WI3413TU
Monte Carlo Methods
Eindopdracht

Lucas de Vries 1522442

23 januari 2012

1

a

De code in appendix 1a rekent de optimale θ uit voor de simulatie met controlvariaat, en geeft daarna de ratio tussen de (sample-) varianties.

De optimale berekende theta is in dit geval 1.1102.

De simulatie geeft een prijs voor de optie van $0.1124 \pm 0.00029(s.e.)$.

De ratio tussen de sample varianties is 182.7. Als we er van uit gaan dat de control variate methode op zijn meest 2x zo veer werk kost, dan schatten we dat deze op zijn minst 91.4 keer zo snel is.

b

De code in appendix 1b shift de gebruikte random variables met een μ en weight daarna de discounted payoffs met een IS weight.

Een binary search voor $\mu \in [0, 1]$ tot op twee decimalen nauwkeurig om de variantie zo klein mogelijk te maken geeft een optimale waarde van $\mu = 0.31$.

De uiteindelijke schatting voor de optieprijs is $0.1098 \pm 0.0242(s.e.)$

De ratio tussen de variantie van deze schatting en de sample variantie van de gewone monte carlo methode is 6.4238. Aannemend dat het berekenen van de IS weights geen significante tijd kost betekend dat dus dat deze methode geschat 6.42 keer zo snel is.

c

De drie besproken methoden zijn antithetische variaten, control variaten en importance sampling.

De control variate levert de beste snelheidswinst op - dit is intuitief ook correct, want dit is de methode die het minst generiek is en dus het meest gebruik kan maken van de unieke aspecten van de dataset: de controlvariaat is gekozen op basis van het probleem zelf.

We kunnen refereren naar opdracht P21.1 voor de variantie ratio van de antithetische methode. Deze levert minder snelheidswinst dan de importance sampling methode.

De code in appendix 1c gebruikt zowel importance sampling met de in 1b uitgerekende μ, als een controlvariaat zoals gebruikt in 1a.

De schatting voor de optieprijs is $0.1118 \pm 0.00019(s.e.)$.

De verhouding tussen de variantie van deze schatting en de sample variantie van de normale monte carlo methode is 806.42.

Er van uit gaand dat de control variate methode maximaal twee keer zo veel tijd kost, en dat het berekenen van de IR weights geen significante tijd in beslag neemt kunnen we schatten dat de combinatie van deze twee methoden minimaal 403.21 keer zo snel is.

d

De code in appendix 1d is aangepast van 1a om ook de Δ de schatten voor een bepaalde h door de optieprijs twee keer te berekenen (een keer voor S_0 en een keer voor $S_0 + h$).

De onderstaande twee tabellen geven de Δ van de twee methoden voor verschillende h.

Normale Monte Carlo

h	schatting
0.1	$0.1849 \pm 0.0045(s.e.)$
0.01	$0.1752 \pm 0.0045(s.e.)$
0.001	$0.1742 \pm 0.0045(s.e.)$
0.0001	$0.1742 \pm 0.0045(s.e.)$

Control Variaat

h	schatting
0.1	$0.1776 \pm 0.0068(s.e.)$
0.01	$0.1687 \pm 0.0070(s.e.)$
0.001	$0.1682 \pm 0.0071(s.e.)$
0.0001	$0.1684 \pm 0.0071(s.e.)$

Als er geen bias bestaat verwachten we dat de waardes bij grotere h niet meer dan een paar standard errors verschillen van de waardes bij kleinere h. Dit is hier het geval: beide schattingen op $h = 0.1$ liggen niet meer dan 2.5 standard errors af van de schattingen bij $h = 0.01$ tot $h = 0.0001$. We verwachten dus dat er geen significante bias aanwezig is.

2

Voor elke individuele vraag en in alle code wordt tegelijk zowel de put als call waarde berekent, dus het antwoord op (e) zit in de voorgaande vragen.

a

Om de optieprijs te bepalen simuleren we eerst de aandeelprijs op tijd $2T/3$. Als de optie daar in the money is hebben we gelijk de payoff. Ander simuleren we door tot tijd T en verdisconteren we de payoff die daar plaatsvind.

De code hiervoor is appendix 2a.

Dit geeft de waarde van de call als $1.448 \pm 0.0171(s.e.)$ en de waarde van de put als $0.356 \pm 0.0057(s.e.)$.

b

De a mag niet afhankelijk zijn van de random trekkingen (moet constant zijn voor alle M simulaties). Monte carlo simulatie is hier niet nuttig, maar simpele iteratie zou best goed kunnen werken.

We gaan er eerst van uit dat $a_{opt} < 100$. Dit is een redelijke aanname want de kans dat de normale trekking zo uitkomt dat $|S(2T/3) - E|$ groter dan dit is is astronomisch klein.

Om a op twee decimalen uit te rekenen hoeven we alleen te itereren door alle mogelijke waarden van twee decimalen voor a tussen 0 en 100, en de waarde te kiezen die het hoogste resultaat geeft. Voor snelheidsredenen worden de matrices met de mogelijke payoff waardes eerst uitgerekend en wordt daarna voor elke a de gemiddelde payoff uitgerekend. De a met de hoogste gemiddelde (verdisconteerde) payoff wordt weergegeven.

De code hiervoor is gegeven in appendix 2b.

Het resultaat voor de call optie is $a = 2.35$, en voor de put optie $a = 1.71$.

Hoewel a exact op 2 decimalen nauwkeurig is aan de optimale a voor de gesimuleerde set paden, is deze set aandeelpaden natuurlijk niet compleet, en zal er dus altijd een onnauwkeurigheid zitten in de schatting voor a_{opt} die best meer dan 0.01 kan zijn.

c

De verdisconteerde gemiddelde payoff is ook gelijk de prijs uitgaande van die drempelstrategie, dus de maximale verdisconteerde payoffs gegeven door de code in bijlage 2b is ook gelijk de prijs bij optimale a.

Deze prijs is $1.540 \pm 0.0187(s.e.)$ voor de call en $0.402 \pm 0.0068(s.e.)$ voor de put.

Deze prijs is "correct"voor deze drempelstrategie. Aangezien deze a optimaal is kan het dus niet zo zijn dat de optie onder een andere drempelstrategie meer waard is. Er is echter niets dat aangeeft dat de drempelstrategie de optimale strategie is, dus zolang dat niet bewezen is is de prijs niet correct voor alle uitoefenstrategieën.

d

De code in bijlage 2d rekent twee keer de optiewaardes uit. Een keer voor S en een keer voor $S + h$. We gebruiken $h = 0.01$ om de Δ te schatten.

Dit geeft voor de call $\Delta = 0.682 \pm 0.052(s.e.)$ en voor de put $\Delta = -0.247 \pm 0.011(s.e.)$.

3

Antwoorden voor S=9.5 en S=6.5 tegelijk gegeven.

a

De rekentijd neemt **absoluut niet** explosief toe als je een sliding window average bijhoud over alle stappen. Je loopt door alle tijdpunten en houd de som van de afgelopen d/dT bij door elke stap de huidige waarde er bij op te tellen en de waarde van tijd t-d er van af te trekken. Door dit te delen door d/dT krijg je het gemiddelde in constante tijd. Het algoritme draait nog steeds in $O(N * M)$, ongeacht de stapgrootte (een naïeve implementatie zou elke stap het gemiddelde opnieuw berekenen, en dus in $O(N^2 * M)$ draaien, maar geen enkele programmeur zou het in zijn hoofd halen om het op die manier te programmeren).

Het detecteren van waardes hoger dan B veranderd de complexiteit niet. Er moet nog steeds door alle waardes heengelopen worden, en het controleren van $S(t) > B$ is niet goedkoper dan het simpelweg updaten van het running average. Verspilde moeite dus.

De code in bijlage 3a berekent de optieprijs gebaseerd op een lopend gemiddelde.

Het algoritme simuleert een aandeelpad, en rekent dan door een efficient lopend gemiddelde op elke stap in het pad het gemiddelde van de afgelopen d tijd uit. Als dit boven B uitkomt wordt de optiewaarde voor dit pad gezet op de payoff van $B - E$ verdisconteerd vanaf het moment dat het gemiddelde zo kwam. Als dit nooit het geval is wordt de bull spread payoff verdisconteerd van tijd T en wordt dat als optiewaarde gebruikt.

Voor $S = 9.5$ is dit $1.6234 \pm 0.0053(s.e.)$.

Voor $S = 6.5$ is dit $0.0179 \pm 0.0010(s.e.)$.

b

We verwachten dat op plaatsen waar een gewone call boven gemiddeld is, de indonesische optie ook boven gemiddeld is, en vice versa. (Hogere call betekend dat de barrier waarschijnlijk eerder wordt bereikt en de verdisconteerde payoff dus meer is). Als we de variantie willen verkleinen lijkt een europese call optie dus een goede keus voor controlvariaat.

De code in appendix 3b gebruikt de europese call als control variate om de variantie te reduceren.

Voor $S = 9.5$ geeft dit een schatting van $1.6211 \pm 0.0035(s.e.)$. De variantie is 2.2805 keer zo klein.

Voor $S = 6.5$ geeft dit een schatting van $0.0173 \pm 1.99 * 10^{-16}(s.e.)$. De variantie is $2.77 * 10^{25}$ keer zo klein.

Omdat deze optie bijna nooit de barrier zal halen is hij vrijwel identiek aan een normale europese call, en dit verklaart de enorm kleine standard error.

c

Al beantwoord in voorgaande vragen.

d

Zoals altijd: we berekenen de waarde van de optie voor $S_0 = S$ en $S_0 = S + h$.

Voor een goede schatting gebruiken we $h = 0.001$. De code voor het uitrekenen van de delta is te vinden in appendix 3d. Het gebruikt de code van appendix 3b als functie o3b(S).

Voor $S = 9.5$ geeft dit $\Delta = 0.504 \pm 0.082(s.e.)$.

Voor $S = 6.5$ geeft dit $\Delta = 0.0636 \pm 1.0 * 10^{-15}(s.e.)$.

4

a

We doen M (1e4) sets van N (1e3) simulaties. En rekenen de mean failure rate uit voor elke set. Zie de code in appendix 4a.

Dit geeft een schatting van $P_{fail} = 0.063878 \pm 0.000078(s.e.)$.

b

We zouden antithetische variaten kunnen toepassen door de antithetische van elk van de normale variabelen te gebruiken in een aparte set N simulaties, en dan de gemiddelde failure rates van de normale en de antithetische set bij elkaar op te tellen en te halveren.

Een linearisering van G rond de means van de variabelen kan gebruikt worden als controlvariaat: als de variabelen boven gemiddeld zijn zal de linearisering dat ook zijn, dus zal deze controlvariaat tot variantiereductie leiden.

Importance sampling zou ook toegepast kunnen worden door de functie k zo te maken dat de inputs allemaal iid $N(0, 1)$ zijn. In feite gebeurt dit in de code van 4a al op de output van de random number generator.

c

In de code van appendix 4b worden de antithetische waardes van elk van de variabelen gebruikt voor variantiereductie.

Dit geeft een schatting van $P_{fail} = 0.063676 \pm 0.000053(s.e.)$. De variantie is ongeveer 2.16 keer zo klein: maar een kleine verbetering in snelheid dus als we er van uit gaan dat het berekenen van beide versies twee keer zo veel tijd kost.

In de code van appendix 4c wordt de linearisering van G rond het gemiddelde van elk van de variabelen gebruikt als controlvariaat.

Deze linearisering is gelijk aan:

$$\begin{aligned} G &= G(1040, 3.5, 1.5, 0.1) + \frac{\partial G}{\partial \rho}(\rho - 1040) + \frac{\partial G}{\partial k}(k - 3.5) + \frac{\partial G}{\partial u}(u - 1.5) + \frac{\partial G}{\partial d}(d - 0.1) \\ &= 278870 - 212.625\rho - 210.6k - 294840u + 2211300d + 45497.1 \\ \mathbb{E}[G] &= 237970 \end{aligned}$$

Dit geeft een schatting van $P_{fail} = 0.080057 \pm 0.000059(s.e.)$.

De variantie is 1.02 keer zo klein, er heeft dus geen significante reductie in variantie plaatsgevonden.

Appendix 1

a

```
% Monte Carlo on an arithmetic average price Asian option
% using a geometric average price Asian as control variate

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%
S = 4; E = 4; sigma = 0.25; r = 0.03; T = 1;
Dt = 1e-2; N = T/Dt; M = 1e4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%% Geom Asian exact mean  %%%%%%%%%%%
sigsqT= sigma^2*T*(N+1)*(2*N+1)/(6*N*N);
muT = 0.5*sigsqT + (r - 0.5*sigma^2)*T*(N+1)/(2*N);

d1 = (log(S/E) + (muT + 0.5*sigsqT))/(sqrt(sigsqT));
d2 = d1 - sqrt(sigsqT);

N1 = 0.5*(1+erf(d1/sqrt(2)));
N2 = 0.5*(1+erf(d2/sqrt(2)));

geo =  exp(-r*T)*( S*exp(muT)*N1 - E*N2 );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Spath = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*randn(M,N)),2);

% Standard Monte Carlo
arithave = mean(Spath,2);
Parith = exp(-r*T)*max(arithave-E,0);  % payoffs
Pmean = mean(Parith)
Pstd = std(Parith)
Pse = Pstd/sqrt(M)
Pvar = var(Parith)
confmc = [Pmean-1.96*Pstd/sqrt(M), Pmean+1.96*Pstd/sqrt(M)]

% Control Variate
geoave = exp((1/N)*sum(log(Spath),2));
Pgeo = exp(-r*T)*max(geoave-E,0);      % geo payoffs

%Calculate theta
covar = cov(Parith, Pgeo);
theta = covar(1, 2) / var(Pgeo)

Z = Parith + (geo - Pgeo) * theta;     % control variate version
Zmean = mean(Z)
Zstd = std(Z)
Zse = Zstd/sqrt(M)
Zvar = var(Z)
```

```
confcv = [Zmean-1.96*Zstd/sqrt(M), Zmean+1.96*Zstd/sqrt(M)]

ratio=Pvar/Zvar
```

b

```
% Monte Carlo on an arithmetic average price Asian option
% using importance sampling

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%
S = 7; E = 8.5; sigma = 0.3; r = 0.02; T = 1;
Dt = 0.04; N = T/Dt; M = 1e4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Standard Monte Carlo
Spath = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*randn(M,N)),2);
arithave = mean(Spath,2);
Parith = exp(-r*T)*max(arithave-E,0);  % payoffs
Pmean = mean(Parith)
Pstd = std(Parith);
Pse = Pstd/sqrt(M)
Pvar = var(Parith)
confmc = [Pmean-1.96*Pstd/sqrt(M), Pmean+1.96*Pstd/sqrt(M)]

P = zeros(M, 1);
W = zeros(M, 1);
mu=0.31
for i = 1:M
    Z = randn(N, 1);
    Y = Z + mu; % shift random variables

    Spath = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*Y));

    P(i) = exp(-r*T)*max(mean(Spath)-E,0); % discounted payoff
    W(i) = exp(sum(mu*(0.5*mu-Y))); % weight from IS
end
V = W.*P; % modify payoffs by weights
Vmean = mean(V)
Vstd = std(V);
Vse = Vstd/sqrt(M)
Vvar = var(V)
confmc = [Vmean-1.96*Vstd/sqrt(M), Vmean+1.96*Vstd/sqrt(M)]

varRatio=Pvar/Vvar
```

c

```
% Monte Carlo on an arithmetic average price Asian option
```

```
% using both importance sampling and a control variate

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%%
S = 7; E = 8.5; sigma = 0.3; r = 0.02; T = 1;
Dt = 0.04; N = T/Dt; M = 1e4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%% Geom Asian exact mean  %%%%%%%%%%%%
sigsqT= sigma^2*T*(N+1)*(2*N+1)/(6*N*N);
muT = 0.5*sigsqT + (r - 0.5*sigma^2)*T*(N+1)/(2*N);

d1 = (log(S/E) + (muT + 0.5*sigsqT))/(sqrt(sigsqT));
d2 = d1 - sqrt(sigsqT);

N1 = 0.5*(1+erf(d1/sqrt(2)));
N2 = 0.5*(1+erf(d2/sqrt(2)));

geo =  exp(-r*T)*( S*exp(muT)*N1 - E*N2 );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Standard Monte Carlo
Spath = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*randn(M,N)),2);
arithave = mean(Spath,2);
Parith = exp(-r*T)*max(arithave-E,0);  % payoffs
Pmean = mean(Parith)
Pstd = std(Parith);
Pse = Pstd/sqrt(M)
Pvar = var(Parith)
confmc = [Pmean-1.96*Pstd/sqrt(M), Pmean+1.96*Pstd/sqrt(M)]

P = zeros(M, 1);
C = zeros(M, 1);
W = zeros(M, 1);
mu=0.31
for i = 1:M
    Z = randn(N, 1);
    Y = Z + mu; % shift random variables

    Spath = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*Y));

    geoave = exp((1/N)*sum(log(Spath)));
    C(i) = exp(-r*T)*max(geoave-E,0); % control variate payoff
    P(i) = exp(-r*T)*max(mean(Spath)-E,0); % discounted payoff
    W(i) = exp(sum(mu*(0.5*mu-Y))); % weight from IS
end

Vi = W.*P; % Regular payoffs with IR weights
Vc = W.*C; % Control variate payoffs with IR weights
```

```
%Calculate theta
covar = cov(Vi, Vc);
theta = covar(1, 2) / var(Vc)

V = Vi + (geo - Vc) * theta;

Vmean = mean(V)
Vstd = std(V);
Vse = Vstd/sqrt(M)
Vvar = var(V)
confmc = [Vmean-1.96*Vstd/sqrt(M), Vmean+1.96*Vstd/sqrt(M)]

varRatio=Pvar/Vvar
```

d

```
randn('state',1337)

h=0.0001;

%%%%%%"] = Problem and method parameters %%%%%%%%
S = 7; E = 8.5; sigma = 0.3; r = 0.02; T = 1;
Dt = 0.04; N = T/Dt; M = 1e4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%% Geom Asian exact mean  %%%%%%%%%%
sigsqT= sigma^2*T*(N+1)*(2*N+1)/(6*N*N);
muT = 0.5*sigsqT + (r - 0.5*sigma^2)*T*(N+1)/(2*N);

d1_n = (log(S/E) + (muT + 0.5*sigsqT))/(sqrt(sigsqT));
d2_n = d1_n - sqrt(sigsqT);

N1_n = 0.5*(1+erf(d1_n/sqrt(2)));
N2_n = 0.5*(1+erf(d2_n/sqrt(2)));

geo_n =  exp(-r*T)*( S*exp(muT)*N1_n - E*N2_n );

d1_h = (log((S+h)/E) + (muT + 0.5*sigsqT))/(sqrt(sigsqT));
d2_h = d1_h - sqrt(sigsqT);

N1_h = 0.5*(1+erf(d1_h/sqrt(2)));
N2_h = 0.5*(1+erf(d2_h/sqrt(2)));

geo_h =  exp(-r*T)*( (S+h)*exp(muT)*N1_h - E*N2_h );
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

R = randn(M,N);
Spath_n = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*R),2);
Spath_h = (S+h)*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*R),2);

% Standard Monte Carlo
```

```
arithave_n = mean(Spath_n,2);
Parith_n = exp(-r*T)*max(arithave_n-E,0);

arithave_h = mean(Spath_h,2);
Parith_h = exp(-r*T)*max(arithave_h-E,0);

normDelta = (Parith_h - Parith_n)/h;
normDeltamean = mean(normDelta)
normDeltase = std(normDelta)/sqrt(M)

% Control Variate
geoave_n = exp((1/N)*sum(log(Spath_n),2));
Pgeo_n = exp(-r*T)*max(geoave_n-E,0);

geoave_h = exp((1/N)*sum(log(Spath_h),2));
Pgeo_h = exp(-r*T)*max(geoave_h-E,0);

covar_n = cov(Parith_n, Pgeo_n);
theta_n = covar(1, 2) / var(Pgeo_n);

Z_n = Parith_n + (geo_n - Pgeo_n) * theta_n;

covar_h = cov(Parith_h, Pgeo_h);
theta_h = covar(1, 2) / var(Pgeo_h);

Z_h = Parith_h + (geo_h - Pgeo_h) * theta_h;

cDelta = (Z_h - Z_n)/h;
cDeltamean = mean(cDelta)
cDeltase = std(cDelta)/sqrt(M)
```

Appendix 2

a

```
% Monte Carlo for a bermuda option

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%%
S = 6; E = 5; sigma = 0.4; r = 0.03; T = 1;
Dt = 1e-2; N = T/Dt; M = 1e4; pT = 2*T/3;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

C = zeros(M, 1);
P = zeros(M, 1);
for i = 1:M
    Z = randn(N, 1);
    Spath = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*Z));

    Sfirst = Spath(floor(pT / Dt));
    Slast = Spath(end);

    % Call option
    if Sfirst - E > 0
        C(i) = (Sfirst - E) * exp(-r*pT);
    else
        C(i) = max(Slast - E, 0) * exp(-r*T);
    end

    % Put option
    if E - Sfirst > 0
        P(i) = (E - Sfirst) * exp(-r*pT);
    else
        P(i) = max(E - Slast, 0) * exp(-r*T);
    end
end

Cmean = mean(C)
Cse = std(C)/sqrt(M)

Pmean = mean(P)
Pse = std(P)/sqrt(M)
```

b

```
% Monte Carlo for a bermuda option

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%%
S = 6; E = 5; sigma = 0.4; r = 0.03; T = 1;
```

```
Dt = 1e-2; N = T/Dt; M = 1e4; pT = 2*T/3;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

C1 = zeros(M, 1);
C2 = zeros(M, 1);
P1 = zeros(M, 1);
P2 = zeros(M, 1);
for i = 1:M
    Z = randn(N, 1);
    Spath = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*Z));

    Sfirst = Spath(floor(pT / Dt));
    Slast = Spath(end);

    % Call option
    C1(i) = (Sfirst - E) * exp(-r*pT);
    C2(i) = max(Slast - E, 0) * exp(-r*T);

    % Put option
    P1(i) = (E - Sfirst) * exp(-r*pT);
    P2(i) = max(E - Slast, 0) * exp(-r*T);
end

Cmax = 0;
Ca = 0;

Pmax = 0;
Pa = 0;

for a = 0:0.01:100
    Cc = C1 >= a;
    C = Cc .* C1 + (1 - Cc) .* C2;
    Cm = mean(C);

    if Cm > Cmax
        Cmax = Cm;
        Cval = C;
        Ca = a;
    end

    Pc = P1 >= a;
    P = Pc .* P1 + (1 - Pc) .* P2;
    Pm = mean(P);

    if Pm > Pmax
        Pmax = Pm;
        Pval = P;
        Pa = a;
    end
end
```

```
Ca
Cmax
Cse=std(Cval)/sqrt(M)

Pa
Pmax
Pse=std(Pval)/sqrt(M)
```

d

```
% Monte Carlo for a bermuda option

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%
S = 6; E = 5; sigma = 0.4; r = 0.03; T = 1;
Dt = 1e-2; N = T/Dt; M = 1e4; pT = 2*T/3;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Ca = 2.35;
Pa = 1.71;
h = 0.01

C_n = zeros(M, 1);
C_h = zeros(M, 1);
P_n = zeros(M, 1);
P_h = zeros(M, 1);
for i = 1:M
    Z = randn(N, 1);

    Spath_n = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*Z));

    Sfirst_n = Spath_n(floor(pT / Dt));
    Slast_n = Spath_n(end);

    % Call option
    if Sfirst_n - E > Ca
        C_n(i) = (Sfirst_n - E) * exp(-r*pT);
    else
        C_n(i) = max(Slast_n - E, 0) * exp(-r*T);
    end

    % Put option
    if E - Sfirst_n > Pa
        P_n(i) = (E - Sfirst_n) * exp(-r*pT);
    else
        P_n(i) = max(E - Slast_n, 0) * exp(-r*T);
    end

    Spath_h = (S+h)*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*Z));
```

```
    Sfirst_h = Spath_h(floor(pT / Dt));
    Slast_h = Spath_h(end);

    % Call option
    if Sfirst_h - E > Ca
        C_h(i) = (Sfirst_h - E) * exp(-r*pT);
    else
        C_h(i) = max(Slast_h - E, 0) * exp(-r*T);
    end

    % Put option
    if E - Sfirst_h > Pa
        P_h(i) = (E - Sfirst_h) * exp(-r*pT);
    else
        P_h(i) = max(E - Slast_h, 0) * exp(-r*T);
    end
end

CDelta = (C_h - C_n)/h;
CDelta_mean = mean(CDelta)
CDelta_se = std(CDelta)/sqrt(M)

PDelta = (P_h - P_n)/h;
PDelta_mean = mean(PDelta)
PDelta_se = std(PDelta)/sqrt(M)
```

Appendix 3

a

```
% Monte Carlo for an indonesian option

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%%
B = 10; S = 6.5; E = 8; sigma = 0.1; r = 0.05; T = 1;
Dt = 1e-3; N = T/Dt; M = 1e4; d = 0.1; dS = ceil(d/Dt);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V = zeros(N, 1);
for i = 1:M
    Z = randn(N, 1);
    Spath = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*Z));

    V(i) = min(B - E, max(0, Spath(end) - E)) * exp(-r*T);

    a = 0;
    for j = 1:N
        a = a + Spath(j);
        if j > dS
            a = a - Spath(j - dS);

            if a/dS > B
                V(i) = (B - E) * exp(-r*(j*Dt));
                break
            end
        end
    end
end

Vmean = mean(V)
Vse = std(V)/sqrt(M)
```

b

```
% Monte Carlo for an indonesian option

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%%
B = 10; S = 9.5; E = 8; sigma = 0.1; r = 0.05; T = 1;
Dt = 1e-3; N = T/Dt; M = 1e4; d = 0.1; dS = ceil(d/Dt);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Black-Scholes value of european call
d1 = (log(S/E) + (r + 0.5*sigma^2)*(T))/(sigma*sqrt(T));
d2 = d1 - sigma*sqrt(T);
```

```
N1 = 0.5*(1+erf(d1/sqrt(2)));
N2 = 0.5*(1+erf(d2/sqrt(2)));
Cavg = S*N1-E*exp(-r*(T))*N2;

% Monte-Carlo simulation for indonesian option and european call
C = zeros(N, 1);
V = zeros(N, 1);
for i = 1:M
    Z = randn(N, 1);
    Spath = S*cumprod(exp((r-0.5*sigma^2)*Dt+sigma*sqrt(Dt)*Z));

    C(i) = exp(-r*T) * max(Spath(end) - E, 0);
    V(i) = min(B - E, max(0, Spath(end) - E)) * exp(-r*T);

    a = 0;
    for j = 1:N
        a = a + Spath(j);
        if j > dS
            a = a - Spath(j - dS);

            if a/dS > B
                V(i) = (B - E) * exp(-r*(j*Dt));
                break
            end
        end
    end
end

% Calculate theta
covar = cov(V, C);
theta = covar(1, 2) / var(C)

% Create new payoffs
Y = V + (Cavg - C) * theta;
Ymean = mean(Y)
Yse = std(Y)/sqrt(M)
varRatio = var(V)/var(Y)
```

d

```
% Calculate delta for indonesian option
% o3b(S) is the function that returns the (variance reduced) array of
% simulated option values when starting with S_0 = S. Code identical to
% appendix 3B

h=0.001
S=9.5

Delta = (o3b(S + h) - o3b(S))/h;
Delta_mean = mean(Delta)
Delta_se = std(Delta)/sqrt(1e4)
```

Appendix 4

a

```
% Monte Carlo for failure rate

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%%
M = 1e4; N = 1e3;
p_u = 1040; p_s = 10;
k_u = 3.5; k_s = 0.7;
u_u = 1.5; u_s = 0.45;
d_u = 0.1; d_s = 0.01;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

P = zeros(N, 1);
for i = 1:M
    p = p_s * randn(N, 1) + p_u;
    k = k_s * randn(N, 1) + k_u;
    u = u_s * randn(N, 1) + u_u;
    d = d_s * randn(N, 1) + d_u;

    S = 500000 - 2.7 * (p .* k .* u .* u) ./ d;
    P(i) = mean(S < 0);
end

Pmean = mean(P)
Pse = std(P)/sqrt(M)
Pvar = var(P)
```

b

```
% Monte Carlo for failure rate

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%%
M = 1e4; N = 1e3;
p_u = 1040; p_s = 10;
k_u = 3.5; k_s = 0.7;
u_u = 1.5; u_s = 0.45;
d_u = 0.1; d_s = 0.01;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

P = zeros(N, 1);
for i = 1:M
    p_r = randn(N, 1);
    k_r = randn(N, 1);
    u_r = randn(N, 1);
    d_r = randn(N, 1);
```

```
    p = p_s * p_r + p_u;
    k = k_s * k_r + k_u;
    u = u_s * u_r + u_u;
    d = d_s * d_r + d_u;

    p_a = p_s * -p_r + p_u;
    k_a = k_s * -k_r + k_u;
    u_a = u_s * -u_r + u_u;
    d_a = d_s * -d_r + d_u;

    L = 500000 - 2.7 * (p .* k .* u .* u) ./ d;
    A = 500000 - 2.7 * (p_a .* k_a .* u_a .* u_a) ./ d_a;
    P(i) = (mean(L < 0) + mean(A < 0)) / 2;
end

Pmean = mean(P)
Pse = std(P)/sqrt(M)
Pvar = var(P)
```

c

```
% Monte Carlo for failure rate

randn('state',1337)

%%%%%% Problem and method parameters %%%%%%%%%
M = 1e4; N = 1e3;
p_u = 1040; p_s = 10;
k_u = 3.5; k_s = 0.7;
u_u = 1.5; u_s = 0.45;
d_u = 0.1; d_s = 0.01;

Lmean = 257970;
a_c = 324367.1;
a_p = -212.625;
a_k = -210.6;
a_u = -294840;
a_d = 2211300;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V = zeros(N, 1);
C = zeros(N, 1);
for i = 1:M
    p = p_s * randn(N, 1) + p_u;
    k = k_s * randn(N, 1) + k_u;
    u = u_s * randn(N, 1) + u_u;
    d = d_s * randn(N, 1) + d_u;

    S = 500000 - 2.7 * (p .* k .* u .* u) ./ d;
    V(i) = mean(S < 0);
```

```
    L = a_c + a_p .* p + a_k .* k + a_u .* u + a_d .* d;
    C(i) = mean(L < 0);
end

% Calculate theta
covar = cov(V, C);
theta = covar(1, 2) / var(C)

P = V + (Lmean - C) * theta;
Pmean = mean(P)
Pse = std(P)/sqrt(M)
```