This is a **closed-book exam**: 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 exam: 3 hours.

The exam consists of 8 pages, including this one. Make sure you have all 8 pages.

The exam consists of 6 questions. **Answer all 6 questions** in the exam booklets provided. The mark for each question is listed at the start of the question.

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

The exam 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. [5 marks]

   Joe knew that there was something wrong with his MatLab program, but he didn't know how to fix it. He traced the problem to the statement

   ```
   F = ( 1 - cos(x) ) / x^2                    (1)
   ```

   in his program. If he evaluated the statement (1) for small, but positive, values of `x`, he got very inaccurate answers. For example, if he evaluated the statement (1) for `x = 1.0e-8`, he got that the computed value of `F` is 0, but he knew the true value of `F` should be very close to 1/2.

   Explain why the statement (1) computes such inaccurate values of `F` for such small, but positive, values of `x` and show Joe how to rewrite the statement (1) so that it computes accurate values of `F` for such small, but positive, values of `x` as well as for larger values of x (e.g., $x = 1$).

2. [5 marks]

Harry was also having trouble with his MatLab program, and, like Joe, he didn't know how to fix it. He traced the problem to the statement

```
G = sqrt( x^2 + y^2 )                    (2)
```

in his program. However, he seemed to have two problems with the statement (2):

- If `x` and/or `y` were very large in magnitude, then the computed value of `G` was `Inf` even though the true value of `G` could be approximated very accurately by a normalized IEEE double-precision floating-point number. For example, for `x = 1.0e+200` and `y = 1.0`, the computed value of `G` was `Inf`, whereas the true value of `G` is very close to `1.0e+200`, which is orders of magnitude smaller than the largest normalized IEEE double-precision floating-point number.
- If `x` and `y` were both very small in magnitude, then the computed value of `G` was 0, even though the true value of `G` was orders of magnitude larger than the smallest positive normalized IEEE double-precision floating-point number. For example, for `x = y = 1.0e-200`, the computed value of `G` is 0, while the true value of `G` is $\sqrt{2}$ `* 1.0e-200`, which is orders of magnitude larger than the smallest positive normalized IEEE double-precision floating-point number.

Show Harry how to rewrite the statement (2) so that it always produces an accurate approximation to the true value of `G` whenever this is possible. In particular, your new expression should not overflow unless the true value of `G` overflows and the computed value of `G` should not underflow if `x` and `y` are normalized IEEE double-precision floating-point numbers. (However, your new expression could have an underflow in an intermediate expression, but this underflow should be "harmless".)

Hint 1: when you rewrite (2), you may find it helpful to use either an if–then-else statement or the min and max functions.

Hint 2: the largest IEEE double-precision floating-point number is about `1.7977e+308` and the smallest positive normalized IEEE double-precision floating-point number is about `2.2251e-308`.

3. [15 marks: 5 marks for each part]

Suppose you want to generate a pseudo-random variable X having the pdf

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

Although you can write the associated CDF

$$F(x) = \int_{-\infty}^{x} f(t)\, dt \qquad \text{for } x \in \mathbb{R}$$

in closed form (using the error function, $\operatorname{erf}(x)$), I don't think you can easily use the CDF $F(x)$ in the inverse transform method to generate a pseudo-random variable X. However, for an appropriately chosen $\gamma > 0$, $f(x)$ is fairly similar to

$$g_\gamma(x) = \gamma\, |x|\, \mathrm{e}^{-\gamma x^2} \qquad \text{for } x \in \mathbb{R}$$

Moreover,

$$G_\gamma(x) = \int_{-\infty}^{x} g_\gamma(t)\, dt = \begin{cases} \frac{1}{2}\, \mathrm{e}^{-\gamma x^2} & \text{for } x \le 0 \\ 1 - \frac{1}{2}\, \mathrm{e}^{-\gamma x^2} & \text{for } x \ge 0 \end{cases}$$

(a) For any $\gamma > 0$, explain how you can use the CDF $G_\gamma(x)$ to generate a pseudo-random variable Y having pdf $g_\gamma(x)$ and CDF $G_\gamma(x)$.

(b) In part (c) below, you are asked to use the acceptance-rejection method to generate a pseudo-random variable X having the pdf $f(x)$ using the the proposal pseudo-random variable Y having the pdf $g_\gamma(x)$. How should you choose the free parameter $\gamma > 0$ in $g_\gamma(x)$ to make the acceptance-rejection method developed in part (c) as efficient as possible?

Note that you can answer part (b) even if you didn't answer part (a) or you got the wrong answer to part (a).

(c) Use the results from parts (a) and (b) above to write an acceptance-rejection method for the pseudo-random variable X having the pdf $f(x)$ using the proposal pseudo-random variable Y having the pdf $g_{\gamma^*}(x)$, where γ^* is your choice of γ from part (b).

If you didn't do part (a), just assume that you can generate such a proposal pseudo-random variable Y. If you didn't do part (b), use any $\gamma^* \in (0, 1)$ for part (c).

4. [10 marks: 5 marks for each part]

Suppose we want to compute the price at time $t = 0$ of an *exchange option* having the payoff

$$P_T = \max(S_T^{(1)} - S_T^{(2)}, 0) \tag{3}$$

at time $t = T$, where $S_t^{(1)}$ and $S_t^{(2)}$ satisfy the SDEs

$$\begin{aligned} dS_t^{(1)} &= rS_t^{(1)}\,dt + \sigma_1 S_t^{(1)}\,dW_t^{(1)}, \qquad S_0^{(1)} \text{ given} \\ dS_t^{(2)} &= rS_t^{(2)}\,dt + \sigma_2 S_t^{(2)}\,dW_t^{(2)}, \qquad S_0^{(2)} \text{ given} \end{aligned}$$

Assume that the parameters r, σ_1, σ_2 and T are known and $W_t^{(1)}$ and $W_t^{(2)}$ are two uncorrelated standard Brownian motions. (It would be more realistic to assume that $W_t^{(1)}$ and $W_t^{(2)}$ are correlated, but I think this makes the problem harder.)

We could use a simple Monte Carlo method to compute the price of the exchange option described above, but it is often very helpful to use some variance reduction technique. In this case, we could use Control Variates. To this end, note that if we change the payoff (3) to

$$\hat{P}_T = \max(S_T^{(1)} - K, 0) \tag{4}$$

where $K = \mathbb{E}[S_T^{(2)}]$, then the payoff (4) is just the payoff of a standard call option in the Black-Scholes framework. So, we know the price of this option in closed form.

(a) For the Control Variates method we discussed in class, we want to compute $\mathbb{E}[X]$ for some random variable X. There is another related random variable Z for which we know $\mathbb{E}[Z]$. In the Control Variates method, we introduce another random variable

$$Y = X + c\Big(Z - \mathbb{E}[Z]\Big) \tag{5}$$

for an appropriately chosen c and do a Monte Carlo simulation to estimate $\mathbb{E}[Y]$. Discuss how you can use the price of the call option associated with the payoff (4) as a Control Variate for the exchange option discussed above. In particular,

(i) explain what X and Z are in this case,

(ii) explain how to compute $\mathbb{E}[Z]$ in this case,

(iii) explain how to compute $K = \mathbb{E}[S_T^{(2)}]$,

(iv) explain how to compute the parameter c in (5).

(b) Write a MatLab pseudo-code to implement your Control-Variates Monte-Carlo method described in part (a). You can use all the standard MatLab functions such as `rand, randn, blsprice,` etc.

(Note: your MatLab pseudo-code does not have to be syntactically correct MatLab. All that is required is that we understand what you are trying to compute.)

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

Suppose we want to compute the price at time $t = 0$ of a *barrier option* having the payoff

$$P_T = \begin{cases} \max(S_T - K_1, 0) & \text{if } S_{T/2} \le L \\ \max(S_T - K_2, 0) & \text{if } S_{T/2} > L \end{cases} \tag{6}$$

at time $t = T$, where S_t satisfies the SDE

$$dS_t = rS_t\, dt + \sigma S_t\, dW_t \qquad S_0 \text{ given}$$

Assume that the parameters r, σ, K_1, K_2, L and T are known and W_t is a standard Brownian motion. The price of the option at time $t = 0$ can be written as

$$C_0 = \mathbb{E}^Q\left[\mathrm{e}^{-rT}\left(\max(S_T - K_1, 0) I_{\{S_{T/2} \le L\}} + \max(S_T - K_2, 0) I_{\{S_{T/2} > L\}}\right)\right]$$

where Q is the risk-free probability measure and

$$I_{\{\text{condition}\}} = \begin{cases} 1 & \text{if condition is true} \\ 0 & \text{if condition is false} \end{cases}$$

For iterate n of a simple Monte Carlo method, you could

(i) generate two independent standard normal random variables $Z_1 \sim N(0,1)$ and $Z_2 \sim N(0,1)$,

(ii) let

$$\begin{aligned} S_{T/2} &= S_0 \mathrm{e}^{(r-\frac{\sigma^2}{2})\frac{T}{2} + \sigma\sqrt{\frac{T}{2}} Z_1} \\ S_T &= S_{T/2} \mathrm{e}^{(r-\frac{\sigma^2}{2})\frac{T}{2} + \sigma\sqrt{\frac{T}{2}} Z_2} \end{aligned}$$

(iii) use the $S_{T/2}$ and S_T computed in step (ii) above to compute

$$C_0^{(n)} = \mathrm{e}^{-rT}\left(\max(S_T - K_1, 0) I_{\{S_{T/2} \le L\}} + \max(S_T - K_2, 0) I_{\{S_{T/2} > L\}}\right)$$

The Monte Carlo estimate

$$\hat{C}_0 = \frac{1}{N} \sum_{n=1}^{N} C_0^{(n)}$$

is an approximation to the option price C_0.

We can improve upon the Monte Carlo method described above by using conditioning as a variance reduction method. The way conditioning is described in your textbook, if you want to compute $\mathbb{E}[X]$, the expectation of a random variable X, you use

$$\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]]$$

for some random variable Y related to X. This method is most effective when you can compute $\mathbb{E}[X|Y]$ in closed form (i.e., you don't have to use a Monte Carlo simulation to estimate it).

(a) For the barrier option pricing problem described above, take

$$X = \mathrm{e}^{-rT}\left(\max(S_T - K_1, 0) I_{\{S_{T/2} \le L\}} + \max(S_T - K_2, 0) I_{\{S_{T/2} > L\}}\right)$$

(Actually, you have to rewrite X a little to answer part (i) below, but, if I tell you how to rewrite X, it will give away too much of the answer).

Discuss how you can choose Y such that

(i) you can compute $\mathbb{E}[X|Y]$ in closed form,

(ii) you need to use only one standard normal random variable $Z \sim N(0,1)$ for each Monte Carlo iteration for this conditioning approach.

(b) Write a MatLab pseudo-code to implement your Conditioning Monte-Carlo method described in part (a). You can use all the standard MatLab functions such as `rand, randn, blsprice,` etc.

(Note: your MatLab pseudo-code does not have to be syntactically correct MatLab. All that is required is that we understand what you are trying to compute.)

6. [10 marks: 5 marks for each part]

Consider the transport PDE
$$\frac{\partial \phi}{\partial t} + c\frac{\partial \phi}{\partial x} = 0$$
where $\phi = \phi(x,t)$ and $c > 0$. Assume also that ϕ satisfies the initial condition
$$\phi(x,0) = f(x) \qquad \text{for all } x \in \mathbb{R}$$

One numerical method that we did not consider in class for this problem is
$$\frac{\phi_n^{m+1} - \phi_n^m}{\delta t} + c\frac{\phi_{n+1}^m - \phi_{n-1}^m}{2\delta x} = 0 \tag{7}$$
where $\delta t > 0$ is the stepsize in the t direction, $\delta x > 0$ is the stepsize in the x direction and $\phi_n^m \approx \phi(n\delta x, m\delta t)$. You can rewrite (7) in the computationally more convenient form
$$\phi_n^{m+1} = \phi_n^m - c\,\frac{\delta t}{2\delta x}\left(\phi_{n+1}^m - \phi_{n-1}^m\right)$$
which in turn can be rewritten as
$$\phi_n^{m+1} = \phi_n^m + \rho\left(\phi_{n+1}^m - \phi_{n-1}^m\right) \tag{8}$$
where $\rho = -c\,\dfrac{\delta t}{2\delta x}$.

(a) What is the order of consistency of the numerical method (7)?

That is, find the truncation error associated with the numerical method (7), show that the truncation error can be written in the form $\mathcal{O}((\delta t)^p) + \mathcal{O}((\delta x)^q)$, and state the largest values possible for p and q.

(b) Is the numerical method (7) stable?

To answer this, it might be easier to consider the equivalent form (8) of the numerical method (7).

Have a Happy Holiday