
# Computational Statistics


## Programming – week #5

Dr. Mark Bourassa

Topics:
Review of Variable Types
Review of Functions

Remember to turn in AS4_your_last_name.f90, Best_fit_your_last_name.f90, correlate_your_last_name.f90, AND Gaussian_your_last_name.f90

# Review of Programming Basics: Program Flow

- Good programs – meaning ones that are easy to read – flow from the top down.
  - There will be some looping structure, but the typical flow is from the top of the file towards the bottom.
- Declare variables
- Initialize variables
- Process data
  - Make sure each variable used on the right side of an equation has a value prior to use in the equation.
  - If data are being read in and NOT stored, then calculations based directly on that data must be done in the read loop.
    - E.g., sums
    - Secondary calculations (based on the sums) should be done after the read loop
- End with an END

# Review of Programming Basics: Declaration of Variables

- REAL vs. INTEGER
  - Is the decimal important? If it is, then use REAL. If not use INTEGER.
  - Exception: if you want to do a lot of math, where the integer is treated as real number, then you might was well declare it as a REAL.
- ARRAY vs. Non_ARRAY
  - Do you want the program to store in (easily accessible) memory a series of data. If yes, then an ARRAY is probably a good idea.
  - For huge arrays you might want to use dynamic memory allocation (we will cover this in another class)
- LOGICAL vs. INTEGER
  - Are there only two allowed values, and are they used qualitatively? Then LOGICAL would be good.

# Procedures (AKA Subprograms)

- There are two types of procedures: functions and subroutines.
- In general, procedures have zero or more arguments
  - E.g., subprogram1( x1, x2, x3, x4)
  - The variables x1, x2, x3, and x4 are arguments
  - Each procedure must end with the END command
  - Each procedure will cause the program to stop if the program reaches any END command
  - If procedure is not suppose to cause the program to stop, then the program must reach a RETURN command prior to the END.
- Subroutines change the value of one or more of the arguments.
  - Executed with a CALL command. E.g., CALL MEAN( x, ave )
- Functions do not alter any arguments, but return a value.
  - Example: y = MEAN( x )
- There are several ways to declare procedures.
  - Procedures must be declared in any program that uses them.

# FORTRAN Tidbit

- There are mathematical functions that are likely to be used. There are also some that seem needed, but aren't available for good reason.
- log(x) natural log of x
- log10(x) log to the base 10 of x
- exp(x) exponent of x
- ~~exp10(x)~~ 10.0**x
- cos(x), sin(x) cosine and sine of x (x is in radians)
- acos(x), asin(x) inverse cosine and inverse sine (arccosine, arcsine)
- atan inverse tangent, radians $-\pi/2$ to $\pi/2$
- atan2(y,x) inverse tangent based on vector components y & x. radians $-\pi < \text{atan2}(y,x) < \pi$
- mod(a,p) modulus: remainder of a / p
  a – int(a / p) * p
- sqrt(x) squareroot of x

# FORTRAN90 Example of a Function

- Consider a subroutine to calculate standard deviation.

```
FUNCTION STANDEV( x, n )
! n   the number of values in array x
! x   array of values for which the standard deviation will be determined
INTEGER :: i_data, n
REAL :: standev, sum_x, sum_x_sqd
REAL, dimension( n ) :: x
sum_x = 0.0
sum_x_sqd = 0.0
DO i_data = 1, n
   sum_x = sum_x + x(i_data)
   sum_x_sqd = sum_x_sqd + x(i_data) ** 2
ENDDO
standev = SQRT( sum_x_sqd – sum_x ** 2 / REAL(n) ) / REAL( n – 1)
RETURN
END FUNCTION STANDEV
```

# Review of Programming Basics: Where to Assign Values to Variables

- You must always assign a value to a variable prior to using that variable on the right hand side of and equation, or in a IF statement.
- Input (not output) variables to subroutines should have values assigned prior to calling the subroutine.
  - Otherwise, even a correctly coded subroutine will give you weird output.
  - Computer speak is ‘garbage in, garbage out.’

# Floating Point Exceptions

- Floating point exceptions mean that the code has calculated a number that is too big (either positive or negative).
- Common causes are
  - dividing by zero
  - Taking the log of zero
  - Dividing by a small number.

# Assignment #4 (Part 1 of 2): Create a Linear Best Fit Subroutine & See How Random Error Modifies Your Results

- The big picture.
  - We will see how linear best fit statistics change when Gaussianly distributed random noise is added to the data.
  - We are interested in this problem because most statistical tests assume random errors in the data will have negligible influence on the statistics.
  - In reality, there are often substantial random errors in observations, and these can influence the outcome of linear best fits.
  - In part 2 of this assignment we will visualize the data and the fit.
- Note that the following instructions are incomplete. They provide all the main steps, but some of the details are a little vague.
- Change directories to your met3220 directory.

# Assignment #4 Part 1: Main Program

- A) Copy your main program from the previous assignment to the current assignment (changing AS3 to AS4)
  - cp as3_your_last_name.f90 as4_your_last_name.f90
  - Get this right because you don't want to delete as3!
- B) Delete the opening and reading of data from the third city (you should be reading from the files 80369user.txt and 80478user.txt. Delete code working on cases 2 through 7.
- C) Delete junk2 from the declarations, and replace 'A12' in the read statements with '12X'.
- D) After the first call of the correlate subroutine, add a call of the best fit subroutine.
- E) Add the following print statements (or make a formatted write with the same information).

```
PRINT*, 'Case 0:'
PRINT*, '       x_std  =  ',  standev_x, '  y_std  =  ', standev_y, '  r  =  ', r
PRINT*, '       Slope  =  ',  slope, '  +/-  ', sig_slope
PRINT*, '       Y-int  =  ',  y_int,  '  +/-  ', sig_yint
PRINT*, ' '
```

# Assignment #4 Part 1: Main Program

- F) Add a call to the subroutine random_seed, so that each student's program will generate a different series of random numbers.

```
call random_seed()
```

- G) Code a DO loop within a DO loop, as follows

```
DO i_wt1 = 1, n_wt1
  DO i_wt2 = 1, n_wt2
    wt1 = 0.25 * REAL( i_wt1 )
    wt2 = 0.25 * REAL( i_wt2 )
  ! more code will be inserted here
  ENDDO
ENDDO
```

  - These loops will be used to change the amount of noise that we will add to the temperature data.

# Assignment #4 Part 1: Main Program

- H) Add code at the location mentioned in (G). Add one loop that loops through all the days of temperature data. Within this loop do the following.
- i) get a random number
- ii) Use that random number to get a z-value (number of standard deviations from the mean). Add the z-value times 'the weight (wt1 or wt2) for that data set' times 'the standard deviation of the original data to the data'. For example,

  tmin1_mod(i_pts) = tmin1(i_pts) + wt1 * standev_tmin1 * gaussian(rand_out)

  - Note that we are storing the 'messed up' data in a new array. We do this because we want to be able to keep using the original data.
- iii) repeat (i) and (ii) for the second set of temperature data.
- iv) call correlate and best fit subroutines, with the noisy data as input
- v) Print the data to screen (or use a formatted write) as follows

```
PRINT*, 'Case ', (i_wt1-1)*n_wt2 + i_wt2, ':  wt1 = ', wt1, '  wt2 = ', wt2
PRINT*, '       x_std = ', standev_x,' y_std = ', standev_y,'r = ',r
PRINT*, '       Slope = ', slope ,' +/- ', sig_slope
PRINT*, '       Y-int = ', y_int, '+/- ', sig_yint
print*, ' '
```

# Assignment #4: Linear Best Fit Subroutine

- Make a subroutine to calculate the fitting parameters (slope and y-intercept) for a linear fit to the data.
- Note that this best fit is based on minimizing the square of the differences between the line and the data.
  - These differences are measured in y-direction.
  - You get different answers if the lines are measured perpendicular to the line.
- The arguments for your subroutine will be

| | | |
|---|---|---|
| x | an array of values, with n elements | INPUT |
| y | an array of values, with n elements | INPUT |
| n | the number of array elements | INPUT |
| yint | the y-intercept (b in y=mx + b) | OUTPUT |
| slope | the best fit slope (m in y=mx + b) | OUTPUT |
| sig_yint | uncertainty in y-intercept | OUTPUT |
| sig_slope | uncertainty in slope | OUTPUT |

- Use your correlate subroutine for guidance on program structure.

# Assignment #4
# Subroutine Calculations

- The calculations for the y-intercept and slope are:

$$\text{y\_int} = \frac{\left(\sum_i^n x_i^2\right)\left(\sum_i^n y_i\right) - \left(\sum_i^n x_i\right)\left(\sum_i^n x_i y_i\right)}{\Delta}$$

$$\text{slope} = \frac{n\left(\sum_i^n x_i y_i\right) - \left(\sum_i^n x_i\right)\left(\sum_i^n y_i\right)}{\Delta}$$

$$\Delta = n\left(\sum_i^n x_i^2\right) - \left(\sum_i^n x_i\right)^2$$

# Assignment #4
# Subroutine Uncertainty Calculations

- The uncertainty in y is

$$\sigma_y^2 = \frac{1}{n-2}\sum_i^n \left(y_i - mx_i - b\right)^2$$

  - Where *m* is the slope, and *b* is the y-intercept.
- The uncertainty in the slope and y-intercept are

$$\sigma_{slope}^2 = n\sigma_y^2 / \Delta$$

$$\sigma_{y-\text{int}}^2 = \sigma_y^2 \left(\sum_{i=1}^n x_i^2\right) / \Delta$$

# Assignment #4 Part 1: Best Fit Subroutine

- Copy correlate_your_last_name.f90 to best_fit_your_last_name.f90. You can use most of this code in your best fit subroutine.
- Change the program name and arguments to best_fit( x, y, n_pts, y_int, slope, sig_yint, sig_slope ), or change the argument names to be more to your liking.
- B) Updated declarations, the last for arguments can have decimal values. Delete the calculation of r.
- C) After the current DO loop, calculate the slope and y-intercept (as shown on the previous pages).
- D) Use the calculated slope and y-intercept to estimate the uncertainty in the estimates of the slope and y-intercept (as shown on the previous page).

# Assignment #4 Part 1: Gaussian function

- A) Copy the Gaussian2.f90 function to your directory.
  - If your window is for the metlab server, the Gaussian2.f90 file is located in the directory /u/b/users/met3220-01/
  - Alternatively, if your window is for the local terminal, the file is located in the directory /net/b/met3220-01/
- B) You must declare the two variables used in the program
- C) The current version of the code outputs only positive numbers of standard deviations from the mean. You must modify the code (by adding an IF statement), to make half the output negative. There are several ways this could be done, but your answer must be consistent with the value of the cumulative probability.
- Recall the syntax of IF statements (two examples)

```
IF (condition) statement
```

```
IF (condition) THEN
   statement
ELSE
   different statement
ENDIF
```

# Assignment #4 Part 1: Putting it Together

- Attempt to compile the program

  f90 AS4_your_last_name.f90 correlate_your_last_name.f90 best_fit_your_last_name.f90 Gaussian2_your_last_name.f90 –o as4

  - Type all the above on one line
- Run the program
- Comment the main steps in the code – Explain what it is doing and what it means statistically. In other words answer the following questions. Does random noise influence the results of a correlation? How do the quality of the results depend on the amount of noise? This is the bulk of your grade. Discuss the meaning or your correlations. Include this discussion in your comments at the top of the code.
- Turn in the working source codes (AS4_your_last_name.f90 & correlate_your_last_name.f90 & best_fit_your_last_name & Gaussian2_your_last_name.f90) to the TA. I suggest using the digital dropbox.
- Due date: **Wednesday**, Feb. 21, before 5:00PM (local time). You will want to have the above coding done (or close to it) by Feb. 14th, so that you can work on part 2 of this lab.