Solution to the MMF 2021 Exam

1. [5 marks]

   The students were told that the MatLab statement

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

   produced inaccurate results if `x` is small, but positive. For example, for `x = 1.0e-8`, the computed value of `F` is 0, but the true value of `F` is very close to 1/2.

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

   The reason why (1) computes such inaccurate values of `F` for small, but positive, values of `x` is that there is *catastrophic cancellation* between 1 and `cos(x)` when `x` is small. That is because

   $$\cos(x) = 1 - x^2/2 + \mathcal{O}(x^4) \tag{2}$$

   So, for `x = 1.0e-8` for example, the computed value of `cos(x)` is 1, and the $\texttt{x}^2/2 + \mathcal{O}(\texttt{x}^4)$ term is lost from the computed value of (2). Hence, the computed value of `1 - cos(x)` is 0, due to catastrophic cancellation.

   On the other hand, the true value of `1 - cos(x)` is $\texttt{x}^2/2 + \mathcal{O}(\texttt{x}^4)$. Therefore, for `x` sufficiently small, but nonzero, the computed value of `F` is zero, but the true value is small, but nonzero. Therefore, the computed value is very inaccurate in a relative error sense.

   We can find a mathematically equivalent expression for (1) that is computationally superior to (1) by using the "multiplying by the conjugate" trick:

   $$\begin{aligned}\frac{1-\cos(x)}{x^2} &= \frac{1-\cos(x)}{x^2}\,\frac{1+\cos(x)}{1+\cos(x)} \\ &= \frac{1-\cos^2(x)}{x^2(1+\cos(x))} \\ &= \frac{\sin^2(x)}{x^2(1+\cos(x))}\end{aligned}$$

   This last expression does not suffer from any cancellation if $\cos(x) \geq 0$, which is the case if $x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.

   Hence, we can replace (1) by

   ```
   F = (sin(x))^2 / ( x^2 * (1 + cos(x)) )
   ```
   (3)

   The new expression (3) computes a very accurate approximation to the true value of `F` if `x` is small, but nonzero. For example, (3) is very accurate for `x = 1.0e-8`. It is also very accurate for all nonzero normalized floating-point numbers in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$, excluding `x = 0`.

2. [5 marks]

The students were told that the MatLab statement

```
G = sqrt( x^2 + y^2 )
```
(4)

has problems with

- overflows if `x` and/or `y` are very large in magnitude, and
- underflows if `x` and `y` are both very small in magnitude.

See the exam paper for details.

They were asked to rewrite the statement (4) in a mathematically equivalent form so that it always produces an accurate approximation to the true value of `G` whenever this is possible. In particular, they were asked to show that their new expression does not overflow unless the true value of `G` overflows and the computed value of `G` does not underflow if `x` and `y` are normalized IEEE double-precision floating-point numbers. (However, their new expression could have an underflow in an intermediate expression, but this underflow should be "harmless".)

I gave them the following two hints.

Hint 1: when you rewrite (4), 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`.

I'll use an if–then–else statement in my solution below, but it is easy to see how my approach can be adapted to compute `G` using a little MatLab code fragment that contains min and max functions.

I should have also said that they can use the MatLab abs function and an if–then–else statement even in the min/max approach. Accept any reasonable MatLab code fragment to compute a mathematically equivalent, but computationally superior, form of `G`.

To rewrite (4) as a mathematically equivalent expression that is computationally superior to (4), first note that, if $|x| > |y|$ (whence $x \neq 0$), then

$$\sqrt{x^2 + y^2} = |x|\sqrt{1 + (y/x)^2} \tag{5}$$

The computed value of the right side of (5) cannot overflow unless the true value of $\sqrt{x^2 + y^2}$ overflows. The term $(y/x)^2$ can underflow, but this underflow is harmless,

since, when coupled with $1+(y/x)^2$, it results in a very small relative error in the computed value of $1+(y/x)^2$. Finally, note that $|x|\sqrt{1+(y/x)^2} \geq |x|$, so this computation cannot result in an underflow if x is a normalized IEEE double-precision floating-point number. Thus, the right side of (5) is a computationally effective way to evaluate the left side of (5) in the case $|x| > |y|$ (whence $x \neq 0$).

Similarly, if $|x| \leq |y|$ and $y \neq 0$, then

$$\sqrt{x^2+y^2} = |y|\sqrt{1+(x/y)^2} \tag{6}$$

Finally, if x and y are both zero, then

$$\sqrt{x^2+y^2} = 0 \tag{7}$$

We can combine (5)–(7) to get the following computationally effective MatLab code fragment to evaluate `G`.

```
if (x == 0 & y == 0)
   G = 0;
elseif (abs(x) > abs(y))
   G = abs(x) * sqrt(1 + (y/x)^2);
else  % note in this case abs(x) <= abs(y) and y != 0
   G = abs(y) * sqrt(1 + (x/y)^2);
end
```

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

I gave them the pdf

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

for a pseudo-random variable X. Obviously, the associated CDF is

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

However, I told them that 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,

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

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

You can use the inverse transform method to generate a pseudo-random variable Y having pdf $g_\gamma(x)$ and CDF $G_\gamma(x)$. To this end, first generate a uniform pseudo-random variable $U \sim \mathrm{Unif}[0, 1]$ and then solve $G_\gamma(Y) = U$ for the pseudo-random variable Y having pdf $g_\gamma(x)$ and CDF $G_\gamma(x)$.
Note that $G_\gamma(0) = 1/2$. Therefore, if $U \leq 1/2$, we solve

$$\frac{1}{2}\mathrm{e}^{-\gamma Y^2} = U \tag{8}$$

for Y. Note also that $Y \leq 0$ in this case. Therefore, the solution of (8) that we seek is

$$Y = -\sqrt{-\log(2U)/\gamma}$$

On the other hand, if $U > 1/2$, we solve

$$1 - \frac{1}{2}\, \mathrm{e}^{-\gamma Y^2} = U \tag{9}$$

for Y. Note also that $Y > 0$ in this case. Therefore, the solution of (9) that we seek is

$$Y = \sqrt{-\log(2(1-U))/\gamma}$$

Putting this together, we get the following method to generate a pseudo-random variable Y having pdf $g_\gamma(x)$ and CDF $G_\gamma(x)$.

Generate $U \sim \text{Unif}[0,1]$
if $U \leq 1/2$ then
$\quad Y = -\sqrt{-\log(2U)/\gamma}$
else
$\quad Y = \sqrt{-\log(2(1-U))/\gamma}$
end

(b) In part (c) below, the students are asked to use the acceptance-rejection method to generate a pseudo-random variable X having the pdf $f(x)$ using the proposal pseudo-random variable Y having the pdf $g_\gamma(x)$. In this part of the question (i.e., in part (b)), they are asked how to choose the free parameter $\gamma > 0$ in $g_\gamma(x)$ to make the acceptance-rejection method developed in part (c) as efficient as possible.

I also told them that they can answer this part of the question (i.e., part (b)) even if you didn't answer part (a) or they got the wrong answer to part (a).

A requirement of the acceptance-rejection method is that you have a constant c_γ such that

$$f(x) \le c_\gamma \, g_\gamma(x) \qquad \text{for all } x \in \mathbb{R} \tag{10}$$

I called the constant c_γ, rather than just c, because it obviously depends on the choice of γ. Also, later we'll minimize c_γ with respect to γ. However, for now, assume γ is a fixed constant.

For each fixed γ, we want to find the smallest c_γ that satisfies (10), since the smaller c_γ is the more efficient the acceptance-rejection method is. Clearly, for γ fixed, the smallest constant c_γ that satisfies (10) is

$$c_\gamma = \max_{x \in \mathbb{R}} \frac{f(x)}{g_\gamma(x)} \tag{11}$$

Let

$$h_\gamma(x) = \frac{f(x)}{g_\gamma(x)} = \frac{2}{\gamma\sqrt{\pi}} \, |x| \, \mathrm{e}^{-x^2(1-\gamma)} \tag{12}$$

We see from (12) that, if $\gamma \ge 1$, then $h_\gamma(x) \to \infty$ as $x \to \pm\infty$, which implies that $c_\gamma = \infty$. However, c must be finite in the acceptance-rejection method. So, we must have $\gamma < 1$. Combining this with the requirement that $\gamma > 0$ (stated at the start of the question and required for $g_\gamma(x)$ to be a pdf), we have that γ must satisfy $0 < \gamma < 1$. So, for the rest of this question, assume $0 < \gamma < 1$.

Now consider a fixed $\gamma \in (0, 1)$ and consider maximizing $h_\gamma(x)$ with respect to x. First note that $h_\gamma(x)$ is symmetric about $x = 0$ (i.e., $h_\gamma(-x) = h_\gamma(x)$). Therefore, we need only consider maximizing $h_\gamma(x)$ for $x \ge 0$. Also note that, for $x \ge 0$,

$$h_\gamma(x) = \frac{2}{\gamma\sqrt{\pi}} \, x \, \mathrm{e}^{-x^2(1-\gamma)}$$

Moreover, $h_\gamma(0) = 0$ and $h_\gamma(x) \to 0$ as $x \to \infty$. So, the maximum of $h_\gamma(x)$ must occur at some finite positive value of x. To find the maximum of $h_\gamma(x)$, consider

$$h'_\gamma(x) = \frac{2}{\gamma\sqrt{\pi}} \mathrm{e}^{-x^2(1-\gamma)} \Big(1 - 2x^2(1-\gamma) \Big)$$

Note that $h'_\gamma(x) = 0$ if and only if

$$1 - 2x^2(1-\gamma) = 0 \tag{13}$$

Since

$$x^* = \frac{1}{\sqrt{2(1-\gamma)}}$$

is the only positive root of (13), x^* is the only positive root of $h'_\gamma(x)$, whence the maximum of $h_\gamma(x)$ occurs at x^* and

$$h_\gamma(x^*) = \frac{2}{\gamma\sqrt{\pi}} \frac{1}{\sqrt{2(1-\gamma)}} \mathrm{e}^{-\frac{1}{2}} = \sqrt{\frac{2}{\pi \mathrm{e}}} \frac{1}{\gamma\sqrt{1-\gamma}} \tag{14}$$

Thus, we get from (11), (12) and (14) that

$$c_\gamma = \sqrt{\frac{2}{\pi \mathrm{e}}} \frac{1}{\gamma\sqrt{1-\gamma}} \tag{15}$$

Recall that our goal is to choose $\gamma \in (0,1)$ to make c_γ as small as possible. To this end, we change the notation c_γ to the functional notation $c(\gamma)$, where

$$c(\gamma) = c_\gamma = \sqrt{\frac{2}{\pi \mathrm{e}}} \frac{1}{\gamma\sqrt{1-\gamma}}$$

since we want to minimize $c(\gamma)$ with respect to γ for $\gamma \in (0,1)$. To this end, note that

$$\begin{aligned}
c'(\gamma) &= \sqrt{\frac{2}{\pi \mathrm{e}}} \left(\frac{1}{2} \gamma^{-1}(1-\gamma)^{-3/2} - \gamma^{-2}(1-\gamma)^{-1/2} \right) \\
&= \sqrt{\frac{2}{\pi \mathrm{e}}} \gamma^{-1}(1-\gamma)^{-1/2} \left(\frac{1}{2} (1-\gamma)^{-1} - \gamma^{-1} \right) \\
&= \sqrt{\frac{2}{\pi \mathrm{e}}} \frac{1}{\gamma(1-\gamma)^{1/2}} \frac{\frac{1}{2}\gamma - (1-\gamma)}{\gamma(1-\gamma)} \\
&= \sqrt{\frac{2}{\pi \mathrm{e}}} \frac{\frac{3}{2}\gamma - 1}{\gamma^2(1-\gamma)^{3/2}}
\end{aligned}$$

Since $\gamma^* = 2/3$ is the unique root of $c'(\gamma)$ for $\gamma \in (0,1)$ and $c(\gamma) \to \infty$ as $\gamma \to 0$ and $\gamma \to 1$, $\gamma^* = 2/3$ must be the minimizer of $c(\gamma)$ for $\gamma \in (0,1)$. So the γ they should use for their acceptance-rejection method in part (c) is $\gamma^* = 2/3$.

The corresponding c_γ is

$$\begin{aligned} c_{\gamma^*} &= c(\gamma^*) \\ &= c(2/3) \\ &= \sqrt{\frac{2}{\pi \mathrm{e}}} \frac{1}{\frac{2}{3}\sqrt{1-\frac{2}{3}}} \\ &= \sqrt{\frac{2}{\pi \mathrm{e}}} \frac{3}{2}\sqrt{3} \\ &= \frac{3}{2}\sqrt{\frac{6}{\pi \mathrm{e}}} \end{aligned}$$

They wouldn't be able to calculate the value of the final expression above on the exam, but it is easy to compute that

$$c_{\gamma^*} \approx 1.2573$$

Hence, using $\gamma = \gamma^* = 2/3$, the probability of acceptance in the acceptance-rejection method in part (c) below is

$$\frac{1}{c_{\gamma^*}} \approx 0.79534 \approx 0.8$$

So, this turns out to be a fairly effective acceptance-rejection method.

(c) I asked the students to 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 their choice of γ from part (b).

I also told them that, if they didn't do part (a), they can assume that they can generate such a proposal pseudo-random variable Y. Also, if they didn't do part (b), they can use any $\gamma^* \in (0, 1)$ for part (c).

This question essentially comes down to writing out the acceptance-rejection method. My MatLab pseudo-code for this follows.

1. Let $\gamma^* = 2/3$ and $c_{\gamma^*} = \frac{3}{2}\sqrt{\frac{6}{\pi \mathrm{e}}}$
2. Generate a $U \sim \text{Unif}[0, 1]$ and a $Y \sim G_{\gamma^*}(x)$
3. if $U \leq \dfrac{f(Y)}{c_{\gamma^*}\, g_{\gamma^*}(Y)}$ then
4. accept $X = Y$ and stop
5. else
6. reject Y and go to step 2
7. end

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

I asked the students to consider computing the price at time $t = 0$ of an *exchange option* having the payoff

$$P_T = \max(S_T^{(1)} - S_T^{(2)}, 0) \tag{16}$$

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

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

I also told them to 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.

I also told them to consider using the Control Variates variance reduction technique with the payoff

$$\hat{P}_T = \max(S_T^{(1)} - K, 0) \tag{17}$$

where $K = \mathbb{E}[S_T^{(2)}]$. I reminded them that the payoff (17) is just the payoff of a vanilla European call option in the Black-Scholes framework. So, they know the price of this option in closed form.

(a) I also reminded them that, 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

$$Y = X + c\Big(Z - \mathbb{E}[Z]\Big) \tag{18}$$

for an appropriately chosen c and do a Monte Carlo simulation to estimate $\mathbb{E}[Y]$. I asked them to discuss how you can use the price of the vanilla European call option associated with the payoff (17) 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 (18).

Here are my answers to parts (i)–(iv) above.

(i) X is the random variable associated with the discounted payoff (16). That is,

$$X = \mathrm{e}^{-rT} \max(S_T^{(1)} - S_T^{(2)}, 0)$$

The price of the associated exchange option with payoff (16) is $\mathbb{E}[X]$.
Z is the random variable associated with the discounted payoff (17). That is,

$$Z = \mathrm{e}^{-rT} \max(S_T^{(1)} - K, 0)$$

where $K = \mathbb{E}[S_T^{(2)}]$. Note $\mathbb{E}[Z]$ is the price of a vanilla European call option in the Black-Scholes framework.

(ii) As noted in (i) above, $\mathbb{E}[Z]$ is the price of a vanilla European call option in the Black-Scholes framework. Therefore, we can use the Black-Scholes formula to compute $\mathbb{E}[Z]$. In MatLab, we could compute $\mathbb{E}[Z]$ as follows.

[Call,Put] = blsprice($S_0^{(1)}$, K, r, T, σ_1)
$\mathbb{E}[Z]$ = Call

where $K = \mathbb{E}[S_T^{(2)}]$. (See (iii) below for how to compute $K = \mathbb{E}[S_T^{(2)}]$.)

(iii) We used several times in class that, if

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

then

$$\mathbb{E}[S_T] = S_0\,\mathrm{e}^{rT}$$

Hence,

$$K = \mathbb{E}[S_T^{(2)}] = S_0^{(2)}\,\mathrm{e}^{rT}$$

Alternatively, they could use that

$$S_T^{(2)} = S_0^{(2)}\,\mathrm{e}^{\left(r - \frac{\sigma_2^2}{2}\right)T + \sigma_2\sqrt{T}\,N} \tag{19}$$

where N is a standard normal random variable (i.e., $N \sim N(0,1)$), and then compute $\mathbb{E}[S_T^{(2)}]$ from (19) by applying the standard formula for computing the expectation of a function of a random variable.

(iv) The optimal choice for c in (18) is

$$c^* = -\frac{\mathrm{Cov}(X,Z)}{\mathrm{Var}(Z)}$$

We can approximate c^* by doing a pilot computation. That is, compute several samples of X and Z and use the MatLab functions cov and var to compute $\mathrm{Cov}(X,Z)$ and $\mathrm{Var}(Z)$.

(b) I asked the students to write a MatLab pseudo-code to implement the Control-Variates Monte-Carlo method described in part (a). I told them that they can use all the standard MatLab functions such as `rand, randn, blsprice,` etc.
I also told them that their MatLab pseudo-code does not have to be syntactically correct MatLab. All that is required is that we understand what they are trying to compute.

Here's my MatLab pseudo-code.

```
% Initialize the parameters r, sigma1 = σ1, sigma2 = σ2, T,
% S10 = S0^(1) and S20 = S0^(2) for the exchange option

% First do a pilot computation to approximate c = - cov(X,Z) / var(Z)

M = 10000; % Any reasonable value for M is fine here.

N1 = randn(M,1);
S1T = S10 * exp( (r - sigma1^2/2)*T + sigma1*sqrt(T)*N1 );

N2 = randn(M,1);
S2T = S20 * exp( (r - sigma2^2/2)*T + sigma2*sqrt(T)*N2 );

X = exp(-r*T) * max(S1T - S2T, 0);

K = S20 * exp(r*T);
Z = exp(-r*T) * max(S1T - K, 0);

C = cov(X,Z);
c = - C(1,2) / var(Z);

% Use the Black-Scholes formula to compute the exact value of E[Z]
[Call,Put] = blsprice(S10, K, r, T, sigma1);
EZ = Call;

% Now do a Monte Carlo simulation using control variates to compute the
% price of the exchange option.

M = 1000000; % Any reasonable value for M is fine here.

N1 = randn(M,1);
S1T = S10 * exp( (r - sigma1^2/2)*T + sigma1*sqrt(T)*N1 );

N2 = randn(M,1);
S2T = S20 * exp( (r - sigma2^2/2)*T + sigma2*sqrt(T)*N2 );

X = exp(-r*T) * max(S1T - S2T, 0);
```

```
Z = exp(-r*T) * max(S1T - K, 0);
Y = X + c * (Z - EZ);
price = mean(Y);
```

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

I asked the students to consider pricing at time $t = 0$ a *barrier option* having the payoff

$$P_T = \begin{cases} \max(S_T - K_1, 0) & \text{if } S_{T/2} \le L \\ \max(S_T - K_2, 0) & \text{if } S_{T/2} > L \end{cases} \tag{20}$$

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

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t \qquad S_0 \text{ given} \tag{21}$$

I told them to assume that the parameters r, σ, K_1, K_2, L and T are known and W_t is a standard Brownian motion. I also told them that, if we let

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

then the price of the option is

$$C_0 = \mathbb{E}[X]$$

I also gave them an outline of a simple Monte Carlo method for approximating C_0. (See the exam paper for the details.)

I also told them that we can improve upon the simple Monte Carlo method I gave them by using conditioning as a variance reduction method.

The way conditioning is described in the course textbook, if you want to compute $\mathbb{E}[X]$, 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, I asked them to 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.

The key idea here is to condition on $Y = S_{T/2}$ as follows.

$$
\begin{aligned}
\mathbb{E}[X] &= \mathbb{E}\left[\mathrm{e}^{-rT}\left(\max(S_T-K_1,0)I_{\{S_{T/2}\le L\}}+\max(S_T-K_2,0)I_{\{S_{T/2}>L\}}\right)\right] \\
&= \mathbb{E}\left[\mathrm{e}^{-rT/2}\left(\mathrm{e}^{-rT/2}\max(S_T-K_1,0)I_{\{S_{T/2}\le L\}}\right.\right. \\
&\qquad\left.\left.+\ \mathrm{e}^{-rT/2}\max(S_T-K_2,0)I_{\{S_{T/2}>L\}}\right)\right] \\
&= \mathbb{E}\left[\mathrm{e}^{-rT/2}\,\mathbb{E}\left[\mathrm{e}^{-rT/2}\max(S_T-K_1,0)I_{\{S_{T/2}\le L\}}\right.\right. \\
&\qquad\left.\left.+\ \mathrm{e}^{-rT/2}\max(S_T-K_2,0)I_{\{S_{T/2}>L\}}\,\middle|\,S_{T/2}\right]\right] \\
&= \mathbb{E}\left[\mathrm{e}^{-rT/2}\left(\mathbb{E}\left[\mathrm{e}^{-rT/2}\max(S_T-K_1,0)I_{\{S_{T/2}\le L\}}\,\middle|\,S_{T/2}\right]\right.\right. \\
&\qquad\left.\left.+\ \mathbb{E}\left[\mathrm{e}^{-rT/2}\max(S_T-K_2,0)I_{\{S_{T/2}>L\}}\,\middle|\,S_{T/2}\right]\right)\right]
\end{aligned}
\tag{22}
$$

Now note that we know both

$$
\mathbb{E}\left[\mathrm{e}^{-rT/2}\max(S_T-K_1,0)I_{\{S_{T/2}\le L\}}\,\middle|\,S_{T/2}\right]
$$

$$
\mathbb{E}\left[\mathrm{e}^{-rT/2}\max(S_T-K_2,0)I_{\{S_{T/2}>L\}}\,\middle|\,S_{T/2}\right]
$$

in closed form, since, if we know $S_{T/2}$, we know both $I_{\{S_{T/2}\le L\}}$ and $I_{\{S_{T/2}\le L\}}$ and we also know that

$$
\mathbb{E}\left[\mathrm{e}^{-rT/2}\max(S_T-K_1,0)\,\middle|\,S_{T/2}\right]
$$

$$
\mathbb{E}\left[\mathrm{e}^{-rT/2}\max(S_T-K_2,0)\,\middle|\,S_{T/2}\right]
$$

are just the prices of two vanilla European call options, each starting with the value $S_{T/2}$ at time $t=T/2$, but having strike prices K_1 and K_2, respectively. Therefore, their values can be computed in closed form from the Black-Scholes formula.

To be more specific, let $C(S_{T/2},K,r,T/2,\sigma)$ be the price of a vanilla European call option with strike price K, risk free interest rate r, volatility σ, where the underlying S_t has value $S_{T/2}$ at time $t=T/2$ and evolves according to the SDE (21) for $t\in(T/2,T]$ and expires at time $t=T$. Then, from the equations (22) above,

$$
\begin{aligned}
\mathbb{E}[X] &= \mathbb{E}\left[\mathrm{e}^{-rT}\left(\max(S_T-K_1,0)I_{\{S_{T/2}\le L\}}+\max(S_T-K_2,0)I_{\{S_{T/2}>L\}}\right)\right] \\
&= \mathbb{E}\left[\mathrm{e}^{-rT/2}\left(\mathbb{E}\left[\mathrm{e}^{-rT/2}\max(S_T-K_1,0)I_{\{S_{T/2}\le L\}}\,\middle|\,S_{T/2}\right]\right.\right. \\
&\qquad\left.\left.+\ \mathbb{E}\left[\mathrm{e}^{-rT/2}\max(S_T-K_2,0)I_{\{S_{T/2}>L\}}\,\middle|\,S_{T/2}\right]\right)\right] \\
&= \mathbb{E}\left[\mathrm{e}^{-rT/2}\left(C(S_{T/2},K_1,r,T/2,\sigma)I_{\{S_{T/2}\le L\}}\right.\right. \\
&\qquad\left.\left.+C(S_{T/2},K_2,r,T/2,\sigma)I_{\{S_{T/2}>L\}}\right)\right]
\end{aligned}
\tag{23}
$$

Note that a Monte Carlo method to approximate the final expectation of (23) above needs to simulate $S_{T/2}$ only; it does not need to simulate S_T. So, we need only one standard normal random variable $Z \sim N(0,1)$ to compute

$$S_{T/2} = S_0 \mathrm{e}^{\left(r-\frac{\sigma^2}{2}\right)T/2+\sigma\sqrt{T/2}Z}$$

for each Monte Carlo iteration for this conditioning approach that culminates in the final expectation of (23) above.

(b) I asked them to write a MatLab pseudo-code to implement their Conditioning Monte-Carlo method described in part (a). I told them that they could use all the standard MatLab functions such as `rand, randn, blsprice,` etc.

I also told them that their MatLab pseudo-code does not have to be syntactically correct MatLab. All that is required is that we understand what they are trying to compute.

Here's my MatLab pseudo-code.

```
% Initialize the parameters r, sigma = σ, T, S0 = S_0,
% K1 = K_1, K2 = K_2 and L for the barrier option

% Now do a Monte Carlo simulation for the barrier option based on the
% conditioning approach described in part (a).

N = 100000; % Any reasonable value for N is fine.

Z = randn(N,1);
ST2 = S0 * exp( (r-sigma^2/2)*T/2 + sigma*sqrt(T/2)*Z );

[Call1,Put] = blsprice(ST2, K1, r, T/2, sigma);
[Call2,Put] = blsprice(ST2, K2, r, T/2, sigma);

Y = exp(-r*T/2) * ( Call1 .* (ST2 <= L) + Call2 .* (ST2 > L) );

price = mean(Y)
```

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

I asked the students to consider the transport PDE

$$\frac{\partial \phi}{\partial t} + c\frac{\partial \phi}{\partial x} = 0 \tag{24}$$

where $\phi = \phi(x,t)$, $c > 0$ and ϕ satisfies the initial condition

$$\phi(x,0) = f(x) \qquad \text{for all } x \in \mathbb{R} \tag{25}$$

One numerical method that we did not consider in class for this problem is

$$\frac{\phi_n^{m+1} - \phi_n^m}{\delta t} + c\,\frac{\phi_{n+1}^m - \phi_{n-1}^m}{2\delta x} = 0 \tag{26}$$

where $\delta t > 0$ is the stepsize in the t direction, $\delta x > 0$ is the stepsize in the x direction and $\phi_n^m \approx \phi(n\delta x, m\delta t)$. I also told them that you can rewrite (26) 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

$$\phi_n^{m+1} = \phi_n^m + \rho\left(\phi_{n+1}^m - \phi_{n-1}^m\right) \tag{27}$$

where $\rho = -c\,\dfrac{\delta t}{2\delta x}$.

(a) I asked them to determinate the order of consistency of the numerical method (26). That is, find the truncation error associated with the numerical method (26), 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.

To determine the truncation error associated with the numerical method (26), substitute the exact solution $\phi(n\delta x, m\delta t)$ for the numerical solution ϕ_n^m in (26) to get

$$\begin{aligned}
&\frac{\phi(n\delta x,(m+1)\delta t) - \phi(n\delta x, m\delta t)}{\delta t} + c\,\frac{\phi((n+1)\delta x, m\delta t) - \phi((n-1)\delta x, m\delta t)}{2\delta x} \\
&\quad = \phi_t(n\delta x, m\delta t) + \mathcal{O}(\delta t) + c\,\phi_x(n\delta x, m\delta t) + \mathcal{O}((\delta x)^2) \\
&\quad = \mathcal{O}(\delta t) + \mathcal{O}((\delta x)^2)
\end{aligned} \tag{28}$$

because $\phi_t(n\delta x, m\delta t) + c\,\phi_x(n\delta x, m\delta t) = 0$ in the middle equation of (28), since $\phi(x,t)$ satisfies the transport equation (24).

So, the numerical method (26) is first-order consistent in t and second-order consistent in x.

(b) I asked them to determinate whether the numerical method (26) is stable.
I suggested, to answer this, it might be easier to consider the equivalent form (27) of the numerical method (26).

I think this is the hardest question on the exam. The numerical method (26) is not stable, but it is a little tricky to get a good explanation for why it is not stable.

A numerical method of the type (26) is stable if there is a constant K such that, if you perturb the initial conditions for (26) by a small amount and you perturb the numerical method (26) itself by a small amount, then the difference between the solution to the original numerical method (26) and the perturbed version can be bounded by K times the maximum of the perturbations.

To be more specific, let

$$\phi_n^0 = f(n\delta x)$$
$$\hat{\phi}_n^0 = f(n\delta x) + e_n^0$$

Note ϕ_n^0 satisfies the initial condition (25) of the transport equation (24) and $\hat{\phi}_n^0$ satisfies a slightly perturbed version of the initial condition (25) (assuming all the e_n^0 are small in magnitude).

For $m \geq 0$, ϕ_n^{m+1} can be computed from (26). The corresponding perturbed version of (26) is

$$\frac{\hat{\phi}_n^{m+1} - \hat{\phi}_n^m}{\delta t} + c\,\frac{\hat{\phi}_{n+1}^m - \hat{\phi}_{n-1}^m}{2\delta x} = \epsilon_n^m \tag{29}$$

Now, if you let $e_n^m = \hat{\phi}_n^m - \phi_n^m$ and you subtract (26) from (29), you get the following equations for the errors, e_n^m:

$$\frac{e_n^{m+1} - e_n^m}{\delta t} + c\,\frac{e_{n+1}^m - e_{n-1}^m}{2\delta x} = \epsilon_n^m \tag{30}$$

A method is stable if there exists a constant K such that for all n and all $M \geq 1$ satisfying $M\delta t \leq T$

$$|e_n^M| \leq K\Big(\max_n |e_n^0| + \max_{\substack{n \\ m<M}} |\epsilon_n^m|\Big)$$

To show that the method (26) is not stable, we need to show that such a K does not exist.

In class, we discussed the *Bad example of a finite difference scheme* in section 5.2.1. of Brandimarte's textbook. To show that method was not stable, we let $e_n^0 = (-1)^n\epsilon$ for all n, where ϵ is a small positive constant, and $\epsilon_n^m = 0$ for all m and n. We were able to show that $|e_n^M|$ cannot be bounded by $K\epsilon$ for any K. (The key here is to note that, even though $M\delta t \leq T$, you can take δt arbitrary small while keeping $\rho = -c\,\frac{\delta t}{2\delta x}$ constant by decreasing δx at the same rate as δt and thereby allowing M to be arbitrarily large.

For the method (26), I don't think we can use such a simple choice of e_n^0 and ϵ_n^m, but let's use again $\epsilon_n^m = 0$ for all n and m and assume

$$e_n^0 = v_{n \bmod 4}\epsilon$$

where $n \bmod 4 = q$ if $n = 4p + q$ for an integer p and an integer $q \in \{0, 1, 2, 3\}$. Let's rewrite (30) in the form (27):

$$\mathrm{e}_n^{m+1} = \mathrm{e}_n^m + \rho\left(\mathrm{e}_{n+1}^m - \mathrm{e}_{n-1}^m\right) \tag{31}$$

where we have used $\epsilon_n^m = 0$ and $\rho = -c\,\dfrac{\delta t}{2\delta x}$.
Now consider (31) for $m = 0$:

$$\begin{aligned}\mathrm{e}_n^1 &= \mathrm{e}_n^0 + \rho\left(\mathrm{e}_{n+1}^0 - \mathrm{e}_{n-1}^0\right)\\ &= \left(v_{n \bmod 4} + \rho\left(v_{(n+1) \bmod 4} - v_{(n-1) \bmod 4}\right)\right)\epsilon\end{aligned} \tag{32}$$

Now let

$$\begin{aligned}v &= [v_0, v_1, v_2, v_3]^T\\ &= [\mathrm{e}_{4p}^0, \mathrm{e}_{4p+1}^0, \mathrm{e}_{4p+2}^0, \mathrm{e}_{4p+3}^0]^T\end{aligned}$$

for some integer p. Using the periodicity of the e_n^0, we can write the right side of the second equation in (32) in vector form as

$$\Big((I + \rho B)v\Big)\epsilon$$

where

$$B = \begin{pmatrix} 0 & 1 & 0 & -1\\ -1 & 0 & 1 & 0\\ 0 & -1 & 0 & 1\\ 1 & 0 & -1 & 0 \end{pmatrix}$$

Now, if we choose v to be an eigenvector of B with eigenvalue λ (i.e., $Bv = \lambda v$), then

$$(I + \rho B)v = (1 + \rho\lambda)v$$

So, for all n, $e_n^1 = (1 + \rho\lambda)e_n^0$. Clearly, we can continue in this way to get

$$e_n^m = (1 + \rho\lambda)^m e_n^0 = (1 + \rho\lambda)^m v_{n \bmod 4}\epsilon \tag{33}$$

So, if $|1 + \rho\lambda| > 1$, this method will be unstable, because, as noted above, we can take m arbitrarily large in (33), since we can decrease δt at the same rate as δx, thereby keeping ρ constant, allowing m to be arbitrarily large while still satisfying $m\delta t \le T$.

So, all that remains is to show that $|1 + \rho\lambda| > 1$ for some eigenvalue λ of B. To this end, note that B is skew-symmetric (i.e., $B^T = -B$). Therefore, all the eigenvalues of B are either 0 or purely complex (i.e., of the form $\lambda = i\mu$ for some nonzero real μ). All the eigenvalues of B cannot be 0, since B can be written as $B = V^*DV$ where V is a unitary matrix and D is a diagonal matrix with the eigenvalues of B on its diagonal. Hence, if all the eigenvalues of B were 0, D would be the zero matrix and $B = V^*DV$ would be the zero matrix too. Therefore, B must have a purely complex eigenvalue $\lambda = i\mu$ for some nonzero real μ. Therefore,

$$|1 + \rho\lambda| = \sqrt{1 + (\rho\mu)^2} > 1$$

So, we've shown that the numerical method (26) is unstable.

There is one thing that is unsatisfying about this proof that (26) is unstable. The vector v is complex and it seems a little artificial to allow the initial error $e_n^0 = v_{n \bmod 4}\epsilon$ to be complex.

However, if λ and v are an eigenvalue-eigenvector pair of B, then $\bar{\lambda}$ and $\bar{v}$ are also an eigenvalue-eigenvector pair of B. So, we can let $w = (v + \bar{v})$ be the real part of v and set $e_n^0 = w_{n \bmod 4}\epsilon$. With a little more work, I believe you can show that e_n^m also blows-up. However, the proof of this is a little more complex and so I'll leave it to another time.