COMP 1012 Fall 2016 Assignment 2

Due Date: Friday, October 14, 2016, 11:59 PM

Material Covered

- output using `print` and formatting
- calculations using integer and floating point values
- functions from `math`
- multi-precision computation

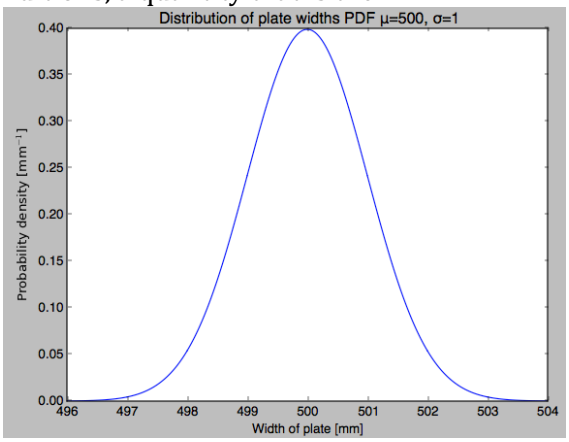
Notes:

- When programming, follow the posted programming standards to avoid losing marks. A program called `CheckStandardsV2.py` is provided to enable you to check that you are following standards.
- Name your script file as follows: `<LastName><FirstName>A2Q1.py.` For example, `LiJaneA2Q1.py` is a valid name for a student named Jane Li. If you wish to add a version number to your file, you may add it to the end of the file name. For example, `SmithRobA2Q1V2.py` is a valid name for Rob Smith's script file.
- Submit output in a similarly-name output file: e.g., `<LastName><FirstName>A2Q1Output.txt.` To generate this file, open a new empty tab in Spyder and save it under the name above. Then copy and paste the output of your program from the console window to this file, and save it.
- You must complete the checklist for a ***Blanket Honesty Declaration*** in UM Learn to have your assignment counted. This one honesty declaration applies to all assignments in COMP 1012.
- To submit the assignment follow the instructions on the course website carefully (link: http://www.cs.umanitoba.ca/~comp1012/Documents/Handin_UMLearn.pdf). You will upload both script file and output via a dropbox on the course website. There will be a period of several days before the due date when you can submit your assignment. ***Do not be late!*** If you try to submit your assignment within 48 hours ***after*** the deadline, it will be accepted, but you will receive a penalty (of 25% per day late).

Question 1—Gaussian Probability

Description

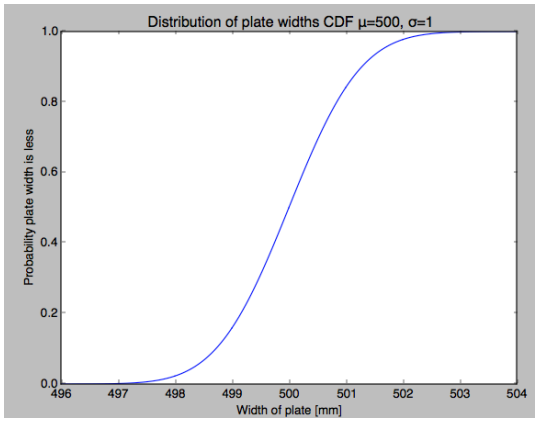
The normal or Gaussian probability distribution describes many common phenomena[1]. For example, the height of male or of female Canadians can be approximated by a normal distribution. If you as an engineer request a contractor to produce a number of metal plates of specified dimensions, deviations from those dimensions will also likely follow a normal distribution. The Central Limit Theorem of statistics says that under very general conditions, a quantity that is the sum of numerous effects (e.g., genetic and environmental factors for height) will tend to follow a normal distribution.



The *probability density function* (pdf) for a normal distribution with mean μ (Greek letter mu) and standard deviation σ (sigma) is given by this equation:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (1)$$

The plot to the right shows the pdf for the width of a plate that is normally distributed with mean 500 mm and standard deviation of 1 mm. This plot is commonly called a "bell-shaped curve". The plot tells us the proportion of plates between two widths using the area under the curve. These can be calculated, for instance, using μ, σ and standard normal tables[2].



The *cumulative distribution function* (cdf) for a normal distribution is given by the following formula:

$$F(x) = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right) \quad (2)$$

(The *error function* erf(x) is a standard mathematical function, like log, exp or sin.) The plot to the right shows the cdf. You can read from the plot that the probability is 0.5 that the plate width is less than 500 mm, and almost 1 that the plate width is less than 503 mm.

Fortunately, Python's math library contains erf, so it can be evaluated like log, exp or sin. But what if it didn't? How would we evaluate erf then? The following infinite series shows one way that we could evaluate it for $z = (x-\mu)/(\sigma\sqrt{2})$.[3]

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n z^{2n+1}}{n!(2n+1)} = \frac{2}{\sqrt{\pi}}\left(z - \frac{z^3}{3} + \frac{z^5}{10} - \frac{z^7}{42} + \frac{z^9}{216} - \cdots\right) \quad (3)$$

---

[1] http://www.usablestats.com/lessons/normal

[2] https://en.wikipedia.org/wiki/Standard_normal_table

[3] http://en.wikipedia.org/wiki/Error_function

Equation (3) is mathematically correct, but as we shall see, it may be difficult to evaluate numerically.

Details

Write a Python script that calculates pdf and cdf values for a normal distribution with mean value μ and standard deviation σ. Use μ = 900 mm and σ = 1 mm in your calculations, but the numbers 900 and 1 should show up only once in your script, where you define identifiers for μ and σ.

First, your script should evaluate pdf and cdf values for integer values of *x* from 890 to 910 inclusive using functions from the `math` library. NOTE: you must also define identifiers for these two values, but do NOT use the numerical values for μ or σ directly when you do so. Instead, define the pdf and cdf values with an expression involving μ.

Then evaluate just the cdf values again using the series expansion in Equation (3). In evaluating each cdf value, continue to add terms to the series until the absolute value of the term you would add next is less than 10^{-20} in magnitude.

If you examine the sample output below you (for μ = 500 mm and σ = 1 mm) will see that values for the cdf obtained using the series become more and more inaccurate as the *x* value gets further from the mean, until the value quoted is just nonsense. For instance, the cdf value for *x* = 490 is the probability that a random normal value is 10 standard deviations below the mean value. It should be extremely small, near 0, but the series gives a result of 3881.5, which is completely wrong, since probabilities must lie between 0 and 1. This result is *not* due to a mistake in the code; the equation is correct and the code is correct, but numerically the cdf value cannot be approximated accurately by this method using 16-digit floating point numbers.

One way to deal with this problem would be to use a better (more numerically stable) algorithm. That's what the library function for erf does. (See the Wikipedia article on the error function, cited on the previous page, for better algorithms.) A different way to deal with the problem is to sum the infinite series using integer arithmetic, since integers have unlimited precision in Python. They are not restricted to 16 digits. Notice that $y = (x-\mu)/\sigma = z\sqrt{2}$ is an integer for the values we have chosen to evaluate. It gives the number of standard deviations above or below the mean. The following equation in *y* has the property in the last line that the sum in the bracket can be carried out using 50-digit arithmetic by using only integer calculations, if *y* is an integer.

$$
\begin{aligned}
\operatorname{erf}(z) = \operatorname{erf}\left(\frac{y}{\sqrt{2}}\right) &= \frac{2}{\sqrt{\pi}}\left(\frac{y}{\sqrt{2}} - \frac{y^3}{3\cdot 2\sqrt{2}} + \frac{y^5}{10\cdot 4\sqrt{2}} - \frac{y^7}{42\cdot 8\sqrt{2}} + \frac{y^9}{216\cdot 16\sqrt{2}} - \cdots\right) \\
&= \frac{2}{\sqrt{2\pi}}\left(y - \frac{y^3}{3\cdot 2} + \frac{y^5}{10\cdot 4} - \frac{y^7}{42\cdot 8} + \frac{y^9}{216\cdot 16} - \cdots\right) \\
&= \frac{2\cdot 10^{-50}}{\sqrt{2\pi}}\left(10^{50}y - \frac{10^{50}y^3}{3\cdot 2} + \frac{10^{50}y^5}{10\cdot 4} - \frac{10^{50}y^7}{42\cdot 8} + \frac{10^{50}y^9}{216\cdot 16} - \cdots\right)
\end{aligned}
\tag{4}
$$

You will redo the series calculations using this integer series to produce the more accurate results similar to those shown in the third part of the sample output.

Sample Output

The output below shows the results for integer values of *x* from 490 to 500. Your output, will use different values of μ and σ and uses integer values of x from 890 to 910, will have twice as many lines, but otherwise it should be as similar to the sample output as possible.

```
DISTRIBUTION OF A PLATE WIDTH: μ = 500 mm, σ = 1 mm

Using functions from the math library ...
 x         f(x)           F(x)
490     7.6946e-23     0.0000e+00
491     1.0280e-18     0.0000e+00
492     5.0523e-15     6.1062e-16
493     9.1347e-12     1.2798e-12
494     6.0759e-09     9.8659e-10
495     1.4867e-06     2.8665e-07
496     1.3383e-04     3.1671e-05
497     4.4318e-03     1.3499e-03
498     5.3991e-02     2.2750e-02
499     2.4197e-01     1.5866e-01
500     3.9894e-01     5.0000e-01

Using a series to evaluate erf(z) ...
 x          F(x)      Number of terms
490     3.8815e+03        171
491    -1.4276e-01        145
492     2.9402e-05        121
493     8.2477e-09         99
494     9.1897e-10         80
495     2.8665e-07         64
496     3.1671e-05         49
497     1.3499e-03         37
498     2.2750e-02         26
499     1.5866e-01         17
500     5.0000e-01          0

Using an accurate integer series to evaluate erf(z) ...
 x          F(x)      Number of terms
490     0.0000e+00        223
491     0.0000e+00        193
492     6.1062e-16        167
493     1.2798e-12        143
494     9.8659e-10        121
495     2.8665e-07        101
496     3.1671e-05         83
497     1.3499e-03         65
498     2.2750e-02         51
499     1.5866e-01         35
500     5.0000e-01          0

Programmed by The Instructors
Date: Thu Sep 22 18:42:45 2016
End of processing
```