

HONOURS M. A. AND HONOURS B. Sc. EXAMINATION

MATHEMATICS AND STATISTICS

Paper MT5612 & MT4112 : Computing In Mathematics

May 2008

Time allowed : Two hours

Attempt ALL THREE questions

[See over

***** PLEASE NOTE *****

- In **all** questions where you are required to write code you should follow the rules you have been taught for neat, structured and indented code.
 - You should also use ‘**IMPLICIT NONE**’ where appropriate and **all** dummy arguments should be declared with the correct ‘**INTENT**’ attribute.
 - A matrix with ‘ m ’ rows and ‘ n ’ columns will be referred to as an (m, n) matrix.
-

1. Consider the following two matrix operations, horizontal concatenation and vertical concatenation, described below using the matrices \mathbf{A} , \mathbf{B} and \mathbf{C} .

- Horizontal concatenation is where each row of a matrix \mathbf{B} is appended to the end of the equivalent row of a matrix \mathbf{A} to give matrix \mathbf{C} . Therefore both \mathbf{A} and \mathbf{B} must have the same number of rows (m). If \mathbf{A} is a (m, n) matrix and \mathbf{B} is a (m, k) matrix then \mathbf{C} will be an $(m, n + k)$ matrix. For example, if

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 6 & 7 \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} 3 & 4 & 5 \\ 8 & 9 & 10 \end{pmatrix}, \quad \text{therefore, } \mathbf{C} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{pmatrix}$$

- In a similar fashion vertical concatenation is where each column of matrix \mathbf{B} is appended to the end of the equivalent column of matrix \mathbf{A} to give matrix \mathbf{C} . Therefore both \mathbf{A} and \mathbf{B} must have the same number of columns (n). If \mathbf{A} is a (m, n) matrix and \mathbf{B} is a (p, n) matrix then \mathbf{C} will be an $(m + p, n)$ matrix.

(a) Write a Fortran 90 module, called ‘cat_mod’, that contains three functions and a single subroutine described as follows.

- A function to return the result of ‘horizontal concatenation’ of two matrices making use of a **single** ‘DO’ loop.

‘FUNCTION hconcat(mat1,mat2)’ [3]

- A function to return the result of ‘vertical concatenation’ of two matrices making use of a **single** ‘DO’ loop.

‘FUNCTION vconcat(mat1,mat2)’ [3]

- A function to return an arbitrary sized matrix read in from the keyboard.

‘FUNCTION getmat(m,n)’ [1]

- A subroutine to write out an arbitrary sized matrix to the screen.

‘SUBROUTINE outmat(mat)’ [1]

(b) Write a ‘main program unit’ called ‘concat’ that will ‘USE’ the module from part (a) above to do the following.

- Declare two matrices of type ‘REAL’, ‘mat1’ a $(3, 4)$ and ‘mat2’ a $(3, 2)$.
- Read in the two matrices ‘mat1’ and ‘mat2’ from the keyboard.
- Print the two matrices out to the screen.
- Horizontally concatenate ‘mat1’ and ‘mat2’ and place the result into ‘mat3’.
- Print ‘mat3’ out to the screen. [2]

[See over

2. (a) Consider the Fortran 90 program unit ‘MODULE’ .
- Explain how you would make available to a separate ‘program unit’ entities that you have defined in a module.
 - Explain how you would selectively include only certain entities from the module that you require to be made available to a separate ‘program unit’.
 - How you would deal with any name conflicts that may occur in this process? [3]
- (b) Write a Fortran 90 program called ‘quad_complex’ that will solve a given quadratic equation using the ‘COMPLEX’ data type. Your code should;
- Contain only a main program unit without any subroutines or functions.
 - Read in, from the keyboard, the coefficients of the quadratic a , b and c .
 - Print the roots of the quadratic out to the screen.
 - Calculate and print to the screen the turning point and the nature of the turning point for the quadratic. [5]
- (c) (i) Consider the pre-multiplication of an (m, n) matrix by a row-vector containing (m) elements. For example,

$$(1 \ 2 \ 3) \cdot \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix} = (14 \ 32 \ 50 \ 68)$$

Write a Fortran 90 function, ‘FUNCTION mulvecmat(*vec*,*mat*)’, to carry out this vector-matrix multiplication. Use one dimensional arrays to store any vectors and two dimensional arrays for any matrices. [3]

- (ii) There is a Fortran 90 function called ‘MATMUL’ which performs matrix multiplication. Write your own function to do this, called ‘mulmat’.
- ‘mulmat’ should accept two arrays as its only arguments with each array being two dimensional and used to represent matrices.
 - Your ‘mulmat’ function should check to see that pre-multiplying the second argument by the first is a valid matrix operation.
 - The matrix product should be calculated making use of **three** nested ‘DO’ loops and no internal Fortran function calls. [5]
- (iii) Your ‘mulmat’ function for part (ii) could also be coded with **two** nested ‘DO’ loops instead of three by removing the innermost ‘DO’ loop and adding some new code. Explain how this can be done. Write down only the new code that would be needed (not the whole function) in place of the innermost ‘DO’ loop . Hint: you will need to use one of Fortran’s internal functions. [2]
- (iv) Can your ‘mulmat’ function, without any modification, be used to carry out the vector-matrix multiplication described above? Please explain your answer. [2]

3. An initial-value problem for a first-order, ordinary differential equation (ODE) takes the form

$$y' = dy/dx = f(x, y),$$

with $y(x_0) = y_0,$

where $f(x, y)$ is a given function and y_0 is a known value of the function y at $x = x_0$. Hamming's method for solving numerically an initial value problem is

$$\begin{aligned} py_{n+1} &= y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}), \\ y_{n+1} &= \frac{1}{8}(9y_n - y_{n-2}) + \frac{3h}{8}(pf_{n+1} + 2f_n - f_{n-1}) \\ x_{n+1} &= x_n + h; \end{aligned}$$

where y_{n+1} is the numerical approximation to y at $x = x_n + h$ and h is the step size in x . The predicted value of y_{n+1} is denoted by py_{n+1} and it then follows that $pf_{n+1} = f(x_{n+1}, py_{n+1})$. For the purpose of this question let

$$f(x, y) = y - x, \tag{1}$$

describe the ODE to be solved.

NOTE Hamming's method is **not** self starting. In order to calculate y_{n+1} the values y_{n-1} , y_{n-2} and y_{n-3} must be known (either to be used explicitly or to calculate f_{n-1} and f_{n-2}). To overcome this starting problem the first few steps can be calculated by a 'Runge-Kutta' 4th order method ('RK4'); 'RK4' has the form

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ x_{n+1} &= x_n + h, \end{aligned}$$

where

$$\begin{aligned} k_1 &= hf(x_n, y_n), \\ k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1), \\ k_3 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2), \\ k_4 &= hf(x_n + h, y_n + k_3). \end{aligned}$$

Write a Fortran 90 module that contains three procedures that together can be used to implement Hamming's method as described **overleaf** in parts (a), (b) and (c).

[See over

- (a) A Subroutine to implement a single step of the ‘RK4’ method that has the header

```
SUBROUTINE rk4(y,x,h),
```

where ‘y’ and ‘x’ enter the subroutine with the values y_n and x_n , respectively, and on exit return the updated values y_{n+1} and x_{n+1} , respectively. The argument ‘h’ is the step size h . [8]

- (b) A Subroutine to implement a **single** step of Hamming’s method. Write this subroutine using the information provided below.

- Use ‘SUBROUTINE hamming(y,x,h)’ for the subroutine header.
- The variables ‘y’ and ‘x’ enter the subroutine with the values y_n and x_n , respectively, and on exit return the updated values y_{n+1} and x_{n+1} , respectively.
- The argument ‘h’ is the step size h . [10]
- Initially there will not be enough starting values to use Hamming’s method. The ‘SUBROUTINE hamming’ should recognise this and make the necessary call to the ‘SUBROUTINE rk4’. It should return results from ‘rk4’ until there are enough starting values to use Hamming’s method.

- (c) A Function to calculate $f(x_n, y_n) = y_n - x_n$ that has the header

```
FUNCTION ode(x,y),
```

where ‘x’ and ‘y’ are supplied as appropriate. [2]