APMTH 207: Advanced Scientific Computing:

Stochastic Methods for Data Analysis, Inference and Optimization

Homework #4

Harvard University
Fall 2018
Instructors: Rahul Dave
Due Date: Saturday, October 13th, 2018 at 11:59pm

Instructions:

- Upload your final answers in the form of a Jupyter notebook containing all work to Canvas.
- Structure your notebook and your work to maximize readability.

Collaborators

Michelle (Chia Chi) Ho, Jiejun Lu, Jiawen Tong

In [1]:
```
import numpy as np
import scipy.stats
import scipy.special

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from matplotlib import cm
import pandas as pd
%matplotlib inline

from scipy.stats import norm, multivariate_normal
from scipy.stats import bernoulli
import statsmodels.discrete.discrete_model as sm
import seaborn as sns
```

Question 1: We'll Always Have that Night Sampling in Monte Carlo

Coding required

Let X be a random variable with distribution described by the following pdf:

$$f_X(x) = \begin{cases} \frac{1}{12}(x-1), & 1 \le x \le 3 \\ -\frac{1}{12}(x-5), & 3 < x \le 5 \\ \frac{1}{6}(x-5), & 5 < x \le 7 \\ -\frac{1}{6}(x-9), & 7 < x \le 9 \\ 0, & otherwise \end{cases}$$

Let h be the following function of X:

$$h(X) = \frac{1}{3\sqrt{2\pi}} \exp\left\{-\frac{1}{18}(X-5)^2\right\}$$

Compute $\mathbb{E}[h(X)]$ via Monte Carlo simulation using the following sampling methods:

1.1. Inverse Transform Sampling

1.2. Rejection Sampling with a uniform proposal distribution (rejection sampling in a rectangular box with uniform probability of sampling any x)

Answer 1.1, 1.2

CDF of X, $F_X(x)$:

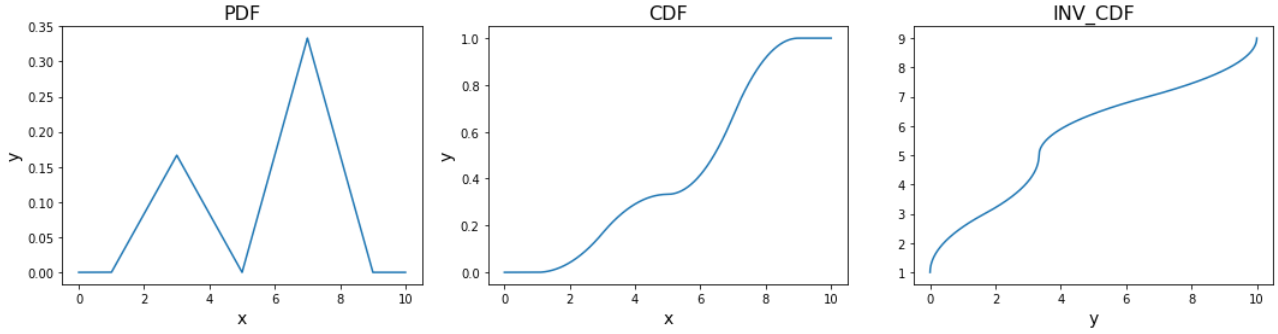
$$F_X(x) = \begin{cases} 0, & x < 1 \\ \frac{1}{24}x^2 - \frac{1}{12}x + \frac{1}{24}, & 1 \le x \le 3 \\ -\frac{1}{24}x^2 + \frac{5}{12}x - \frac{17}{24}, & 3 < x \le 5 \\ \frac{1}{12}x^2 - \frac{5}{6}x + \frac{29}{12}, & 5 < x \le 7 \\ -\frac{1}{12}x^2 + \frac{3}{2}x - \frac{23}{4}, & 7 < x \le 9 \\ 1, & x > 9 \end{cases}$$

Inverse CDF of X, $F_X^{-1}(y)$:

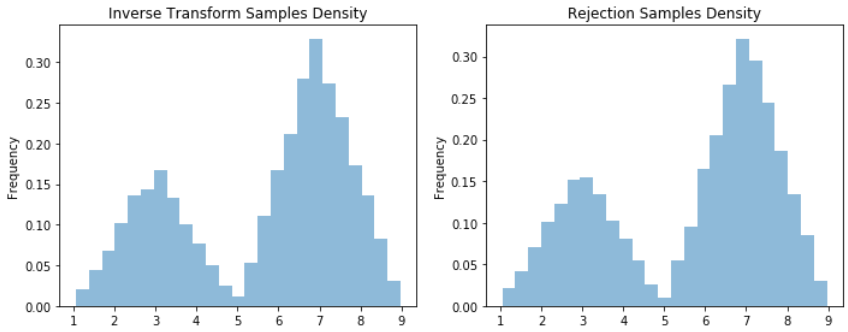
$$F_X^{-1}(y) = \begin{cases} \sqrt{24y} + 1, & 0 \le y \le \frac{1}{6} \\ 5 - \sqrt{8 - 24y}, & \frac{1}{6} < y \le \frac{1}{3} \\ 5 + \sqrt{12y - 4}, & \frac{1}{3} < y \le \frac{2}{3} \\ 9 - \sqrt{12 - 12y}, & \frac{2}{3} < y \le 1 \end{cases}$$

In [2]:

```python
# 1.1, 1.2
def PDF_X(x):
    if x < 1: return 0
    if x >= 1 and x <= 3: return 1/12*(x-1)
    if x > 3 and x <= 5: return -1/12*(x-5)
    if x > 5 and x <= 7: return 1/6*(x-5)
    if x > 7 and x <= 9: return -1/6*(x-9)
    return 0

def CDF_X(x):
    if x <= 1: return 0
    if x > 1 and x <= 3: return 1/24*x**2 - 1/12*x + 1/24
    if x > 3 and x <= 5: return -1/24*x**2 + 5/12*x - 17/24
    if x > 5 and x <= 7: return 1/12*x**2 - 5/6*x + 29/12
    if x > 7 and x <= 9: return -1/12*x**2 + 3/2*x - 23/4
    return 1

def INV_CDF_X(y):
    if y < 0 or y > 1: raise Exception('y must be within (0, 1).')
    if y <= 1/6: return np.sqrt(24*y) + 1
    if y > 1/6 and y <= 1/3: return 5 - np.sqrt(8 - 24*y)
    if y > 1/3 and y <= 2/3: return 5 + np.sqrt(12*y - 4)
    if y > 2/3 and y <= 1: return 9 - np.sqrt(12 - 12*y)
    return 1

x_min = 0
x_max = 10
N = 10000
x_lin = np.linspace(x_min, x_max, N)
PDF_x_lin = np.array([PDF_X(xi) for xi in x_lin])
CDF_x_lin = np.array([CDF_X(xi) for xi in x_lin])

y_lin = np.linspace(0, 1, N)
INV_CDF_lin = np.array([INV_CDF_X(yi) for yi in y_lin])

fig, axes = plt.subplots(1, 3, figsize=(15, 4))
axes[0].plot(x_lin, PDF_x_lin)
axes[1].plot(x_lin, CDF_x_lin)
axes[2].plot(x_lin, INV_CDF_lin)

axes[0].set_title('PDF', fontsize=16)
axes[1].set_title('CDF', fontsize=16)
axes[2].set_title('INV_CDF', fontsize=16)

axes[0].set_xlabel('x', fontsize=14)
axes[0].set_ylabel('y', fontsize=14)
axes[1].set_xlabel('x', fontsize=14)
axes[1].set_ylabel('y', fontsize=14)
axes[2].set_xlabel('y', fontsize=14)
plt.tight_layout()
```



In [3]:
```
# Inverse Transform sampling
CDF_x_min = CDF_X(x_min)
CDF_x_max = CDF_X(x_max)

R_unif = np.random.uniform(CDF_x_min, CDF_x_max, N)
X_inv_samp = np.array([INV_CDF_X(ru) for ru in R_unif])

# Rejection Sampling
y_max = 0.5
accepted = 0 # the number of accepted samples
X_rej_samp = np.zeros(N)
count = 0 # the total count of proposals
while (accepted < N):
    x = np.random.uniform(x_min, x_max) # x ~ uniform[xmin, xmax)
    y = np.random.uniform(0, y_max) # y ~ uniform[0, ymax)
    if y < PDF_X(x): # rejection testing
        X_rej_samp[accepted] = x
        accepted += 1
    count +=1

fig, axes = plt.subplots(1, 2, figsize=(10, 4))
pd.Series(X_inv_samp).plot(kind='hist', bins=25, density=True, alpha=0.5, ax=axes[0],
                           title='Inverse Transform Samples Density')
pd.Series(X_rej_samp).plot(kind='hist', bins=25, density=True, alpha=0.5, ax=axes[1],
                           title='Rejection Samples Density')
plt.tight_layout()
```



In [4]:
```
def h_X(x):
    return 1/(3*np.sqrt(2)*np.pi) * np.exp(-1/18*(x-5)**2)

E_h_inv_samp = np.array([h_X(xi) for xi in X_inv_samp]).mean()
E_h_rej_samp = np.array([h_X(xi) for xi in X_rej_samp]).mean()

print('E[h(x)] - Inverse Transform Samples: ', E_h_inv_samp)
print('E[h(x)] - Rejection Samples: ', E_h_rej_samp)
```

```
E[h(x)] - Inverse Transform Samples:  0.058996755143681946
E[h(x)] - Rejection Samples:  0.058785306521202486
```

Question 2: The Consequences of O-ring Failure can be Painful and Deadly

Coding required

In 1986, the space shuttle Challenger exploded during take off, killing the seven astronauts aboard. It is believed that the explosion was caused by the failure of an O-ring (a rubber ring that seals parts of the solid fuel rockets together), and that the failure was caused by the cold weather at the time of launch (31F).

In the file chall.txt, you will find temperature (in Fahrenheit) and failure data from 23 shuttle launches, where 1 stands for O-ring failure and 0 no failure. We assume that the observed temperatures are fixed and that, at temperature t, an O-ring fails with probability $f(\theta_1 + \theta_2 t)$ conditionally on $\Theta = (\theta_1, \theta_2)$.

$f(\vec{z})$ is defined to be the logistic function -- $f(\vec{z}) = 1/(1 + \exp(\vec{-z}))$

2.1. Based on your own knowledge and experience, suggest a prior distribution for the regression parameters (θ_1, θ_2). Make sure to explain your choice of prior.

2.2. Produce 5000-10000 samples from the posterior distribution of Θ using rejection sampling, and plot them and their marginals. (This may take a while.)

2.3. Use the logit package in the `statsmodels` library to compute 68% confidence intervals on the θ parameters. Compare those intervals with the 68% credible intervals from the posterior above. Overlay these on the above marginals plots.

2.4. Use the MLE values from `statsmodels` and the posterior mean from **2.2** at each temperature to plot the probability of failure in the frequentist and bayesian settings as a function of temperature. What do you see?

2.5. Compute the mean posterior probability for an O-ring failure at $t = 31^{\circ}F$. To do this you must calculate the posterior at $31^{\circ}F$ and take the mean of the samples obtained.

2.6. You can instead obtain the probability from the posterior predictive. Use the posterior samples to obtain samples from the posterior predictive at $31^{\circ}F$ and calculate the fraction of failures.

2.7. The day before a new launch, meteorologists predict that the temperature will be $T \sim N(68, 1)$ during take-off. Estimate the probability for an O-ring failure during this take-off. (You will calculate multiple predictives at different temperatures for this purpose).

Answer 2.1

Prior Distribution:

$$\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix} \sim \mathcal{N}(\mu, \Sigma) \quad \text{where } \mu = \begin{bmatrix}15\\ -1\end{bmatrix} \quad \Sigma = \begin{bmatrix}5 & 0\\ 0 & 1\end{bmatrix}$$

Based on the `sns.regplot` below, we chose a weakly informative, $\mu(\theta_1) = 15$ and $\mu(\theta_2) = -1$ with reasonable variances.

Answer 2.2, 2.3

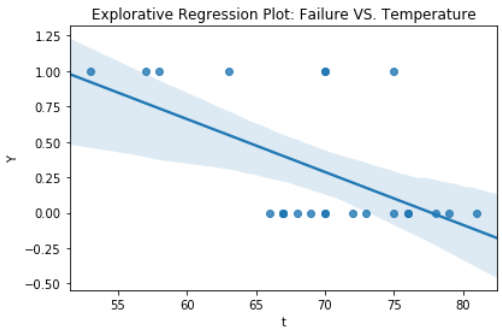
In [5]:
```
df_chall = pd.read_csv('chall.txt', sep='    ', header=None, names=['t', 'Y'])
df_chall.head()
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: ParserWarning: Falling back to the 'python' engine because the 'c' e
ngine does not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); you can avoid this
warning by specifying engine='python'.
  """Entry point for launching an IPython kernel.
```

Out[5]:

	t	Y
0	66	0
1	70	1
2	69	0
3	68	0
4	67	0

In [7]:
```
ax = sns.regplot(df_chall['t'], df_chall['Y'])
ax.set_title('Explorative Regression Plot: Failure VS. Temperature')
plt.tight_layout()
```



In [17]:
```
def sigmoid_theta(theta_1, theta_2, t):
    z = theta_1 + theta_2 * t
    return 1/(1+ np.exp(-z))

def prior_x_likelihood(theta_1, theta_2, X, Y, proposal='uniform'):
    if proposal == 'uniform':
        prior = (1/50) * (1)
    elif proposal == 'normal':
        prior = norm.pdf(theta_1, loc=15, scale=5) * norm.pdf(theta_2, loc=-1, scale=1)
    Z = theta_1 + theta_2 * X
    P = 1 / (1+ np.exp(-Z))
    likelihood = (P**Y * (1-P)**(1-Y)).prod()
    return prior * likelihood

def get_max_post(proposal):
    n_samp = 10
    theta_1_unif = np.linspace(-25, 45, n_samp)
    theta_2_unif = np.linspace(-4, 2, n_samp)
    M = 0
    for _i in range(n_samp):
        theta_1_i = theta_1_unif[_i]
        for _j in range(n_samp):
            v = prior_x_likelihood(theta_1_i, theta_2_unif[_j], df_chall['t'], df_chall['Y'], proposal)
            M = max(v, M)
    return M

def rej_sampling(X, Y, post_max, N, proposal='uniform'):
    accepted = 0
    count = 0
    intercept_samples, coef_samples = [],[]

    while (accepted < N):
        t1 = np.random.uniform(-25, 45)
        t2 = np.random.uniform(-4, 2)
        y = np.random.uniform(0, 1)
        if y < prior_x_likelihood(t1, t2, X, Y, proposal) / post_max:
            intercept_samples.append(t1)
            coef_samples.append(t2)
            accepted += 1
        count += 1
        if count % 100 == 0:
            print('accepted = {}, count = {}'.format(accepted, count), end='\r')

    return intercept_samples, coef_samples

```

In [18]:
```
M_norm = get_max_post('normal')
M_norm
```

Out[18]:
```
1.1968762975420554e-11
```

In [19]:
```
t1_samples_norm, t2_samples_norm = rej_sampling(df_chall['t'], df_chall['Y'], M_norm, 5000, proposal='normal')
```

```
accepted = 4998, count = 741000
```

In [20]:
```
# 2.3
ones_col = np.ones((df_chall.shape[0], 1))
t_ones = np.concatenate((ones_col, df_chall[['t']]), axis=1)
logit_results = sm.Logit(df_chall['Y'], t_ones).fit()
result_CI = logit_results.conf_int(alpha=0.32)

```

```
Optimization terminated successfully.
         Current function value: 0.441635
         Iterations 7
```

In [21]:
```
result_CI.rename(index={'const':'intercept', 'x1':'coef'}, columns={0: 'sm.logit.32%', 1: 'sm.logit.68%'},
                 inplace=True)
result_CI
```

Out[21]:

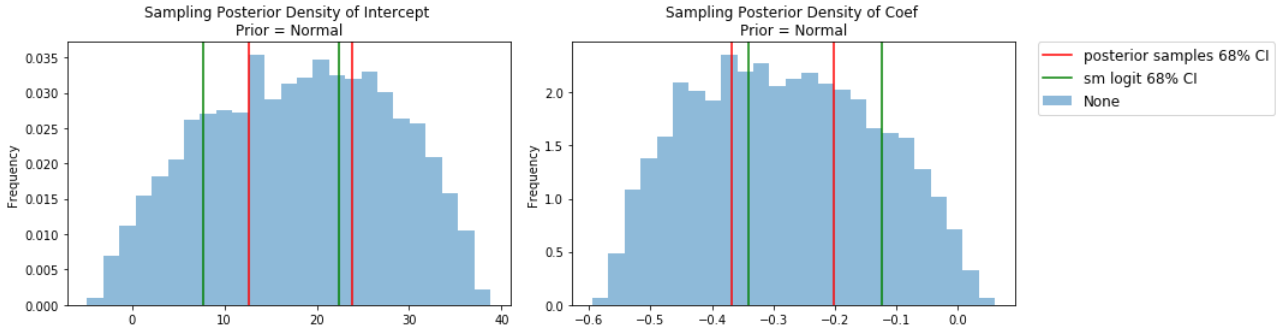
	sm.logit.32%	sm.logit.68%
intercept	7.705159	22.380645
coef	-0.339799	-0.124526

In [22]:
```
result_CI['norm.post.32%'] = [np.percentile(t1_samples_norm, 32), np.percentile(t2_samples_norm, 32)]
result_CI['norm.post.68%'] = [np.percentile(t1_samples_norm, 68), np.percentile(t2_samples_norm, 68)]
result_CI
```

Out[22]:

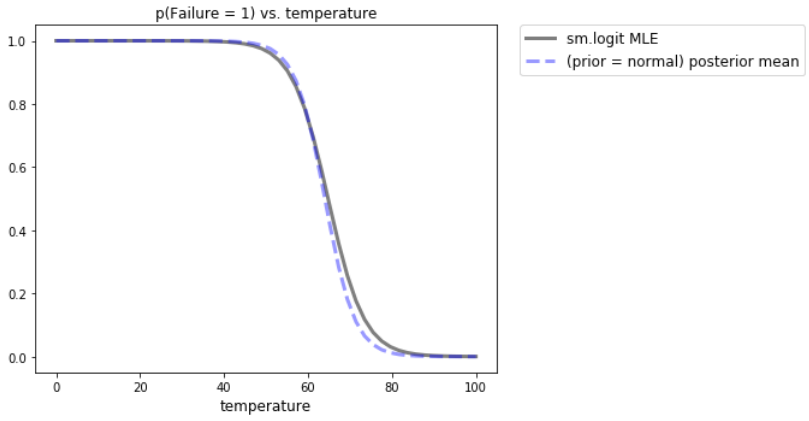
	sm.logit.32%	sm.logit.68%	norm.post.32%	norm.post.68%
intercept	7.705159	22.380645	12.677277	23.784446
coef	-0.339799	-0.124526	-0.366934	-0.200989

In [23]:
```
# === Normal as Prior/Proposal ===

# plot marginal distribution of Interncept & Coef samples
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
pd.Series(t1_samples_norm).plot(kind='hist', density=True, bins=25, alpha=0.5,
                                title='Sampling Posterior Density of Intercept \nPrior = Normal', ax=axes[0])
axes[0].axvline(result_CI['norm.post.32%'].loc['intercept'], c='r')
axes[0].axvline(result_CI['norm.post.68%'].loc['intercept'], c='r', label='posterior samples 68% CI')
axes[0].axvline(result_CI['sm.logit.32%'].loc['intercept'], c='g')
axes[0].axvline(result_CI['sm.logit.68%'].loc['intercept'], c='g', label='sm logit 68% CI')

pd.Series(t2_samples_norm).plot(kind='hist', density=True, bins=25, alpha=0.5,
                                title='Sampling Posterior Density of Coef \nPrior = Normal', ax=axes[1])
axes[1].axvline(result_CI['norm.post.32%'].loc['coef'], c='r')
axes[1].axvline(result_CI['norm.post.68%'].loc['coef'], c='r', label='posterior samples 68% CI')
axes[1].axvline(result_CI['sm.logit.32%'].loc['coef'], c='g')
axes[1].axvline(result_CI['sm.logit.68%'].loc['coef'], c='g', label='sm logit 68% CI')

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., fontsize=12)
plt.tight_layout()
```



Answer 2.4

In [25]:
```
# 2.4
temp_lin = np.linspace(0, 100)
t1_mle, t2_mle = logit_results.params
t1_p_norm, t2_p_norm = np.mean(t1_samples_norm), np.mean(t2_samples_norm)

fig, ax = plt.subplots(1, 1, figsize=(6, 5))
ax.plot(temp_lin, sigmoid_theta(t1_mle, t2_mle, temp_lin), 'k', linewidth=3, alpha=0.5,
        label='sm.logit MLE')
ax.plot(temp_lin, sigmoid_theta(t1_p_norm, t2_p_norm, temp_lin), 'b--',linewidth=3, alpha=0.4,
        label='(prior = normal) posterior mean')

ax.set_title('p(Failure = 1) vs. temperature')
ax.set_xlabel('temperature', fontsize=12)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., fontsize=12)
plt.tight_layout()
```



2.4 Observation: With uniform priors on both θ's, the probability of failure at different temperatures using MLE and posterior mean parameters are the same.

Answer 2.5

In [26]:
```
# 2.5
mean_post_p_norm = sigmoid_theta(np.array(t1_samples_norm), np.array(t2_samples_norm), 31).mean()

mean_post_p_norm
```

Out[26]:
```
0.9457789221168714
```

The mean posterior probability of failure = 1 at $t = 31^{\circ}F$ is 0.946.

Answer 2.6

In [27]:
```
pp_norm = sigmoid_theta(np.array(t1_samples_norm), np.array(t2_samples_norm), 31)
pp_mean_norm = bernoulli.rvs(pp_norm).mean()

pp_mean_norm
```

Out[27]: 0.9446

The fraction of failure computed from posterior predictive samples at $t = 31^{\circ}F$ is 0.945.

Answer 2.7

In [28]:
```
ts_normal_68_1 = np.random.normal(68, 1, 100)
pp_means_unif = []
pp_means_norm = []
for t in ts_normal_68_1:
#     pp_t_unif = sigmoid_theta(np.array(t1_samples_unif), np.array(t2_samples_unif), t)
#     pp_means_unif.append(bernoulli.rvs(pp_t_unif).mean())

    pp_t_norm = sigmoid_theta(np.array(t1_samples_norm), np.array(t2_samples_norm), t)
    pp_means_norm.append(bernoulli.rvs(pp_t_norm).mean())

print('Mean of the posterior predictive means at different temperature ~ N(68, 1):', np.mean(pp_means_norm))
```

```
Mean of the posterior predictive means at different temperature ~ N(68, 1): 0.32616600000000007
```

Question 3: Maximum Uniformity -- Frequentist Bootstraps and the Bayesian Posterior

Coding required

Recall in HW 3 Question 1 we attempted to explore an edge case in using non-parametric bootstrap to construct confidence intervals. Let's revisit the setup of that problem. Suppose you have $\{X_1, X_2, \ldots X_n\}$ datapoints such that X_i are independently and identically drawn from a $Unif(0, \theta)$. Consider the extreme order statistic Y = $X_{(n)}$ = max($X_1, X_2, \ldots X_n$).

3.1. Derive (or possibly re-write from HW3) expressions for $F_Y(y \mid n, \theta)$ the CDF of Y and $f_Y(y \mid n, \theta)$ the pdf of Y.

3.2. In HW3 we had difficulty constructing confidence intervals to estimate θ with percentiles as normal so instead we introduced pivot confidence intervals. Let's reframe the problem so that we can use percentiles to construct our confidence intervals. Define $Z \equiv n \cdot (\theta - Y)$ use elementary calculation to write an expression for $F_Z(z \mid n, \theta)$ the CDF of Z and $f_Z(y \mid n, \theta)$ the pdf of Z.

3.3. What is the limiting distribution of Z (as $n \to \infty$)? Plot that limiting distribution.

3.4. Use scipy/numpy to generate 100000 samples $\{X_i\}$ from Unif(0,100) (i.e. let θ = 100). Store them in Based on your data sample, what's $\hat{\theta}$ the empirical estimate for θ.

3.5. Use non-parametric bootstrap to generate a sampling distribution of 10000 estimates for Z by substituting $\hat{\theta}$ for θ. Plot a histogram of your sampling distribution. Make sure to title and label the plot. Use percentiles to construct the 10% and 68% bootstrap confidence intervals. Plot them in your graph.

Hint: Should the confidence intervals be symmetric around the estimate $\hat{\theta}$?

3.6. Make an argument that we can construct a bootstrap confidence interval that always mismatches the limiting distribution.

3.8. Let's switch to being Bayesian. In 3.1 we came up with an expression for the likelihood $f_Y(y \mid n, \theta)$. Use the Pareto distribution (https://en.wikipedia.org/wiki/Pareto_distribution) to construct a prior for θ. What are some reasonable values to use for the scale and shape?

3.9. Write down an expression for the posterior distribution $f_Y(\theta \mid n, y)$

3.10. Draw 10000 posterior samples and plot a histogram of the posterior distribution. Use percentiles to construct the 68% HPD. Plot the posterior distribution and mark the HPD on your plot.

3.11. How does th 68% HPD compare with the confidence interval generated from bootstrapping? Why doesn't the bayesian interval construction suffer the same concerns you noted in 3.6

Answer 3.1

The CDF of Y:

$$F_Y(y \mid n, \theta) = \int_{-\infty}^{y} f_Y(y \mid n, \theta) dy = P(Y \leq y) = P(\max(X_{1:n}) \leq y) = \begin{cases} 0 & y < 0 \\ (\frac{y}{\theta})^n & 0 \leq y \leq \theta \\ 1 & y > \theta \end{cases}$$

The PDF of Y:

$$\Rightarrow f_Y(y \mid n, \theta) = \frac{\partial F_Y(y \mid n, \theta)}{\partial y} = \begin{cases} 0 & y < 0 \\ \frac{n y^{n-1}}{\theta^n} & 0 \leq y \leq \theta \\ 0 & y > \theta \end{cases}$$

Answer 3.2

$Z \equiv n \cdot (\theta - Y) \Rightarrow Y = \theta - \frac{Z}{n}$

$$\begin{aligned} F_Z(z \mid n, \theta) = P(Z \le z) &= P(Y \ge \theta - \frac{z}{n}) \\ &= P(Y > \theta - \frac{z}{n}) \text{ (as } F_Y \text{ is continuous)} \\ &= 1 - P(Y \le \theta - \frac{z}{n}) \\ &= 1 - F_Y(y \mid n, \theta) \end{aligned}$$

The CDF of Z:

$$F_Z(z \mid n, \theta) = \begin{cases} 0 & z < 0 \\ 1 - (1 - \frac{z}{n\theta})^n & 0 \le z \le n\theta \\ 1 & z > n\theta \end{cases}$$

The PDF of Z:

$$f_Z(z \mid n, \theta) = \begin{cases} 0 & z < 0 \\ \frac{1}{\theta}(1 - \frac{z}{n\theta})^{n-1} & 0 \le z \le n\theta \\ 0 & z > n\theta \end{cases}$$

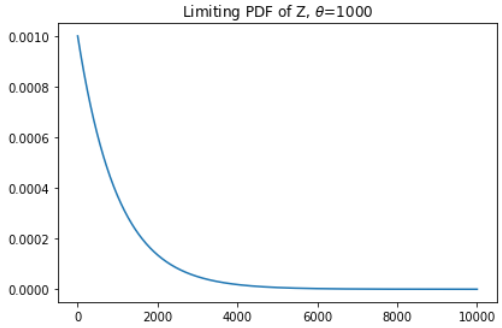
Answer 3.3

$$\lim_{n\to\infty} \left[\frac{1}{\theta}(1 - \frac{z}{n\theta})^{n-1}\right] = \lim_{n\to\infty} \left[\frac{1}{\theta}(1 - \frac{z}{n\theta})^{-1}(1 - \frac{z}{n\theta})^{n}\right] = \frac{1}{\theta}e^{-\frac{z}{\theta}}$$

$$\lim_{n\to\infty} \left[f_Z(z \mid n, \theta)\right] = \begin{cases} 0 & z < 0 \\ \frac{1}{\theta}e^{-\frac{z}{\theta}} & 0 \le z \le n\theta \end{cases}$$

As $n \to \infty, f_Z(z \mid n, \theta) \to$ an exponential distribution with mean θ.

In [29]:
```
def PDF_Z_lim(z, theta):
    if z<0: return 0
    return 1/theta * np.exp(-z/theta)

theta_q3 = 1000
z_lin = np.linspace(0, 10000, 1000)
PDF_z_lin = np.array([PDF_Z_lim(zi, theta_q3) for zi in z_lin])

fig, ax = plt.subplots(1, 1, figsize=(6, 4))
ax.plot(z_lin, PDF_z_lin)
ax.set_title(r'Limiting PDF of Z, $\theta$={}'.format(theta_q3))
plt.tight_layout()
```



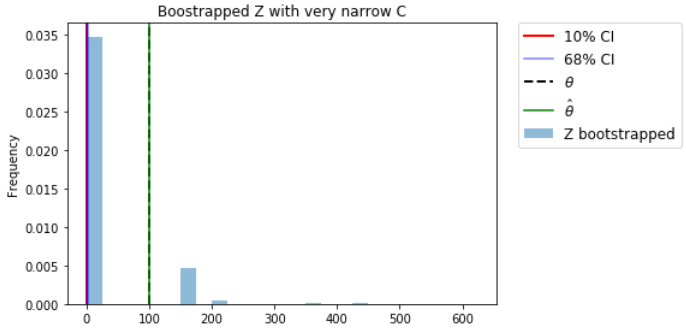
Answer 3.4, 3.5

In [30]:
```
# 3.4
X_q3 = np.random.uniform(low=0, high=100, size=100000)
theta_hat = X_q3.max()
print('Empirical estimate of theta_hat = max(Xi): ', theta_hat)
```

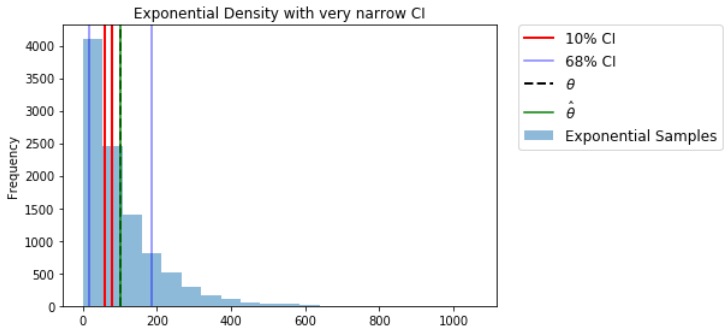
```
Empirical estimate of theta_hat = max(Xi):  99.99992210926924
```

In [31]:
```
# 3.5
X_q3_bootstrapped = np.random.choice(X_q3, size=(10000, len(X_q3)), replace=True)
Y_bootstrapped = X_q3_bootstrapped.max(axis=1)
Z_bootstrapped = 100000 * (theta_hat - Y_bootstrapped)
```

In [32]:
```
fig, ax = plt.subplots(1, 1, figsize=(6, 4))
pd.Series(Z_bootstrapped).plot(kind='hist', density=True, bins=25, alpha=0.5, ax=ax, label='Z bootstrapped')
ax.axvline(np.percentile(Z_bootstrapped, 45), linewidth=2, c='r')
ax.axvline(np.percentile(Z_bootstrapped, 55), linewidth=2, c='r', label='10% CI')
ax.axvline(np.percentile(Z_bootstrapped, 16), c='b', alpha=0.5)
ax.axvline(np.percentile(Z_bootstrapped, 84), c='b', alpha=0.5, label='68% CI')
ax.axvline(100, c='k', linestyle='--', linewidth=2, label=r'$\theta$')
ax.axvline(theta_hat, c='g', label=r'$\hat \theta$')
ax.set_title('Boostrapped Z with very narrow C')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., fontsize=12)
plt.tight_layout()
```



In [33]:
```
# 3.5
n_sample = 10000
exponential_sample = np.random.exponential(theta_hat, 10000)

fig, ax = plt.subplots(1, 1, figsize=(6, 4))
pd.Series(exponential_sample).plot(kind='hist', bins=20, alpha=0.5, label='Exponential Samples')

ax.axvline(np.percentile(exponential_sample, 45), linewidth=2, c='r')
ax.axvline(np.percentile(exponential_sample, 55), linewidth=2, c='r', label='10% CI')
ax.axvline(np.percentile(exponential_sample, 16), c='b', alpha=0.5)
ax.axvline(np.percentile(exponential_sample, 84), c='b', alpha=0.5, label='68% CI')

ax.axvline(100, c='k', linestyle='--', linewidth=2, label=r'$\theta$')
ax.axvline(theta_hat, c='g', label=r'$\hat \theta$')

ax.set_title('Exponential Density with very narrow CI')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., fontsize=12)
plt.tight_layout()
```



Answer 3.6

Since bootstrapped distribution is discrete and right skewed, while the confidence intervals are constructed by percentiles, the distribution mean (= true θ) is expected to be larger than the median. Therefore, a narrow enough confidence interval won't capture the mean.

Answer 3.8

$$\text{Pareto}(\alpha, \beta)$$

$$\text{shape } \alpha > 1 \quad \text{scale } \beta > 0$$

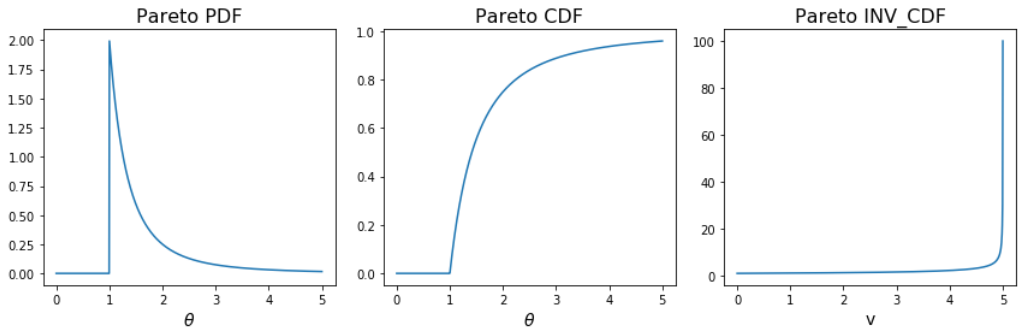
Answer 3.9

$$
\begin{aligned}
p(\theta|\, y, n) \propto p(\theta)(y \mid n, \theta) &= \frac{\alpha\beta^{\alpha}}{\theta^{\alpha+1}} \mathcal{I}_{\{\theta \geq \beta\}} \left(\frac{n y^{n-1}}{\theta^{n}} \right) \mathcal{I}_{\{\theta \geq y\}} \\
&= \frac{\alpha\beta^{\alpha}}{\theta^{\alpha+1}} \frac{n}{\theta^{n}} (y^{n-1}) \mathcal{I}_{\{\theta \geq max(y,\beta)\}} \\
&\propto \frac{\mathcal{I}_{\{\theta \geq max(y,\beta)\}}}{\theta^{\alpha+1+n}} \\
&\propto \text{Pareto}(n+\alpha, \max(\beta, Y))
\end{aligned}
$$

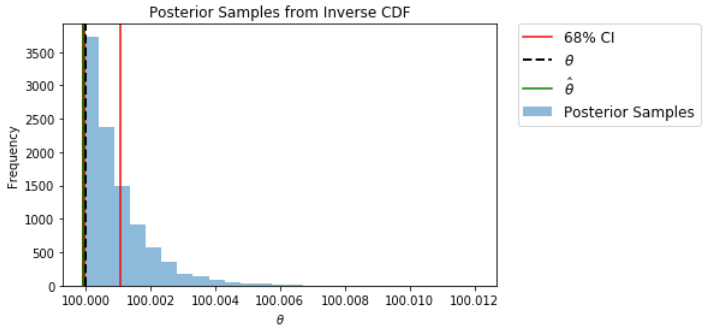
Answer 3.10

In [34]:

```
# 3.10
def PDF_Pareto(theta, alpha, beta):
    if theta < beta: return 0
    return alpha * (beta/theta) ** alpha / theta

def CDF_Pareto(theta, alpha, beta):
    if theta < beta: return 0
    return 1 - (beta/theta)**alpha

def INV_CDF_Pareto(v, alpha, beta):
    return beta / ((1-v)**(1/alpha))

alpha = 2
beta = 1

theta_min = 0
theta_max = 5
N_samples = 1000
theta_lin = np.linspace(theta_min, theta_max, N_samples)
PDF_Pareto_lin = np.array([PDF_Pareto(ti, alpha, beta) for ti in theta_lin])
CDF_Pareto_lin = np.array([CDF_Pareto(ti, alpha, beta) for ti in theta_lin])

v_lin = np.linspace(1e-4, 1-1e-4, N_samples)
INV_CDF_Pareto_lin = np.array([INV_CDF_Pareto(vi, alpha, beta) for vi in v_lin])

fig, axes = plt.subplots(1, 3, figsize=(12, 4))
axes[0].plot(theta_lin, PDF_Pareto_lin)
axes[1].plot(theta_lin, CDF_Pareto_lin)
axes[2].plot(theta_lin, INV_CDF_Pareto_lin)

axes[0].set_title('Pareto PDF', fontsize=16)
axes[1].set_title('Pareto CDF', fontsize=16)
axes[2].set_title('Pareto INV_CDF', fontsize=16)

axes[0].set_xlabel(r'$\theta$', fontsize=14)
axes[1].set_xlabel(r'$\theta$', fontsize=14)
axes[2].set_xlabel(r'v', fontsize=14)
plt.tight_layout()
```



In [37]:
```
# Inverse Transform sampling
N_samples = 10000

alpha_p = alpha + len(X_q3)
beta_p = np.max([beta, theta_hat])

V_unif = np.random.uniform(0, 1, N_samples)
# V_unif = np.random.uniform(CDF_Pareto(theta_hat, alpha_p, beta_p), CDF_Pareto(theta_hat+1, alpha_p, beta_p), N_samples)
theta_inv_samp = np.array([INV_CDF_Pareto(v, alpha_p, beta_p) for v in V_unif])

# plot sampling distribution
fig, ax = plt.subplots(1, 1, figsize=(6, 4))
pd.Series(theta_inv_samp).plot(kind='hist', bins=25, alpha=0.5, ax=ax, label='Posterior Samples')

ax.axvline(np.percentile(theta_inv_samp, 0), c='r', linewidth=3)
ax.axvline(np.percentile(theta_inv_samp, 68), c='r', label='68% CI')
ax.axvline(100, c='k', linestyle='--', linewidth=2, label=r'$\theta$')
ax.axvline(theta_hat, c='g', label=r'$\hat \theta$')
ax.set_xlabel(r'$\theta$')
ax.set_title('Posterior Samples from Inverse CDF')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., fontsize=12)
plt.tight_layout()
```



In [36]:
```
np.percentile(theta_inv_samp, 0)
```

Out[36]:
```
99.99992217050168
```

Answer 3.11

How does the 68% HPD compare with the confidence interval generated from bootstrapping? Why doesn't the bayesian interval construction suffer the same concerns you noted in 3.6

The 68% Highest Posterior Density came from the 68% left-most part of the right-skewed distribution, while the boostrapping confidence interval contains the middle 68% samples. Since the Bayesian posterior of θ,

$$\text{Pareto}(n + \alpha, \max(\beta, Y))$$

has a domain starting from $Y = \hat{\theta}$, the true θ is guaranteed to be captureded within this Highest Posterior Region.