pi n f_p t) \right] \qquad f_p = 1/T_p \quad \text{fundamental freq}$$

9.4) DFT

N data points were sampled w/a constant freq, then the Fourier coef X_n can be obtained.

$$X_n = \sum_{\tau=0}^{N-1} x_\tau * e^{-j2\pi n\tau/N} \qquad \text{w/ } n = 0, \dots, N-1$$

the inverse Fourier transform is given by

$$x_\tau = \frac{1}{N} \sum_{n=0}^{N-1} X_n * e^{j2\pi n\tau/N} \qquad \text{w/ } \tau = 0, \dots, N-1$$

Conventions: coef depend on the sample freq. For example, using this def. the largest Fourier coef for one cycle of a pure sine wave correspond to half the number of sample points.

FFT → algorithm for DFT

If # of data points is exactly $N = 2^n$, the number of multiplications required can be reduced by many orders of magnitude, especially for long signals.

To make use of the speed benefits of the FFT, signals that contain less than 2^n data are often <u>zero-padded</u> i.e. extended w/ 0s until their length matches next 2^n.

Zero-padding in time domain is also used extensively in practice to compute heavily interpolated spectra by taking the DFT of the zero padded signal.

Real valued Signals

$$X(f) = X(-f)^*$$

For real inputs, the magnitude of Fourier spectrum is symmetrical about the Nyquist frequency.

Frequencies

$$f_n = \frac{n}{N \cdot T_s}, \quad 0 \leq n \leq N-1$$ → use np.fftfreq(len(sig), 1/rate)

Single Sided Spectrum

(np.fft.rfft, np.fft.irfft, np.fft.rfftfreq)

9.5) Spectral Density Estimation

Spectral density characterizes the freq content of the signal. (Non-parametric and parametric)

Periodogram: the modulus-squared of DFT

Welch's method: a windowed version of the periodogram that uses time averaging.

Parametric techniques are autoregressive model (AR), moving avg (MA), and ARMA)

9.5.1) Periodogram

Power: P_n $\quad P_n = F_n \cdot F_n^* = |F_n|^2$: periodogram or power spectrum of signal

to calculate it: f, Pxx = scipy.signal.periodogram(data, fs)

For periodogram and Welch's method look for Signal.ipynb

9.6) FT, Convolution and Cross-Correlation

9.6.1) Convolution

Can be calculated using FT. Convolution theorem:

$$F\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

$\mathcal{F}$ denotes Fourier transform

* convolution

· point wise multiplication

$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$ ⇒ efficient way of calculating convolution.

Remember that application of an FIR-filter w/ filter coef b is equivalent to a convolution w/ a signal b.

9.6.2) Cross Correlation

$$Corr(g,h) \leftrightarrow G \cdot H^*$$

g, h are functions of time, G, H the corresponding FT and H^* is complex conjugate of H. If h(t) is real then $H(-f) = H^*(f)$

9.7) Time Dependent FT

9.7.1) Windowing

FT calculates the freq content of whole signal. But often signals are changing their characteristic over time, → Use Short Time F T → (STFT)

To obtain time selective information from the FT, windowing can be applied to the signal.