1. The 0'th order Bessel function can be computed by adding terms of the power series given as formula 9.1.10 in Abramowitz & Stegun (p. 360).

a. Write a program to compute the series efficiently by directly adding the series, in the range $0 \le x \le 3$. The resultshould have absolute precision of 10^{-8}.

b. Use Chebyshev approximation to find a polynomial fit on the same range that provides the same absolute precision. (Hint: use the table 22.3 found on p. 795 of A&S.)

Solution:

A program to evaluate Bessel functions by summing the series is shown on the next page.

In the range $0 \le x \le 3$ the 9th term of the series

$$J_0(x) = \sum_{n=0}^{\infty} \frac{\left(-x^2/4\right)^n}{(n!)^2}$$

is bounded by 1.12×10^{-8}. If we wish, therefore, we can approximate

$$\xi^9 \overset{df}{=} \left(x^2/9\right)^9$$

by

$$\frac{1}{256}\left[126T_1(\xi) + 84T_3(\xi) + 36T_5(\xi) + 9T_7(\xi)\right]$$

and ξ^8 by

$$\frac{1}{128}\left[35T_0(\xi) + 56T_2(\xi) + 28T_4(\xi) + 8T_6(\xi)\right].$$

The errors from dropping T_8 and T_9 are respectively bounded by

$$\frac{(9/4)^8}{128 \times (8!)^2} \approx 3 \times 10^{-9}$$

and

$$\frac{(9/4)^9}{256 \times (9!)^2} \approx 4 \times 10^{-11}.$$

We cannot economize further since Chebyshev-approximating the term in ξ^7 gives an error of about 2×10^{-7}. Rearranging terms we find

$$\begin{aligned} J_0 \approx\ & 0.99999999684 + 2.25\xi \\ & + 1.26562510101\xi^2 \\ & - 0.3164062500\xi^3 \\ & + 0.04449412386\xi^4 \\ & - 0.0040045166\xi^5 \\ & + 0.00025109036\xi^6 \\ & - 0.00001149255\xi^7 \end{aligned}$$

Some tests are:

```
0.0e0 j0 f. 1.0000000  ok
1.0e0 j0 f. .76519768  ok
2.0e0 j0 f. .22389078  ok
2.4e0 j0 f. .00250768  ok
2.5e0 j0 f. -.04838378  ok
3.0e0 j0 f. -.26005195  ok
```

```
\ 0'th order Bessel function by series evaluation
\   precision set to 1.E-8

\ ------------------------------------------------
\     (c) Copyright 2001  Julian V. Noble.          \
\       Permission is granted by the author to      \
\       use this software for any application pro-  \
\       vided this copyright notice is preserved.   \
\ ------------------------------------------------

\ This is an ANS Forth program requiring the
\   FLOAT wordset.
\
\ Environmental dependences:
\       Assumes independent floating point stack

MARKER -bes0

include ftran201.f

FVARIABLE  xx
FVARIABLE  sum
FVARIABLE  term

: LessThan  ( f: a b --)  ( -- true if a < b)
    F<  ;

: done?   f" LessThan(abs(term), 1e-10)"   ;

: J0    ( f: x -- J0)
    1 LOCALS| n |           \ initialize index
    f" xx="                 \ store variable in xx

    f" xx = - (xx/2)^2"     \ initialize
    f" sum = 1"
    f" term = xx"

    BEGIN                   \ loop until |term| < 1e-10
        n 1+  TO n              \ increment index
        f" sum = sum + term"    \ update sum
        f" term * xx"           \ update term
           n  S>F  f^2 F/
           f" term="            \ and save it

        ( cr  n .  f" term" f.  f" sum" f. ) \ debugging code

    done?  UNTIL

    f" sum + term"
;
```

2. The subroutine at the right is intended to solve linear equations with tri-diagonal matrices. It has neither comments nor documentation. Without looking at the "Numerical Recipes" book, analyze the subroutine and provide

   a) a flow chart;

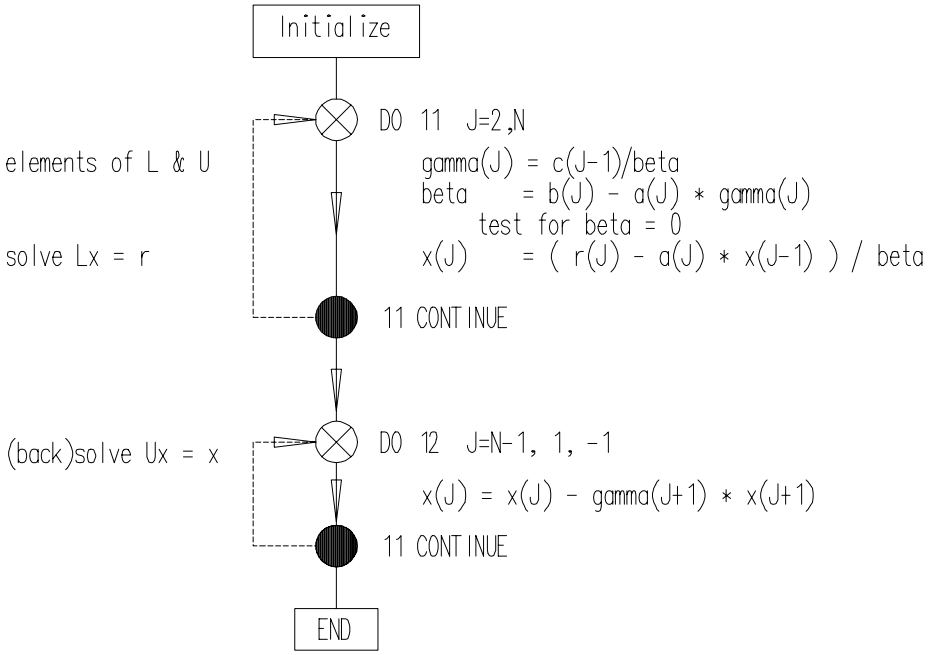
   b) appropriate comments and documentation in the program;

   c) a working and tested version in your own favorite programming language (different from the above).

```
      SUBROUTINE TRIDAG(A,B,C,R,U,N)
      PARAMETER (NMAX=100)
      DIMENSION GAM(NMAX),A(N),B(N),C(N),R(N),U(N)
      IF(B(1).EQ.0.)PAUSE  ' Initialize
      BET=B(1)
      U(1)=R(1)/BET
      DO 11 J=2,N          ' get elts of L & U
        GAM(J)=C(J-1)/BET
        BET=B(J)-A(J)*GAM(J)
        IF(BET.EQ.0.)PAUSE
        U(J)=(R(J)-A(J)*U(J-1))/BET  ' solve Lx=r
   11 CONTINUE
      DO 12 J=N-1,1,-1               ' solve Ux=x
        U(J)=U(J)-GAM(J+1)*U(J+1)
   12 CONTINUE
      RETURN
      END
```

Solution:

Comments and flow chart as shown; program on the Web page.



3. Write a brief program to solve linear systems of equations using the pivotal elimination method (that is, the sequence TRIANGULARIZE $\longrightarrow$ BACKSOLVE). Use it to solve the test case

$$\begin{pmatrix} 3 & 2 & 4 & 6 \\ -5 & 5 & 7 & -6 \\ 0 & 18 & 0 & -8 \\ -9 & 9 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 5 \\ -7 \end{pmatrix}$$

to 6 significant figures of precision. If you wish you may use Maple® or Mathematica® or Matlab® to check your work—or else multiply the matrix by the solution and see whether you obtain the inhomogeneous term.

Solution:

For a program in Forth, see the section "Forth linear equations, etc." Runs fine under Win32Forth and GForth. (ANS Forth systems.) The numerical results are

```
a{{ v{ }}solve
x(0 )= .795035
x(1 )= .025639
x(2 )= .491872
x(3 )= -.567312  ok
a{{ Iperm{ .M
1.00000 -1.00000 -.111111 -.111111
.000000 1.00000 .000000 -.444444
.000000 .000000 1.00000 -1.01724
.000000 .000000 .000000 1.00000  ok
```

4. Write a short, efficient program to perform bit-reversal of numbers from 0 to 2^{k-1} where $k=5, 6, \ldots, 10$. Make sure you also create a tool to test your answer by displaying the input and output in binary format, in a field of k digits, including the leading 0's. (That is, if the standard integer is 32 bits, there should be $32-k$ leading 0's.)

Solution:

Here are 3 example programs:

Fortran:

```
      INTEGER FUNCTION STIB(N,K)
C
C     Return the bit-reversed integer corresponding to the
C     K least significant (rightmost) bits of N.
C     Algorithm:
C     Shift STIB one bit to the left and add the LSB of N.
C     Shift N one bit to the right. Repeat K times.
C
C                                     initialize result
      STIB=0
C                                     repeat K times
C
C                                          initialize result
      STIB=0
C                                          repeat K times

      DO 10 I=1,K
C                                          right-shift 1 bit
           NP = N/2
C                                          left-shift old result, add LSB of N
           STIB=N+2*(STIB-NP)
C                                          replace N by right-shifted version
           N=NP
10    CONTINUE
      RETURN
      END
```

C:

```
int stib( int n, int k ) // reverse bit order of #
{    int i;
     int j=0;                  // initialize answer
     for( i=1; i<=k; i++ )     // loop k times
     {    j=2*j+(n && 1);      // left-shift result and add 1's bit of n
          n=n/2;               // shift n one place to right
     }
     return j;
}
```

Forth:

```
: stib  ( n w -- n') \ reverse bit order of n in [0,2^w-1]
    LOCALS| w n |        \ def local variable names
    0       ( -- 0)      \ n' = 0 is top of stack
    w 0 DO  ( n')        \ loop w times
            2*           \ left-shift n' 1 place
            n 1 AND      \ get 1's bit of n
            +            \ add to n'
            n  2/  TO n  \ right-shift n -- equiv to n = n/2
       LOOP              \ end loop
;
```

(see next page for display routine)

Display subroutine in Forth:

```
\ display an integer in binary form, with leading 0's

\ -------------------------------------------------
\     (c) Copyright 1999  Julian V. Noble.          \
\       Permission is granted by the author to      \
\       use this software for any application pro-  \
\       vided this copyright notice is preserved.   \
\ -------------------------------------------------

\ This is an ANS Forth program requiring the CORE wordset

MARKER  -bdisp

BL PARSE (D.) DUP PAD C! PAD CHAR+ SWAP CHARS MOVE PAD FIND
 NIP 0=
[IF]   : (D.)    TUCK DABS <# #S ROT SIGN #> ;    [THEN]

: ZEROS ( w --)
    0 ?DO [CHAR] 0 EMIT LOOP  ;

: 0.R ( n w --)     \ display a number right-justified in
                    \ a field of width w, with leading 0's
    >R  0 (D.) R>
    OVER - ZEROS    TYPE ;

: 0.B ( n w --)
    BINARY
    CR  0.R
    DECIMAL ;

\ Example
\
\ FLOAD C:\WIN32FOR\JVN_PROGS\B_DISP.F  ok
\ FLOAD C:\WIN32FOR\JVN_PROGS\STIB1.F  ok
\ 17 14 2dup cr 0.b  tuck  stib  swap cr 0.b
\
\ 00000000010001
\
\ 10001000000000 ok
```