Homework 5

In [1]:
```
import numpy as np
import time
```

1. Comparing Gradient Simulation Methods

Suppose $z(\theta) = E[Z(\theta)]$ where $Z(\theta) \sim \exp(\theta)$ for $\theta > 0$. Now, suppose we want to estimate $z'(\theta)$ at $\theta = 0.1$ via gradient simulation using 100 simulation repetitions.

1. Implement FD (with and without common random numbers (CRN)), IPA and LR methods for this gradient simulation task. Report the numeric value of each estimator. (Due to few samples, the results of the same order are acceptable.)

In [2]:
```
def FD(theta:float, n:int, h:float):
    samples = np.empty(n)
    for i in range(n):
        samples[i] = (np.random.exponential(1/(theta+h)) - np.random.exponential(1/(theta-h))) / (2*h)
    return samples

def FD_CRN(theta:float, n:int, h:float):
    samples = np.empty(n)
    for i in range(n):
        exp1 = np.random.exponential(1)
        samples[i] = (exp1/(theta+h) - exp1/(theta-h)) / (2*h)
    return samples

def IPA(theta:float, n:int, h=None):
    """
        exp(theta) = exp(1)/theta
        so d/dtheta exp(theta) = -exp(1)/theta^2
    """
    samples = np.empty(n)
    for i in range(n):
        samples[i] = -np.random.exponential(1) / theta**2
    return samples

def LR(theta:float, n:int, h=None):
    """
        S(theta, Z) = d/dtheta log theta exp(-theta Z) = 1/theta - Z
    """
    samples = np.empty(n)
    for i in range(n):
        Z = np.random.exponential(1/theta)
        S = 1/theta - Z
        samples[i] = Z * S
    return samples
```

In [3]:
```
def test1(func, theta, n, h=None):
    np.random.seed(1) # for reproducibility
    start_time = time.time()
    samples = func(theta, n, h)
    mean, std = samples.mean(), samples.std()
    print(f'{func.__name__}\tmean: {mean:8.3f}, std: {std:7.3f}, CI: ({mean - 1.96*std/np.sqrt(n):8.3f}, {mean +


theta = 0.1
n = 100
test1(FD, theta, n, 0.05)
test1(FD_CRN, theta, n, 0.05)
test1(IPA, theta, n)
test1(LR, theta, n)
```

```
FD      mean: -159.177, std: 217.612, CI: (-201.828, -116.525), time: 0.0042s
FD_CRN  mean: -126.432, std: 121.543, CI: (-150.254, -102.609), time: 0.00046s
IPA     mean:  -94.824, std:  91.157, CI: (-112.691,  -76.957), time: 0.00036s
LR      mean:  -78.188, std: 229.644, CI: (-123.199,  -33.178), time: 0.00039s
```

1. As the true value of $z'(0.1)$ is known, report and compare the mean square error of the three methods estimated using 1,000 simulation rounds (each round consists of 100 repetitions). For FD methods (with and without CRN), you can try different choices of parameter h and report the best one.

In [4]:
```python
def mse(func, theta, N, n, h=None):
    truth = - 1 / theta**2
    sample_mean = np.empty(N)
    for i in range(N):
        samples = func(theta, n, h)
        sample_mean[i] = samples.mean()
    return np.mean((sample_mean - truth)**2)

def test2(func, theta, N, n, h=None):
    np.random.seed(1) # for reproducibility
    start_time = time.time()
    MSE = mse(func, theta, N, n, h)
    print(f'{func.__name__}\tMSE: {MSE:.3f},\ttime: {time.time() - start_time:.3g}s')

N = 1000

theta = 0.1
n = 100
test2(FD, theta, N, n, 0.034) # 0.034 is the best h
test2(FD_CRN, theta, N, n, 1e-4) # almost 0 is the best h
test2(IPA, theta, N, n)
test2(LR, theta, N, n)
```

```
FD      MSE: 788.637,   time: 0.232s
FD_CRN  MSE: 97.743,    time: 0.108s
IPA     MSE: 97.743,    time: 0.104s
LR      MSE: 1198.156,  time: 0.104s
```

2. Combining IPA and LR

A digital call option has discounted payoff as

$$Y = e^{-rT}1_{S_T>K}, \quad S_T = S_0 e^{\left(r-\frac{\sigma^2}{2}\right)T+\sigma\sqrt{T}Z}, \quad Z \sim N(0,1).$$

The payoff is not continuous in S_T (and thus in S_0), so IPA is not directly applicable.

(a) Implement LR method to estimate $\partial_{S_0} E[Y]$. Set $S_0 = K = 100, T = 0.25, \sigma = 0.3, r = 0.05$ and run your algorithm for 10,000 simulation rounds to estimate the variance of LR gradient estimator (of single repetition).

Note that

$$S_T = e^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}Z+\log S_0} = e^{(r-\frac{\sigma^2}{2})T+\sigma\sqrt{T}Z'}, \quad Z' \sim N\left(\frac{\log S_0}{\sigma\sqrt{T}}, 1\right)$$

(you can also use normal random variable with variance not 1) and then the score function is

$$S(z', S_0) = \frac{\partial}{\partial S_0}\log f_{Z'}(z'; S_0) = -\frac{\partial}{\partial S_0}(z' - \frac{\log S_0}{\sigma\sqrt{T}})^2/2 = (z' - \frac{\log S_0}{\sigma\sqrt{T}})/(S_0\sigma\sqrt{T})$$

In [5]:
```python
# parameter
K = 100
T = 0.25
sigma = 0.3
r = 0.05


def LR(S0, n):
    samples = np.empty(n)
    for i in range(n):
        norm_mean = np.log(S0) / (sigma * np.sqrt(T))
        Zprime = np.random.randn() + norm_mean
        score = (Zprime - norm_mean) / (S0 * sigma * np.sqrt(T))
        ST = np.exp((r - sigma**2/2) * T + sigma * np.sqrt(T) * Zprime)
        Y = np.exp(-r * T) * (ST > K)
        samples[i] = Y * score
    return samples

S0 = 100
n = 10000
np.random.seed(1)
samples = LR(S0, n)
print(f'\033[31mLR\tmean: {samples.mean():.3g},\tvariance: {samples.var():.3e}\033[0m')
```

```
LR      mean: 0.0265,   variance: 1.506e-03
```

We can decompose the indicator function as

$$1_{x>K} = f_\varepsilon(x) + h_\varepsilon(x)$$

with

$$f_\varepsilon(x) = \min\left\{1, \frac{\max\{0, x-K+\varepsilon\}}{2\varepsilon}\right\}, \quad h_\varepsilon = 1_{x>K} - f_\varepsilon(x)$$

As a consequence,

$$E[Y] = e^{-rT}E\left[f_\varepsilon\left(S_T\right)\right] + e^{-rT}E\left[h_\varepsilon\left(S_T\right)\right]$$

The function $f_\varepsilon(x)$ is piecewise linear so that we can apply IPA to $f_\varepsilon\left(S_T\right)$ and apply LR to $h_\varepsilon\left(S_T\right)$. In the class we mentioned that variance of the estimator obtained by this combination method is much smaller than the variance directly applying LR to Y.

(b) Verify that $f_\varepsilon(x)$ is continuous and piecewise linear in x.

Ans: Note that for any $\varepsilon > 0$,

$$f_\varepsilon(x) = \begin{cases} 1, & x \geq K+\varepsilon \\ \frac{x-K+\varepsilon}{2\varepsilon}, & K-\varepsilon < x < K+\varepsilon \\ 0, & x \leq K-\varepsilon \end{cases}$$

So $f_\varepsilon(x)$ is continuous and piecewise linear in x.

(c) Implement IPA and LR to estimate $\partial_{S_0}E\left[f_\varepsilon\left(S_T\right)\right]$ and $\partial_{S_0}E\left[h_\varepsilon\left(S_T\right)\right]$ respectively. Thus, we obtain a combination method to estimate $\partial_{S_0}[Y]$.

In [6]:
```python
def IPA_f(S0, n, eps):
    samples = np.empty(n)
    for i in range(n):
        Z = np.random.randn()
        ST = S0 * np.exp((r - sigma**2/2) * T + sigma * np.sqrt(T) * Z)
        ST_derivative = np.exp((r - sigma**2/2) * T + sigma * np.sqrt(T) * Z)
        f_derivative = (ST > K - eps) * (ST < K + eps) / (2 * eps)
        samples[i] = ST_derivative * f_derivative
    return samples

def LR_h(S0, n, eps):
    samples = np.empty(n)
    for i in range(n):
        norm_mean = np.log(S0) / (sigma * np.sqrt(T))
        Zprime = np.random.randn() + norm_mean
        score = (Zprime - norm_mean) / (S0 * sigma * np.sqrt(T))
        ST = np.exp((r - sigma**2/2) * T + sigma * np.sqrt(T) * Zprime)
        h = (ST > K) - min(1, max(0, (ST - K + eps) / (2 * eps)))
        samples[i] = h * score
    return samples
```

(d) Still consider the setting where $S_0 = K = 100, T = 0.25, \sigma = 0.3, r = 0.05$. Run the combination method with $\varepsilon = 30$ for 10,000 rounds to estimate the variance of the gradient estimator (of single repetition). Compare it to the variance of LR obtained in part (a).

In [7]:
```python
# parameter
K = 100
T = 0.25
sigma = 0.3
r = 0.05
eps = 30

S0 = 100
n = 10000
np.random.seed(1)
samples = (IPA_f(S0, n, eps) + LR_h(S0, n, eps)) * np.exp(-r * T)
print(f'\033[31mCombination\tmean: {samples.mean():.3g},\tvariance: {samples.var():.3e}\033[0m')
```

```
Combination     mean: 0.0262,   variance: 4.757e-05
```