This is a **closed-book test**: no books, no notes, no calculators, no phones, no tablets, no computers (of any kind) allowed.

Do **NOT** turn this page over until you are **TOLD** to start.

Duration of the test: 3 hours.

Please fill-in **ALL** the information requested on the front cover of **EACH** test booklet that you use.

The test consists of 4 pages, including this one. Make sure you have all 4 pages.

The test consists of 4 questions. **Answer all 4 questions**. The mark for each question is listed at the start of the question.

Write your answers in the test booklets provided.

The test was written with the intention that you would have ample time to complete it. You will be rewarded for concise well-thought-out answers, rather than long rambling ones. **We seek quality rather than quantity**.

Moreover, an answer that contains relevant and correct information as well as irrelevant or incorrect information will be awarded fewer marks than one that contains the same relevant and correct information only.

**Write legibly. Unreadable answers are worthless.**

1. [10 marks: 5 marks for each part]

   Consider the expression

$$\sqrt{1+\sin^2(x)} - \cos(x) \qquad \text{for } x \in [-\tfrac{\pi}{4}, \tfrac{\pi}{4}] \tag{1}$$

   When evaluated using IEEE Double-Precision Floating-Point Arithmetic, the computed value for expression (1) is inaccurate in a relative error sense for a range of values of $x$ within the interval $[-\frac{\pi}{4}, \frac{\pi}{4}]$, even though computed values of both $\sin(x)$ and $\cos(x)$ are accurate for all $x \in [-\frac{\pi}{4}, \frac{\pi}{4}]$.

   (a) Give a specific value of $x \in [-\frac{\pi}{4}, \frac{\pi}{4}]$ for which the computed value of expression (1) has a very large relative error. Show that the relative error is orders of magnitude larger than the *machine epsilon* (i.e., $\epsilon_{\text{mach}}$) for IEEE Double-Precision Floating-Point Numbers.

   (b) Find another expression that is mathematically equivalent to (1) that is accurate in a relative error sense for all $x \in [-\frac{\pi}{4}, \frac{\pi}{4}]$.

   Explain why you believe the computed value of your new expression is accurate in a relative error sense for all $x \in [-\frac{\pi}{4}, \frac{\pi}{4}]$.

2. [5 marks]

   In class, for a given $\lambda > 0$, we generated a doubly-exponential pseudo-random variable, $X$, with probability density function

$$f(x) = \frac{\lambda}{2}\,\mathrm{e}^{-\lambda|x|} \qquad \text{for } x \in \mathbb{R} \tag{2}$$

   by

   (a) sampling a pseudo-random variable $U_1 \sim \text{Unif}[0,1]$ and setting

$$\hat{X} = -\frac{1}{\lambda}\log(U_1)$$

   (Note: $\hat{X}$ is an exponential pseudo-random variable associated with the probability density function $g(x) = \lambda\,\mathrm{e}^{-\lambda x}$ for $x \geq 0$.)

   (b) sampling another pseudo-random variable $U_2 \sim \text{Unif}[0,1]$ and setting

$$X = \begin{cases} \hat{X} & \text{if } U_2 \leq 1/2 \\ -\hat{X} & \text{if } U_2 > 1/2 \end{cases}$$

   This method requires us to sample two Unif$[0,1]$ pseudo-random variables for each doubly-exponential pseudo-random variable, $X$, that it generates.

   For a given $\lambda > 0$, explain how you can generate a doubly-exponential pseudo-random variable, $X$, with probability density function (2) using only one Unif$[0,1]$ pseudo-random variable for each $X$ that you generate.

   Your method should not use any other pseudo-random variables, other than the one allowed Unif$[0,1]$ pseudo-random variable, to generate $X$.

3. [5 marks]

Let $f(x) = \alpha\hat{f}(x)$, where $\hat{f}(x) \geq 0$ for all $x \in \mathbb{R}$ and $\alpha > 0$ is the normalizing constant so that $\int f(x)\,dx = 1$. That is, $f(x)$ is a probability density function. In many practical problems, you know $\hat{f}(x)$, but you don't know $\alpha$, and $\alpha$ is expensive to compute. However, you may know another probability density function $g(x)$, having a shape similar to $\hat{f}(x)$, for which $\hat{f}(x) \leq Mg(x)$ for all $x \in \mathbb{R}$ and it may be relatively easy to find such an $M$ and to generate a pseudo-random variable $Y$ with probability density function $g(x)$.

Although I don't think I mentioned it in class and it is not discussed in Brandimarte's textbook, you can make a small modification to the acceptance-rejection method that we discussed in class so that you can use this modified version of the acceptance-rejection method to generate pseudo-random variables $X$ having probability density function $f(x)$ **without knowing** $\alpha$. Explain how this can be done.

4. [10 marks: 5 marks each part]

In Assignment 4, you used Monte Carlo with Importance Sampling to price a deep-out-of-the-money call option. You assumed that the underlying $S_t$ for the option satisfies the SDE

$$dS_t = rS_t dt + \sigma S_t dW_t$$

in the risk-neutral world, where $r$ is the risk-free interest rate and $\sigma$ is the volatility. The parameters for the option are:

- the initial stock price is $S_0 = \$80.00$,
- the strike price is $K = \$100.00$
- the time to maturity is $T = 0.25$ years,
- the risk-free interest rate is $r = 0.02$, and
- the volatility is $\sigma = 0.2$.

Another effective way to approximate the price of this deep-out-of-the-money call option is to use Stratified Sampling. To keep things simple, use just two strata:

- one stratum corresponds to $S_T < K$, and
- the other stratum corresponds to $S_T \geq K$.

One thing that you might find useful is that the $\hat{z}$ that satisfies the equation

$$K = S_0 \mathrm{e}^{(r - \frac{\sigma^2}{2})T + \sigma\sqrt{T}\hat{z}}$$

is

$$\hat{z} \approx 2.2314$$

and

$$\begin{aligned} p &= F(\hat{z}) \approx 0.98717 \\ q &= 1 - p \approx 0.012826 \end{aligned}$$

where $F$ is the CDF for the standard normal distribution (i.e., $N(0,1)$).

(a) Describe how you could use Monte Carlo Stratified Sampling with the two strata described above to approximate efficiently the price of this deep-out-of-the-money call option. In describing your algorithm, you can assume you have access to functions that compute $F$, $F^{-1}$ (in MatLab these functions are `normcdf` and `norminv`, respectively) and a pseudo-random number generator (such as `rand` in MatLab) that computes uniform $[0, 1]$ pseudo-random numbers.

Provide sufficient detail in the description of your Monte Carlo Stratified Sampling method so that an experienced MatLab programmer with no knowledge of computational finance can implement your method in MatLab.

(b) Do you expect that your method described in part (a) will be significantly more efficient than simple Monte Carlo for pricing this deep-out-of-the-money call option?

Be as quantative as possible in answering this question.