Harris コーナー検出

初期の Computer vision におけるコーナー検出の試みとして **Chris Harris** と **Mike Stephens** が **A Combined Corner and Edge Detector** という論文で発表した **Harris コーナー** と呼ばれる方法がある。

Harris コーナー

基本的には全方向に対して画素位置 (u, v) の移動量に対する画素値の違いを見つける。

$$E(u,v) = \sum_{x,y} w(x,y) \left[I(x+u, y+v) - I(x,y) \right]^2$$

ここで、$w(x,y)$ は **窓関数** で次に示すような矩形窓や画素に対して重み付けをするガウス窓が用いられる。

窓関数

窓関数とは、**ある有限区間以外で 0 となる関数** である。この関数を別の関数や信号(データ)にかけ合わせると、区間外の数値は **全て 0** になり有限区間内だけが残るので数値解析が容易になる。

フーリエ変換との関係

フーリエ変換の定義によると、「あらゆる周期関数は正弦波と余弦波の重ね合わせで表すことができる」。要するに、周期関数とはあるパターンが一定周期で繰り返される関数のことであり、逆に言えば、**フーリエ変換は周期関数のみに対応している**といえる。

また、フーリエ変換では、関数 $f(x)$ も三角関数も、無限区間 $(-\infty, \infty)$ で定義されている。しかし、コンピュータ上で数値的にフーリエ変換を行うような場合は、無限区間を扱うことができないので、有限区間 $[a, b]$ でフーリエ変換を行い区間外は無視することになる。

これは、関数 $f(x)$ を区間外で 0 とみなすことに等しい。つまり 関数 $f(x)$ と関数

$$w(x) = \begin{cases} 1, if(a \leq x \leq) \\ 0, otherwise \end{cases}$$

の積 $(wf)(x) = w(x)f(x)$ を求め、そのフーリエ変換 $\mathcal{F}(wf)$ を $\mathcal{F}f$ の代わりに得ていることになる。このときかけ合わせた関数 $w(x)$ が窓関数である。矩形窓は、$x = a, b$ に著しい不連続があるため性能があまり良くない。そのため実際に使われる窓関数のほとんどは、両端が滑らかに小さくなり区間外の 0 につながる山形の関数である。

矩形窓

$$w(x) = \begin{cases} 1, if(a \leq x \leq) \\ 0, otherwise \end{cases}$$

ガウス窓

$$w(x, y) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

テイラー展開

関数 f のある点 a まわりテイラー展開は以下のように表される。

$$f(x) = f(a) + f'(a)(x-a) + \frac{1}{2!}f''(a)(x-a)^2 + \frac{1}{3!}f'''(a)(x-a)^3 + \cdots$$

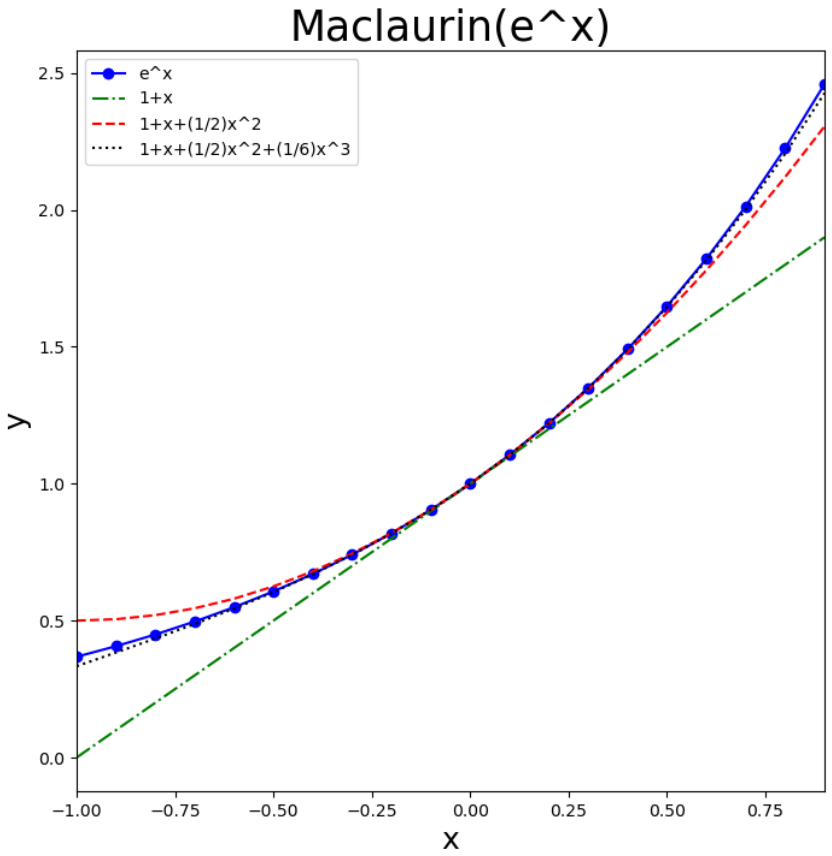
点 a におけるこの関数の値、この関数のグラフの傾きの値、2 回微分の値、3 回微分の値、$\cdots$ などの「溢れんばかりの情報」を使って、a からわずかに離れた x 地点での関数 $f(x)$ の値を言い当てることができるか。

$a = 0$ の場合におけるテイラー展開は、次式で表される。

$$f(x) = f(0) + F'(0)x + \frac{1}{2!}f''(0)x^2 + \frac{1}{3!}f'''(0)x^3 \cdots$$

このような「原点まわりでのテイラー展開」のことを「**マクローリン展開**」と呼ぶ。

例として、e^x をマクローリン展開において 3 回微分までで近似した場合の図を次に示す。



2020年10月23日

先ほどの図は次のプログラムによって作成された。

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(-1, 1, 0.1) # x=[-1.0, -0.9, ... 1.0]
y = np.exp(x)  # y = e^x
y_diff_1 = 1 + x
y_diff_2 = y_diff_1 + (1/2) * x**2
y_diff_3 = y_diff_2 + (1/6) * x**3

# Figureの初期化
fig = plt.figure(figsize=(8, 8))

plt.plot(x, y, color="b", marker="o", label="e^x")
plt.plot(x, y_diff_1, color="g", ls="-.", label="1+x")
plt.plot(x, y_diff_2, color="r", ls="--", label="1+x+(1/2)x^2")
plt.plot(x, y_diff_3, color="k", ls=":", label="1+x+(1/2)x^2+(1/6)x^3")

# 軸ラベルを設定
plt.xlabel("x", fontsize=18)
plt.ylabel("y", fontsize=18)
# X軸の最大値、最小値を指定
plt.xlim(x.min(), x.max())
# タイトルを表示
plt.title('Maclaurin(e^x)', fontsize=25)
# 凡例を表示
plt.legend()

plt.show()
```

画像の膨張 (Dilation)

- 縮小の逆処理。
- OpneCV では、カーネル内に '1' の値を持つ画素が 1 つでも存在すれば、出力画像の注目画素の画素値を '1' にする。
- 収縮した画像に対して膨張処理を行うことで、微小なノイズを消すことができる。

```
dilation = cv2.dilate(img, kernel, iterations = 1)
```