THE UNIVERSITY OF NEW SOUTH WALES
SCHOOL OF MATHEMATICS AND STATISTICS

MATH3311 – Mathematical Computing for Finance
MATH5335 – Computational Methods for Finance

ASSIGNMENT – Session 2, 2016

Due: 11pm Thursday 20 October (Week 12)

- Submit the appropriate files by uploading them using the "Assignments" link on the course web page on UNSW Moodle.
  - Question 1. `AQ1.m` - Matlab script file to answer Question 1 on circulant matrices.
  - Question 1. Report: `AQ1.pdf` - a pdf file which answers Question 1 (i).
  - Question 2. `Clayton_c.m` - Matlab function which evaluates the Clayton J-copula.
  - Question 2. `AQ2.m` - Matlab script file to answer Question 2 on multivariate normal densities.
- You will need to accept the plagiarism declaration before uploading files.

  See `https://student.unsw.edu.au/plagiarism`
- **ONLY PDF files will be accepted via UNSW Moodle for the report.** If you use Microsoft word, then you must save your assignment as a .pdf file before submitting it.
- Late assignments will only be accepted on documented medical or compassionate grounds.
- You may submit each file separately and you may submit the files any number of times. Only the last submission will be marked.
- Your programs will be tested on Matlab 2015b.
- In order to mark your assignment, I will run your Matlab scripts and Matlab functions with file names as specified above and check whether the numerical values your program produces are correct. **Note that it is very important to store results using the correct name.**

  For instance, if you are asked to set the seed of the random number generator by writing `rng(1);` and to create a random column vector of length 10 using `randn` and store it as column vector c, then write

```
rng(1);
c = randn(10,1);
```

If you forget to write `rng(1);` or write `c = randn(1,10)`, `c = rand(10,1)`, `C = randn(10,1)`, `c1 = randn(10,1)`, and so on, then this will be marked as wrong. This is because after running your program I will run a test program, which in this instance, may include the lines:

```
rng(1);
ctest = randn(10,1);
if exist('c') == 0
    fprintf('\n Wrong\n')
elseif isequal(c,ctest)
fprintf('\n Correct\n')
else
fprintf('\n Wrong\n')
end;
```

See also the MATLAB file Instructions.m

- Use the file name exactly as written above. If you are asked to create a MATLAB file called `AQ1.m`, then save it exactly under that name. Files saved as `aq1.m`, `Aq1`, `AQ.m`, `AQ1.txt` and so on will not run and be marked as wrong.
- Do not use any variable name which include the word `test`, i.e., do not use names like `ctest`, `c_test`, and so on;
- Suppress lengthy output using ;

1 Circulant matrices

An $n \times n$ matrix C with element $c_{i,j}$ in row i and column j, $1 \leq i, j \leq n$, is called *circulant* if $c_{i,j} = d_{i-j}$ for $1 \leq i, j \leq n$ and $d_k = d_{k-n}$ for all $k \in \{1, 2, \ldots, n-1\}$. The values $d_0, d_1, \ldots, d_{n-1}$ determine all the entries in the matrix C.

(a) Set $n = 10^3$. Set the seed of the random number generator by using `rng(1)`. Generate a random vector c of length n using the MATLAB command `randn` and store it as a column vector `c`. Display the first 5 elements of the vector c using the MATLAB command `fprintf`. The values should be

```
-6.490138e-01 1.181166e+00 -7.584533e-01 -1.109613e+00 -8.455512e-01
```

(b) Generate a circulant matrix C with first column given by c. Store the matrix using the name `C`.

(c) Is the matrix C symmetric? Check using the matrix 1-norm and an appropriate tolerance level. Use `fprintf` to print 'The matrix C is symmetric' if the matrix norm is smaller than the tolerance level, otherwise print 'The matrix C is not symmetric'.

(d) Generate a random column vector of length n using the MATLAB command `randn` and store it as column vector `b`. Solve the linear system $Cx = b$ and store the solution as column vector `x`. Print the first 5 values of x using `fprintf`. These values are

```
-1.626229e-01 9.297565e-02 7.649837e-02 -1.999898e-01 1.438400e-01
```

(e) Use the MATLAB commands `tic;` and `toc;` to check how long it takes to solve the linear system $Cx = b$.

(f) Let $y = (y_0, y_1, \ldots, y_{n-1})^\top$ be a vector of length n. The Fourier transform is given by

$$\mathcal{F}_k(y) = \widehat{y}_k = \sum_{j=0}^{n-1} y_j \mathrm{e}^{-\mathrm{i}2\pi jk/n} \quad \text{for } k = 0, 1, \ldots, n-1.$$

In MATLAB the Fourier transform can be computed using `fft`.

Let $\widehat{y} = (\widehat{y}_0, \widehat{y}_1, \ldots, \widehat{y}_{n-1})^\top$. The inverse Fourier transform is given by

$$\mathcal{F}_j^{-1}(\widehat{y}) = y_j = \frac{1}{n} \sum_{k=0}^{n-1} \widehat{y}_k \mathrm{e}^{\mathrm{i}2\pi jk/n} \quad \text{for } j = 0, 1, \ldots, n-1.$$

In MATLAB the inverse Fourier transform can be computed using `ifft`.

It is known that the linear system $Cz = b$ is equivalent to

$$\mathcal{F}_k(b) = \mathcal{F}_k(c)\mathcal{F}_k(z) \quad \text{for } k = 0, 1, \ldots, n-1, \tag{1.1}$$

where $c = (c_0, c_1, \ldots, c_{n-1})^\top$ is the first column of the circulant matrix C and $z = (z_0, z_1, \ldots, z_{n-1})^\top$ is the solution of the linear system $Cz = b$.

Use the MATLAB commands `fft` and `ifft` to solve the linear system $Cz = b$ based on (1.1). Store the Fourier transform of z using the variable name `zfft` as column vector and the solution as column vector `z`.

(g) Use the MATLAB commands `tic;` and `toc;` to check how long it takes to solve the linear system $Cz = b$.

(h) Check that your solution is correct by computing the infinity norm between the vectors z and the solution x from Part (d). Store the value of the norm in `normchk`.

1.1 Fast Fourier transform for Toeplitz matrix (pdf file)

(i) Let T be an $n \times n$ Toeplitz matrix. Explain how you can use the Fast Fourier transform to compute the matrix-vector product Ty efficiently. (Hint: Construct a $2n \times 2n$ matrix which is circulant and whose left-upper $n \times n$ submatrix is the matrix T. Then adapt the method from Part (f).)

2 Multivariate densities

2.1 Matlab function for Clayton copula density

The probability density function of the so-called Clayton J-copula is given by

$$c(\boldsymbol{u}) = \prod_{j=1}^{d} (1 + \theta(j-1)) \times \left(\prod_{j=1}^{d} u_j \right)^{-1-\theta} \times \left(1 - d + \sum_{j=1}^{d} u_j^{-\theta} \right)^{-d-1/\theta}, \tag{2.1}$$

where $\boldsymbol{u} = (u_1, u_2, \ldots, u_d) \in (0,1]^d$ and $\theta \in (0, \infty)$.

(a) Write a Matlab function `c = Clayton_c(u, theta)` which returns the value of $c(\boldsymbol{u})$ with parameter $\theta =$`theta`.

(b) If `u` is a matrix, then the output `c` should be a column vector whose jth entry is $c(\boldsymbol{u}_j)$, where $\boldsymbol{u}_j$ is the jth row of the matrix `u`.

(c) Include a check that `theta` is larger than 0. If `theta` ≤ 0, return an appropriate error message.

2.2 Estimating probabilities

In the following set $d = 2$ and $\theta = 1/2$. The goal is to estimate the integral

$$\int_0^{1/2} \int_0^{1/2} c(u_1, u_2) \, \mathrm{d}u_1 \, \mathrm{d}u_2. \tag{2.2}$$

(The exact value of this integral is $(2\sqrt{2} - 1)^{-2}$.)

(a) Use Monte Carlo simulation to estimate the integral (2.2). Set the seed of the random number generator by writing `rng(2);`. Use `rand` to generate $N = 2^{10}$ random numbers in the square $[0, 1]^2$ and then multiply these numbers with 0.5 to get random numbers in $[0, 1/2]^2$. Use these random numbers to obtain a Monte Carlo estimate of the integral (2.2). Store this estimate as `MC_estimate`.

(b) Use the quasi-Monte Carlo method based on scrambled Sobol points to estimate the integral (2.2), where the random number generator is again set to `rng(2);`. A scrambled Sobol point set `Y` in the square $[0, 1]^2$ can be generated in MATLAB by

```
Y = sobolset(d);
Y = scramble(Y,'MatousekAffineOwen');
Y = net(Y,N);
```

where `N` is the number of points and `d` is the dimension. Use again $N = 2^{10}$ points. Store the quasi-Monte Carlo estimate as `QMC_estimate`.

2.3 Math5335 Only: Multivariate normal density

The bivariate standard normal probability density function with mean $\mathbf{0}$ is given by

$$\phi(x_1, x_2) = \frac{1}{2\pi\sqrt{\det(C)}} \exp\left(-\frac{1}{2}(x_1, x_2)C^{-1}(x_1, x_2)^\top\right), \tag{2.3}$$

where C is the covariance matrix. The MATLAB function `mvnpdf` evaluates this function. Assume that C is given by

$$C = \begin{pmatrix} 1 & \varrho \\ \varrho & 1 \end{pmatrix}, \tag{2.4}$$

where $\varrho \in (-1, 1)$. The aim is to estimate the probability

$$\mu = P(X \le \boldsymbol{b}) = \int_{-\infty}^{b_1} \int_{-\infty}^{b_2} \phi(x_1, x_2) \, \mathrm{d}x_1 \, \mathrm{d}x_2,$$

where $\boldsymbol{b} = (2, -2)$ and $\varrho = -0.3$.

(a) Calculate the value μ using the MATLAB function `mvncdf` and store its value as `mu_mvncdf`.

(b) Estimate μ by first truncating the integration domain to get the integral

$$\int_{-6}^{b_1} \int_{-6}^{b_2} \phi(x_1, x_2)\, \mathrm{d}x_1\, \mathrm{d}x_2.$$

Then use a tensor product Simpson rule with 100 nodes in each coordinate (and therefore $100^2 = 10000$ nodes altogether). The tensor product rule can be generated using the MATLAB function `tensor2d.m` in the folder for Lecture 08 on Moodle. Store this value as `mu_Simpson`.

(c) Estimate μ by again truncating the integration domain as above and using a tensor product Gauss-Legendre rule with 100 nodes in each coordinate (and therefore $100^2 = 10000$ nodes altogether). The tensor product rule can be generated using the MATLAB function `tensor2d.m` in the folder for Lecture 08 on Moodle. The standard Gauss-Legendre nodes and weights can be generated using the MATLAB function `gauleg`, which is in the folder for Lecture 08 on Moodle. Store this value as `mu_Gauss`.