HW1 Monte Carlo Method

March 12, 2025

1 HW1 Question 4-6

The probability density func for the cauchy distribution is:

$$f(x;\theta) \;=\; \frac{1}{\pi\,[1+(x-\theta)^2]}.$$

The relative log-likelihood is:

$$\ell(\theta) = \sum_{i=1}^{n} \log f(x_i;\theta) = -\,n\log\pi \;-\; \sum_{i=1}^{n} \log[\,1+(x_i-\theta)^2].$$

As $-n\log\pi$ is a constant irrelative to θ then max $\ell(\theta)$ is same as minimizing the below:

$$\sum_{i=1}^{n} \log[\,1+(x_i-\theta)^2].$$

$$\text{We let}\quad S(\theta) \;=\; \frac{d}{d\theta}\,\ell(\theta) \;=\; -\sum_{i=1}^{n} \frac{2\,(x_i-\theta)}{1\;+\;(x_i-\theta)^2}.$$

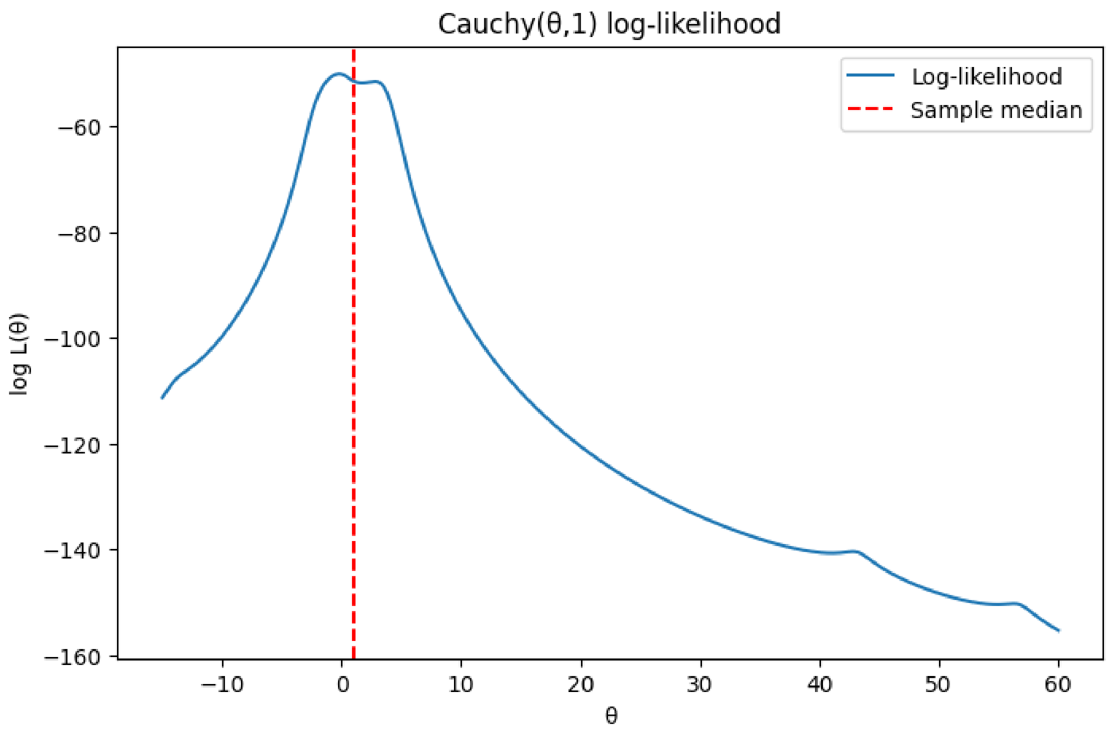
$$\text{To find MLE we need to figure out } S(\theta) \;=\; 0 \;\iff\; \sum_{i=1}^{n} \frac{x_i-\theta}{1+(x_i-\theta)^2} \;=\; 0.$$

- Question 4(a):

[21]:
```
import numpy as np

data = np.array([
    1.77, -0.23, 2.76, 3.80, 3.47, 56.75, -1.34, 4.24, -2.44, 3.29,
    3.71, -2.40, 4.53, -0.07, -1.05, -13.87, -2.53, -1.75, 0.27, 43.21
])
n = len(data)
data
```

[21]:
```
array([  1.77,  -0.23,   2.76,   3.8 ,   3.47,  56.75,  -1.34,   4.24,
        -2.44,   3.29,   3.71,  -2.4 ,   4.53,  -0.07,  -1.05, -13.87,
        -2.53,  -1.75,   0.27,  43.21])
```

[27]:
```
import matplotlib.pyplot as plt
import numpy as np

# log likelihood func
def log_likelihood(theta, x):
    #    - n*log(pi)
    return - np.sum(np.log(1.0 + (x - theta)**2))

thetas = np.linspace(-15, 60, 300)
ll_values = [log_likelihood(th, data) for th in thetas]

plt.figure(figsize=(8, 5))
plt.plot(thetas, ll_values, label="Log-likelihood")
plt.axvline(np.median(data), color='r', ls='--', label="Sample median")
plt.title("Cauchy( ,1) log-likelihood")
plt.xlabel(" ")
plt.ylabel("log L( )")
plt.legend()
plt.show()
```



We let $$S(\theta) = \sum_{i=1}^{n} \frac{(x_i - \theta)}{1 + (x_i - \theta)^2}.$$

We only need to find the zero point of S(θ)

The Newton's method is:

$$\theta_{\text{new}} \;=\; \theta_{\text{old}} \;-\; \frac{S(\theta_{\text{old}})}{S'(\theta_{\text{old}})},$$

where

$$S'(\theta) \;=\; \sum_{i=1}^{n} \frac{d}{d\theta}\left(\frac{x_i-\theta}{1+(x_i-\theta)^2}\right) = \sum_{i=1}^{n} \frac{-(1+(x_i-\theta)^2) \;+\; 2\,(x_i-\theta)^2}{\left[1+(x_i-\theta)^2\right]^2}.$$

For which can be written as:

$$\sum_{i=1}^{n} \frac{-1+(x_i-\theta)^2}{\left[1+(x_i-\theta)^2\right]^2}.$$

```
[52]: def S(theta, x):
          """ i.e. sum( (x_i - theta)/(1 + (x_i - theta)^2 ) ). """
          return np.sum((x - theta) / (1.0 + (x - theta)**2))

      def S_prime(theta, x):
          """ S(theta) derivative """
          num = -1.0 + (x - theta)**2
          den = (1.0 + (x - theta)**2)**2
          return np.sum(num / den)

      def newton_method(theta0, x, max_iter=100, tol=1e-8):
          theta = theta0
          for i in range(max_iter):
              f = S(theta, x)
              fp = S_prime(theta, x)
              if abs(fp) < 1e-21:
                  print(f"             ={i}")
                  break
              theta_new = theta - f/fp
              if abs(theta_new - theta) < tol:
                  break
              theta = theta_new
          return theta

      # testing for all starting points:
      start_points = [-11, -1, 0, 1.5, 8, 38]
      for sp in start_points:
          est = newton_method(sp, data)
          print(f"Newton start at {sp:5}, converged to {est:.4f}")
```

```
             =35
Newton start at   -11, converged to -215268457525.0355
Newton start at    -1, converged to -0.1923
Newton start at     0, converged to -0.1923
Newton start at   1.5, converged to 1.7136
             =35
Newton start at     8, converged to -150101717444.7210
Newton start at    38, converged to 42.7954
```

From the output we can see that Newton's method can be highly sensitive to initial guesses for this Cauchy-likelihood problem. In particular: * Starting at θ_0 = -11 or 8 causes the algorithm to step into a region where the derivative is very small (hence the warning message) and ultimately leads to an extreme value for the estimate. This suggests instability or overshooting in the iterations—typical of Newton's method when the Hessian (or first derivative) becomes very close to zero or the step size is not controlled. * For initial values nearer the (apparent) global maximum (e.g., θ_0 = -1, 0, 1.5), the method converges to more "reasonable" solutions around -0.19 or 1.71. These are presumably local maxima or at least local stationary points close to where the sample median would also suggest an optimum. Given that the dataset contains large outliers (e.g., 56.75 and 43.21), the likelihood can have somewhat complicated curvature in certain regions, which can lead Newton's method astray if started too far away. * The outcome of $\theta \approx -1.51$ or $\theta \approx 42.80$ from some starts (like 8 or 38) further illustrates the presence of potential multiple extrema or at least multiple stationary points in the likelihood function (especially with heavy-tailed Cauchy data that contains severe outliers).

Overall, these results highlight the fact that while Newton's method converges quickly under ideal conditions, it may fail to converge or converge to less meaningful local solutions if the initial guess is not chosen carefully or if there are outliers causing nearly flat regions in the objective. A more robust approach—like using a line search, bisection, or secant method, or simply starting near the sample median—often avoids these instabilities.

- Question 4(b):

```
[60]: def bisection_method(a, b, x, max_iter=50, tol=1e-6):
          fa = S(a, x)
          fb = S(b, x)
          if fa * fb > 0:
              raise ValueError("            ")

          for i in range(max_iter):
              m = 0.5*(a + b)
              fm = S(m, x)
              if abs(fm) < tol:
                  break
              #
              if fa*fm < 0:
                  b = m
                  fb = fm
              else:
                  a = m
                  fa = fm
          return 0.5*(a + b)

      a0, b0 = -1, 1
```

```
try:
    est_bisect = bisection_method(a0, b0, data)
    print(f"Bisection in [{a0},{b0}] --> {est_bisect:.4f}")
except ValueError as e:
    print("   ", e)

a0, b0 = -2, -1
try:
    est_bisect = bisection_method(a0, b0, data)
    print(f"Bisection in [{a0},{b0}] --> {est_bisect:.4f}")
except ValueError as e:
    print("   ", e)
```

```
Bisection in [-1,1] --> -0.1923
```

- Question 4(c): Secant Method

$$\theta_{k+1} = \theta_k \; - \; S(\theta_k)\,\frac{\theta_k - \theta_{k-1}}{S(\theta_k) - S(\theta_{k-1})}.$$

```
[65]: def secant_method(theta0, theta1, x, max_iter=100, tol=1e-6):
          f0 = S(theta0, x)
          f1 = S(theta1, x)
          for i in range(max_iter):
              if abs(f1 - f0) < 1e-14:
                  print("          ")
                  return theta1
              # update theta
              theta2 = theta1 - f1 * (theta1 - theta0)/(f1 - f0)

              if abs(theta2 - theta1) < tol:
                  return theta2

              # next round
              theta0, theta1 = theta1, theta2
              f0, f1 = f1, S(theta2, x)

          return theta1

      # test1: ( ^(0),  ^(1)) = (-2, -1)
      theta_hat_1 = secant_method(-2, -1, data)
      print(f"Secant from (-2, -1) --> {theta_hat_1:.4f}")

      # test2: ( ^(0),  ^(1)) = (-3, 3)
      theta_hat_2 = secant_method(-3, 3, data)
      print(f"Secant from (-3,  3) --> {theta_hat_2:.4f}")
```

```
Secant from (-2, -1) --> -0.1923
Secant from (-3,  3) --> 2.8175
```

Because the Cauchy likelihood with outliers can have multiple stationary points, the secant method (which does not require bracketing) may converge to any one of them. Starting from $\theta^{(0)} = -3$ and $\theta^{(1)} = 3$, it happens to settle at $\theta \approx 2.8175$. This point is simply another stationary solution of $S(\theta)$, rather than the global maximum or the zero we might expect from a better-chosen initial pair.

- Question 5(a)

$$f(x;\theta) \;=\; \frac{1-\cos(x-\theta)}{2\pi}, \quad x \in [0, 2\pi],\; \theta \in [-\pi, \pi].$$

Because $1-\cos(x-\theta)$ is largest around $x \approx \theta+\pi$, the distribution is effectively "centered" $at \theta+\pi$. In fact, a short integral check shows that its mean is $\mathbb{E}[X] = \theta + \pi$.

- Method of Moments (MoM): Matching the sample mean $\bar{X}$ to the theoretical mean $\theta + \pi$ gives

$\hat{\theta}_{\text{MoM}} \;=\; \bar{X} \;-\; \pi,$

- Log-likelihood:

$$\ell(\theta) \;=\; \sum_{i=1}^{n} \log[f(x_i;\theta)] \;=\; \sum_{i=1}^{n} \Big\{ \log[1-\cos(x_i-\theta)] \;-\; \log(2\pi) \Big\}.$$

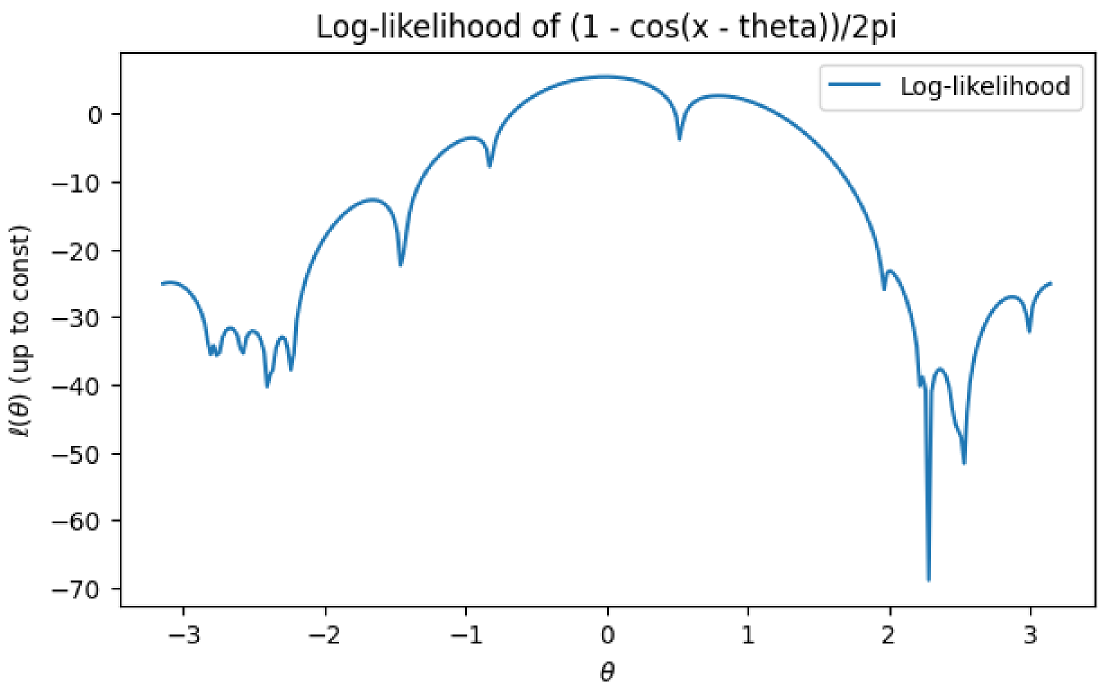
Ignoring the constant $-n\log(2\pi)$, the derivative is

$$\frac{d}{d\theta}\,\ell(\theta) \;=\; -\sum_{i=1}^{n} \frac{\sin(x_i-\theta)}{1-\cos(x_i-\theta)}.$$

Setting this to zero and solving numerically yields the MLE.

[79]:
```python
import numpy as np
import matplotlib.pyplot as plt

# -----------------------------
# (a) Plotting the log-likelihood
# -----------------------------
data = np.array([
    3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28, 1.96,
    2.53, 3.88, 2.22, 3.47, 4.82, 2.46, 2.99, 2.54, 0.52, 2.50
])
n = len(data)

def loglike(theta, x):
    # ignore constant -n*log(2*pi)
    return np.sum( np.log(1.0 - np.cos(x - theta)) )

thetas = np.linspace(-np.pi, np.pi, 300)
```

```
ll_vals = [loglike(th, data) for th in thetas]

plt.figure(figsize=(7,4))
plt.plot(thetas, ll_vals, label='Log-likelihood')
plt.xlabel(r'$\theta$')
plt.ylabel(r'$\ell(\theta)$ (up to const)')
plt.title("Log-likelihood of (1 - cos(x - theta))/2pi")
plt.legend()
plt.show()
```



$$E[X] \;=\; \int_0^{2\pi} x\, f(x;\theta)\, dx \;=\; \int_0^{2\pi} x\, \frac{1-\cos(x-\theta)}{2\pi}\, dx.$$

$z = x - \theta$ $x = z + \theta$ $\quad x \in [0, 2\pi]$ $\quad z \in [-\theta,\, 2\pi - \theta]$

$$E[X] \;=\; \frac{1}{2\pi} \int_{z=-\theta}^{2\pi-\theta} (z+\theta)\,[1-\cos z]\, dz.$$

$$E[X] \;=\; \pi \;+\; \sin(\theta).$$

- Question 5(b)

[89]:
```
import numpy as np

mean_data = np.mean(data)
```

```
val = mean_data - np.pi    # sin(\theta)

if val < -1 or val > 1:
    print("No real solution for MoM: the sample mean is out of [ -1,  +1].")
    theta_mom = None
else:
    theta_mom = np.arcsin(val)
    #       [- ,  ]
    if theta_mom < -np.pi:
        theta_mom += 2*np.pi
    elif theta_mom > np.pi:
        theta_mom -= 2*np.pi

print("Method-of-Moments estimate:", theta_mom)
```

```
Method-of-Moments estimate: 0.05844060614042408
```

- Question 5(c)

[91]:
```
def S(theta, x):
    """Derivative of log-likelihood w.r.t. theta (set =0 for MLE)."""
    return -np.sum( np.sin(x - theta) / (1.0 - np.cos(x - theta)) )

def S_prime(theta, x):
    """2nd derivative of log-likelihood w.r.t. theta (for Newton)."""
    # derivative of - sin(z)/(1 - cos(z)) w.r.t. z => complicated expression
    # We'll do it directly with finite differences or symbolic if we like.
    # For clarity, let's do a straightforward numeric difference approach:
    eps = 1e-8
    return (S(theta+eps, x) - S(theta-eps, x)) / (2*eps)

def newton_method(theta0, x, max_iter=100, tol=1e-7):
    theta = theta0
    for i in range(max_iter):
        f  = S(theta, x)
        fp = S_prime(theta, x)
        if abs(fp)<1e-12:
            break
        theta_new = theta - f/fp
        # keep in [-pi, pi] if we interpret the parameter mod 2pi
        if   theta_new>np.pi:  theta_new -= 2*np.pi
        elif theta_new<-np.pi: theta_new += 2*np.pi

        if abs(theta_new - theta) < tol:
            theta = theta_new
            break
        theta = theta_new
    return theta
```

[93]:
```
# use MoM result from (b) as starting value
theta_hat = newton_method(theta_mom, data)
print(f"MLE from Newton, starting at MoM: {theta_hat:.4f}")

# try the same Newton method from -2.7 and 2.7
theta_hatA = newton_method(-2.7, data)
theta_hatB = newton_method( 2.7, data)

print(f"Newton from -2.7 => {theta_hatA:.4f}")
print(f"Newton from  2.7 => {theta_hatB:.4f}")
```

```
MLE from Newton, starting at MoM: -0.0120
Newton from -2.7 => -2.6667
Newton from  2.7 => 2.8731
```

- Question 6

$$P(Y=1|X;\beta) \;=\; \frac{1}{1+\exp[-(\beta_0+\beta_1X_1+\beta_2X_2)]}.$$

[ ]:
```

```

[103]:
```
import pandas as pd
import statsmodels.api as sm

# 1.   txt
#        /Tab     sep='\s+'
df = pd.read_csv("/Users/dongwenou/Downloads/Statistical Computing/
 ↪HW_chapter1_6.txt", header=None, sep=r'\s+')
df.columns = ['Y', 'X1', 'X2']  #

# 2.
df['intercept'] = 1.0
X = df[['intercept', 'X1', 'X2']]
y = df['Y']

# 3.   statsmodels     (Logit)
logit_model = sm.Logit(y, X)
result = logit_model.fit()  # MLE
print(result.summary())

#    [beta0, beta1, beta2]
print("\nMLE of beta:")
print(result.params)
```

```
Optimization terminated successfully.
         Current function value: 0.445592
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                      Y   No. Observations:                  200
Model:                          Logit   Df Residuals:                      197
Method:                           MLE   Df Model:                            2
Date:                Wed, 12 Mar 2025   Pseudo R-squ.:                  0.3564
Time:                        10:58:07   Log-Likelihood:                -89.118
converged:                       True   LL-Null:                       -138.47
Covariance Type:            nonrobust   LLR p-value:                 3.691e-22
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.1944      0.189      1.031      0.302      -0.175       0.564
X1            -1.2942      0.244     -5.310      0.000      -1.772      -0.816
X2             3.5546      0.550      6.458      0.000       2.476       4.633
==============================================================================

MLE of beta:
intercept    0.194429
X1          -1.294162
X2           3.554608
dtype: float64
```

[131]:
```python
import numpy as np

def read_data_txt(filepath):
    """
      txt
       :  Y X1 X2
    """
    data = np.loadtxt(filepath, dtype=float)
    # data[:,0] => Y
    # data[:,1] => X1
    # data[:,2] => X2
    Y = data[:, 0]
    X1 = data[:, 1]
    X2 = data[:, 2]

    #      X: [1, X1, X2], shape=(n,3)
    n = len(Y)
    intercept = np.ones((n,1))
    X = np.column_stack((intercept, X1, X2))  # shape (n,3)
    return X, Y
```

```
def logistic_p(X, beta):
    """
        p_i = 1 / [1 + exp(-(X beta))]
    X: shape=(n,3)
    beta: shape=(3,) or (3,1)
      shape=(n,)
    """
    z = X.dot(beta)  # (n,)
    return 1.0 / (1.0 + np.exp(-z))

def log_likelihood(X, Y, beta):
    """
       : sum [ y_i*log(p_i) + (1-y_i)*log(1-p_i) ]
    X: (n,3)
    Y: (n,)
    beta: (3,)
    """
    p = logistic_p(X, beta)
    eps = 1e-12  #    log(0)
    ll = np.sum(Y*np.log(p+eps) + (1.0-Y)*np.log(1.0-p+eps))
    return ll

def gradient(X, Y, beta):
    """
      : X^T (Y - p)
      shape=(3,)
    """
    p = logistic_p(X, beta)   # shape=(n,)
    diff = Y - p              # shape=(n,)
    grad = X.T.dot(diff)      # shape=(3,)
    return grad

def hessian(X, Y, beta):
    """
      Hessian  : - X^T [p(1-p) I] X
      p(1-p)
      shape=(3,3)
    """
    p = logistic_p(X, beta)
    W = p * (1.0 - p)         # shape=(n,)
    # X^T diag(W) X
    #    : X^T(W * X )
    #   X    W => shape (n,3)
    Xw = X * W[:, np.newaxis]
    H = -(X.T.dot(Xw))        # Hessian = -[X^T diag(W) X]
    return H
```

```
def newton_raphson_logistic(X, Y, tol=1e-6, max_iter=100):
    """
     Newton-Raphson     MLE
       beta
    """
    n, d = X.shape  # n=  , d=3( +2 )
    beta = np.zeros(d)  #

    ll_history = []
    for i in range(max_iter):
        #    & Hessian
        g = gradient(X, Y, beta)
        H = hessian(X, Y, beta)

        #  : beta_new = beta - H^{-1} g
        #    invert Hessian(      )
        try:
            H_inv = np.linalg.inv(H)
        except np.linalg.LinAlgError:
            print("Hessian is singular - stopping.")
            break

        beta_new = beta - H_inv.dot(g)

        #
        ll_new = log_likelihood(X, Y, beta_new)
        ll_history.append(ll_new)

        #   :  beta
        if np.linalg.norm(beta_new - beta) < tol:
            beta = beta_new
            break
        beta = beta_new

    return beta, ll_history


if __name__ == "__main__":
    #   :
    filepath = "/Users/dongwenou/Downloads/Statistical Computing/HW_chapter1_6.
 ↪txt"

    # 1.
    X, Y = read_data_txt(filepath)  # X.shape=(n,3), Y.shape=(n,)

    # 2. Newton-Raphson
    beta_est, ll_hist = newton_raphson_logistic(X, Y, tol=1e-7, max_iter=200)
```

```python
# 3.
print("Estimated beta:", beta_est)
print("Final log-likelihood:", ll_hist[-1])
print("Number of iterations:", len(ll_hist))
```

```
Estimated beta: [ 0.19442939 -1.29416247  3.55460812]
Final log-likelihood: -89.11835572404411
Number of iterations: 7
```

```
[ ]:
```