Gaussian Cheat Sheet with Python

Steve Witham ess doubleyou at tiac notthis dot net 2020-10-20

You know that Gaussian and normal distributions are the same thing. You know the difference between probability density (PDF), its integral (CDF), one minus the CDF, and the inverse of the CDF, and what you might want each of those for. You need to know the scale constants and what to import in Python, simply organized in one place in a uniform format. If so, you have come to the right place.

Please let me know if you find mistakes here.

$$\begin{aligned} \text{PDF} &= \text{Probability Density Function,} \\ \text{CDF} &= \text{Cumulative Distribution Function} \\ &\qquad \text{(see what they did with the D there),} \\ 1 - \text{CDF} &= \text{the complement, and} \\ \text{CDF}^{-1} &= \text{the inverse.} \end{aligned}$$

But then different communities, personality types, and/or periods of history have different ideas of what default scale and offset to use on each of these, and what to call them all.

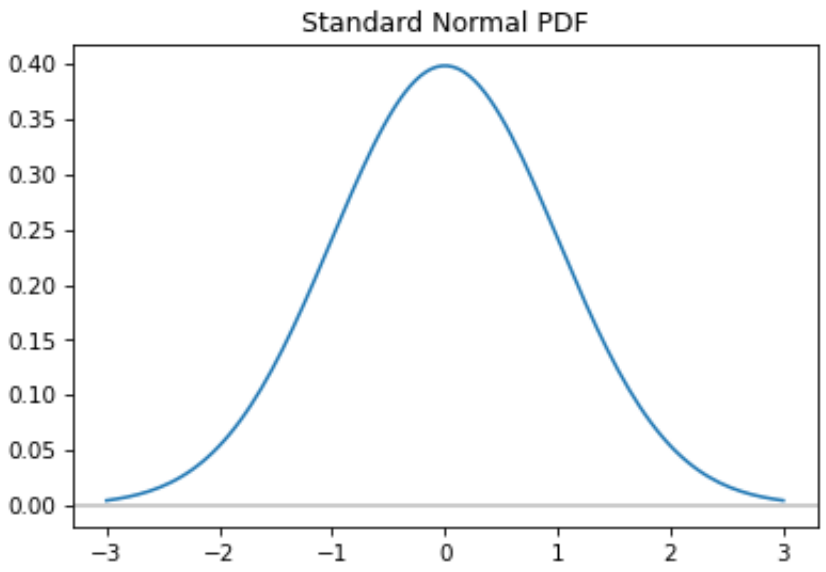
With Python, which standard library you get the related functions from is random, and which of the competing scaling/ offset versions is available is random. The `scipy` library gives (systematic, but) new dotted names to all the functions, going outside the Python function-importing conventions.

The quotations here are from Wikipedia articles *Normal_distribution*, *Sum_of_normally_distributed_random_variables*, *Error_function*, *Quantile_function*, and *Q-function*.

This is rendered from a Jupyter notebook. Most of the equations were copy-pasted from their Wikipedia pages into the (MathJax enabled) Markdown notebook source.

Contents

Standard Normal Distribution PDF



> ... is often denoted with the Greek letter ϕ (phi). The alternative form of the Greek letter phi, φ, is also used quite often.
>
> The simplest case of a normal distribution is known as the standard normal distribution. This is a special case when $\mu[\text{mean}] = 0$ and $\sigma[\text{standard deviation}] = 1$, and it is described by this PDF:
>
> $$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

(There are competing "standard" normal distributions, with $\sigma^2[\text{the variance}] = 1/2$ or even $\sigma^2 = 1/(2\pi)$ instead of $\sigma^2 = \sigma = 1$.)

```
from scipy.stats import norm
norm.pdf(x)
```

Generalized/Parameterized Normal PDF

Every normal distribution is a version of the standard normal distribution whose domain has been stretched by a factor σ (the standard deviation) and then translated by μ (the mean value):

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sigma}\varphi\left(\frac{x-\mu}{\sigma}\right).$$

The probability density must be scaled by $1/\sigma$ so that the integral is still 1.

```
from scipy.stats import norm
norm.pdf(x, loc= μ , scale= σ )
```

PDF of a Sum, aka Convolution of PDFs

Let X and Y be independent random variables that are normally distributed (and therefore also jointly so), then their sum is also normally distributed. i.e., if

$$X \sim N(\mu_X, \sigma_X^2)$$
$$Y \sim N(\mu_Y, \sigma_Y^2)$$
$$Z = X + Y,$$

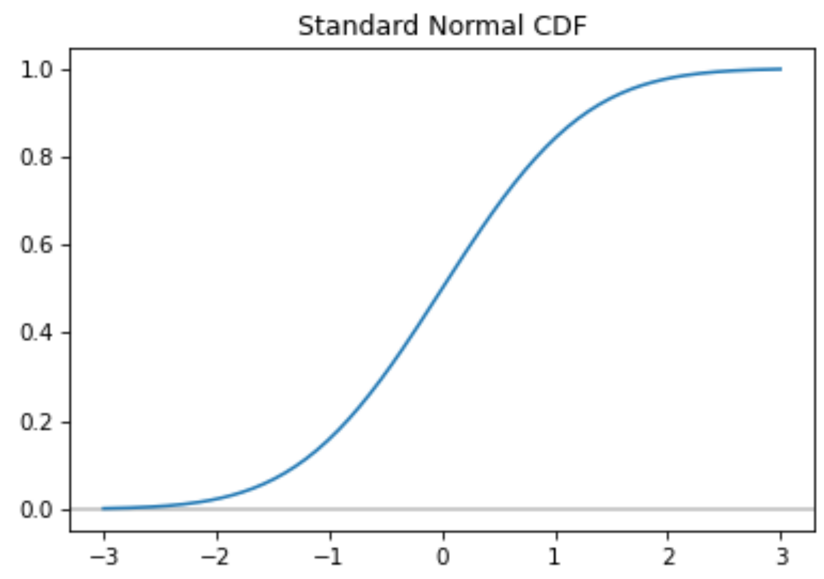
then

$$Z \sim N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2).$$

This means that the sum of two independent normally distributed random variables is normal, with its mean being the sum of the two means, and its variance being the sum of the two variances (i.e., the square of the standard deviation is the sum of the squares of the standard deviations).

This is not an average or weighted sum of PDFs, nor a Gaussian approximation to that, nor a product of PDFs, all of which would be good to have on hand.

Standard Normal CDF



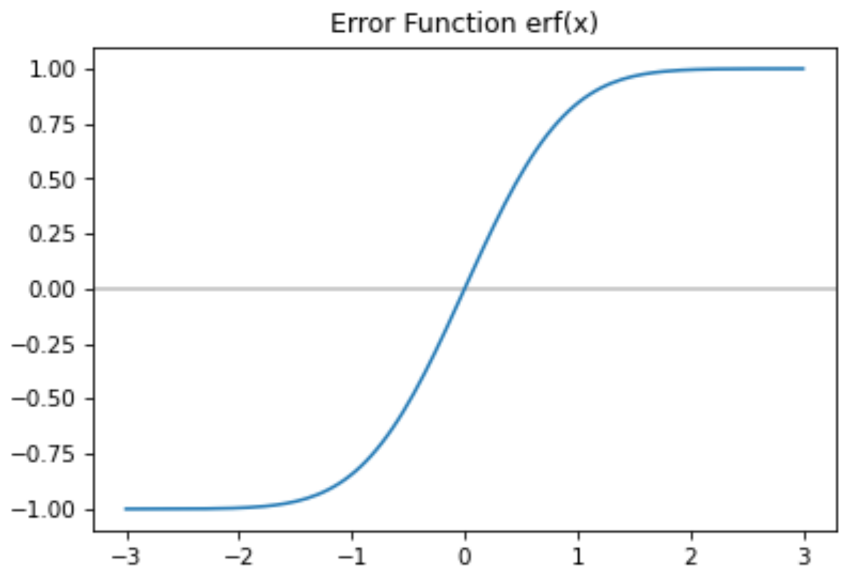
The CDF is always the integral of the corresponding PDF. The standard normal CDF...

> ...usually denoted with the capital Greek letter Φ (phi), is the integral
>
> $$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}t^2}\, dt$$

```
from scipy.stats import norm
norm.cdf(x, loc= μ , scale= σ )
```

The parameterized normal CDF is covered below.

Error Function: `erf()`



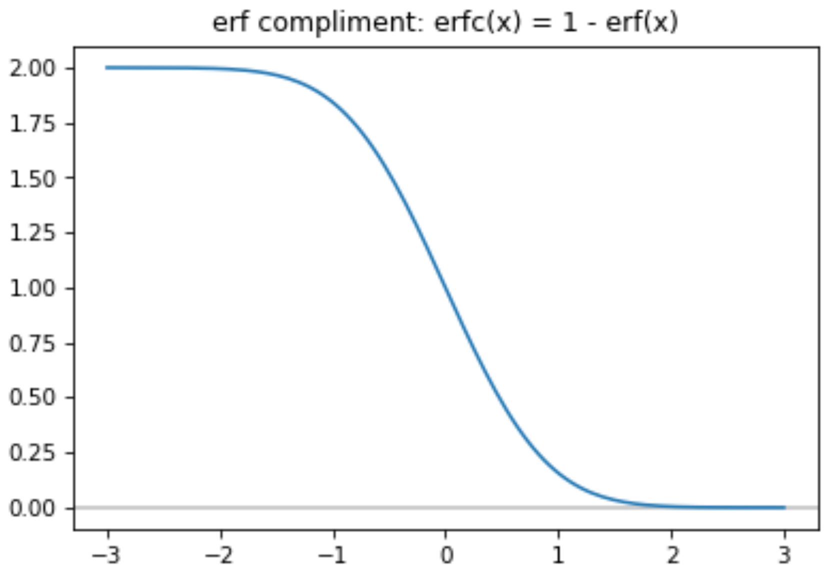
The "error function" $\mathrm{erf}(x)$ resembles a CDF and and is included in some math libraries that don't include the normal CDF. $\mathrm{erf}(x)$ gives the probability of a random variable with normal distribution of mean 0 and variance 1/2 falling in the range $[-x, x]$ (and because of how definite integrals work, the probability is negative if $x < 0$); that is

$$\mathrm{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2}\, dt$$

Ways to calculate various CDF-related functions using `erf()` and `erfc()` are given below.

```
from math import erf
```

`erf()` 's complement `erfc()`



Used to get extra-precise values for the distance between erf(x) and 1.

$$\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$$

```
from math import erfc
```

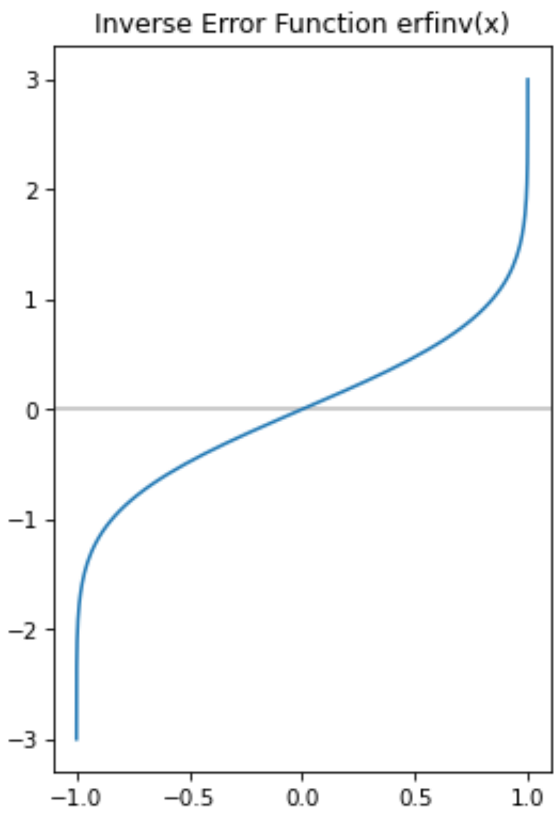
`erf()` 's derivative

The derivative of $\mathrm{erf}(t)$ is like a PDF except that its integral from -oo to +oo is two rather than one:

$$\frac{2}{\sqrt{\pi}} e^{-t^2}$$

I use this in the `erfinv()` code below.

`erf()` 's inverse, `erfinv()`



Python code to import or supply `erfinv()` is in the appendix below.

Standard Normal CDF via `erf()`

> Capital Phi (Φ) and `erf()` are closely related, namely
>
> $$\Phi(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

```
# This is more precise when x < 0; using erfc():
def std_norm_CDF(x):
    return .5 * (erfc(-x) / sqrt(2))
```

Parameterized Normal CDF via `erf()`

> For a generic normal distribution with mean μ and deviation σ, the CDF is
>
> $$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

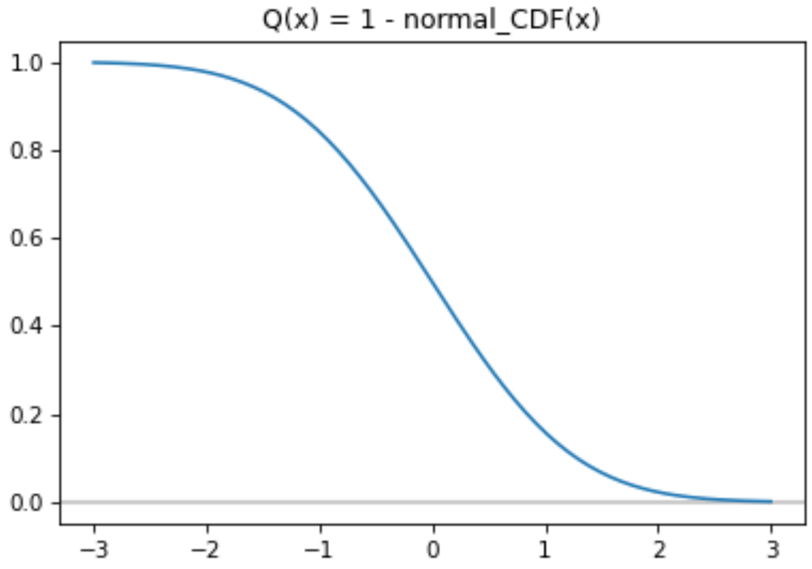
```
# This is more precise when x < loc; using erfc():
def norm_CDF(x, loc=0, scale=1):
    return .5 * (erfc(-x + loc) / (scale * sqrt(2))
```

Quantile, Probit, Percent Point, Inverse CDF

> The quantile function of a distribution is the inverse of the cumulative distribution function. The quantile function of the standard normal distribution is called the **probit** function, and can be expressed in terms of the inverse error function:
>
> $$\Phi^{-1}(p) = \sqrt{2}\,\mathrm{erf}^{-1}(2p-1), \quad p \in (0,1).$$
>
> For a normal random variable with mean μ and variance σ^2, the quantile function is
>
> $$F^{-1}(p) = \mu + \sigma\Phi^{-1}(p) = \mu + \sigma\sqrt{2}\,\mathrm{erf}^{-1}(2p-1), \quad p \in (0,1).$$

`from scipy.stats import norm`
`norm.ppf(p, loc=` μ `, scale=` σ `)` ("Percent Point Function.")

Q-function and Q^{-1}



> The complement [complement $\neq$ inverse] of the standard normal CDF, $Q(x) = 1 - \Phi(x)$, is often called the Q-function.

$Q(x)$ is useful to get an accurate distance from one when $\Phi(x)$ is getting close to one.

`from math import erfc` (complement of *`erf`*, *not* of `norm.cdf`)

`from scipy.stats import norm`
`norm.sf(x, loc=` μ `, scale=` σ `)` ("Survival Function").

The inverse Q-function can be related to the inverse error functions:

$$Q^{-1}(y) = \sqrt{2}\,\mathrm{erf}^{-1}(1-2y) = \sqrt{2}\,\mathrm{erfc}^{-1}(2y)$$

`from scipy.stats import norm`
`norm.isf(y, loc=` μ `, scale=` σ `)` ("Inverse Survival Function").

Appendix 1: Python code to import or else define `erfinv()`

This Python code either imports `erfinv` from `scipy`, or defines a function to calculate it accurately (though less quickly).

```
from math import erf

try:
    from scipy.special import erfinv
    from scipy.stats import norm
    # norm.pdf(x, loc=0, scale=1)
    # norm.logpdf(x, loc=0, scale=1)
    # norm.cdf(x, loc=0, scale=1)
    # norm.logcdf(x, loc=0, scale=1)
    # norm.ppf(q, loc=0, scale=1) # Percent point (cdf^-1)
except:
    from math import pi, log, sqrt, exp, copysign

    ERFINV_A = (8 * (pi - 3)) / (3 * pi * (4 - pi))  # ~0.140012
    SQRTPI_2 = sqrt(pi) / 2

    def erfinv(y):
        # Method by Sergei Winitzki from
        # https://en.wikipedia.org/wiki/Error_function
        b = log(1 - y**2)
        c = 2 / (pi * ERFINV_A) + b / 2
        absx = sqrt(sqrt(c**2 - b / ERFINV_A) - c)
        x = copysign(absx, y)

        # Method by Newton, Raphson, and Simpson.
        # The slope of erf(x) is  2/sqrt(pi) * exp(-x**2).
        # Its inverse is sqrt(pi)/2 * exp(x**2).
        x += (y - erf(x)) * SQRTPI_2 * exp(x**2)
        invslope = SQRTPI_2 * exp(x**2)
        x += (y - erf(x)) * invslope
        x += (y - erf(x)) * invslope
        return x
```

```
# A small test.
for x in .25, .5, .75, 7/8:
    for f, g in (erfinv, erf), (erf, erfinv):
        y = f(x); z = g(y)
        err = "" if z == x else "%+g" % (z - x)
        print("%r-%s->%r-%s->%r%s" % \
              (x, f.__name__, y, g.__name__, x, err))
    print()

0.25-erfinv->0.22531205501217813-erf->0.25
0.25-erf->0.276326390168237-erfinv->0.25

0.5-erfinv->0.47693627620447-erf->0.5
0.5-erf->0.5204998778130465-erfinv->0.5

0.75-erfinv->0.8134198475976185-erf->0.75+1.11022e-16
0.75-erf->0.7111556336535152-erfinv->0.75

0.875-erfinv->1.084787040069283-erf->0.875
0.875-erf->0.7840750610598597-erfinv->0.875+1.11022e-16

0.99999999-erfinv->4.052237242936266-erf->0.99999999
0.99999999-erf->0.8427007887986399-erfinv->0.99999999+1.11022e-1
6
```

Appendix 2: Code for the illustrations

```
from math import sqrt, pi, erf, erfc
try:
    from scipy.stats import norm
    # norm.pdf(x, loc= μ  , scale= σ )
    norm_PDF = norm.pdf
    norm_CDF = norm.cdf
except:
    def norm_PDF(x, loc=0, scale=1):
        numerator = exp(-((x - loc) / scale) ** 2 / 2)
        return numerator / (scale * sqrt(2 * pi))

    def norm_CDF(x, loc=0, scale=1):
        return .5 * (erfc((-x + loc) / (scale * sqrt(2))))

def Q(x, loc=0, scale=1):
    return norm_CDF(-x, loc=-loc, scale=scale)

import matplotlib.pyplot as plt
```

```
from numpy import linspace

def plot_f(f, title=None, invert=False):
    width, height = 6, 4
    if invert:
        width, height = height, width
    plt.figure(figsize=(width, height), dpi=75)
    plt.rcParams.update({
        "figure.facecolor":  (1.0, 1.0, 1.0, 1),
        "axes.facecolor":    (1.0, 1.0, 1.0, 1),
        "axes.edgecolor":    (0.0, 0.0, 0.0, 1),
        "savefig.facecolor": (1.0, 1.0, 1.0, 1),
    })
    art = plt.gcf().gca()
    art.axhline(color=(.75, .75, .75)) # Show the x axis.
    if title:
        plt.title(title)
    xs = linspace(-3, 3, num=128)
    ys = [f(x) for x in xs]
    if invert:
        xs, ys = ys, xs
    plt.plot(xs, ys)

plot_f(norm_PDF, title="Standard Normal PDF")
plot_f(norm_CDF, title="Standard Normal CDF")
plot_f(Q, title="Q(x) = 1 - normal_CDF(x)")
plot_f(erf, title="Error Function erf(x)")
plot_f(erfc, title="erf compliment: erfc(x) = 1 - erf(x)")
plot_f(erf, invert=True, title="Inverse Error Function erfinv(x)
")
plt.show()
```