S1: Principles of Data Science

Problem Sheet 2 Solutions

MPhil in Data Intensive Science

Matt Kenzie
mk652@cam.ac.uk

Michealmas Term 2023

Problem Sheet 2

1. Below is some code (`multivar_normal.py`) which will make these kind of plots.

```
# Solution to Problem Sheet 2 Question 1

# let's not waste any time and use scip multivar_normal
import numpy as np
from scipy.stats import multivariate_normal as mvn
import matplotlib.pyplot as plt
plt.style.use('code/mphil.mplstyle')

def get_mvn( mean, cov, marginal=None, conditional=None, verbose=False):
    """ get mvn distribution or any marginal or conditional subset of it

    Parameters
    ----------
    mean : array
        mu values of the multivariate normal. must be 1 dimensional.
    cov : array
        covariance matrix of the multivariate normal. must be 2 dimensional
    marginal : int or array of int
        index or list of indices for which to marginalise over
    conditional : tuple or array of tuple
        must be a two element tuple or list of two element tuples which give (ind: val),
        the index and value which to be conditional upon
    """
    mean = np.asarray(mean)
    cov  = np.asarray(cov)
    # checks
    assert( mean.ndim==1 )
    assert( cov.ndim==2 )
    assert( mean.shape[0] == cov.shape[0] )
    assert( cov.shape[0] == cov.shape[1] )

```

```
    ndim = mean.shape[0]

    if marginal is not None:
        if np.isscalar( marginal ):
            marginal = [ marginal ]
        marginal = np.asarray( marginal )
        assert( len(marginal) < ndim )

        mmean = np.delete( mean, marginal, axis=0 )
        mcov  = np.delete( np.delete( cov, marginal, axis=0 ), marginal, axis=1 )

        return mvn( mmean, mcov )

    elif conditional is not None:
        conditional = np.asarray( conditional )

        argdrop = conditional[:,0].astype(np.int32)
        argkeep = np.array( [ i for i in range(ndim) if i not in argdrop ] )
        condvals = conditional[:,1]

        m1 = mean[argkeep]
        m2 = mean[argdrop]

        S11 = cov[ np.ix_(argkeep, argkeep) ]
        S22 = cov[ np.ix_(argdrop, argdrop) ]
        S12 = cov[ np.ix_(argkeep, argdrop) ]
        S21 = cov[ np.ix_(argdrop, argkeep) ]

        cov_inv = np.linalg.inv( cov )
        cov1_inv = np.delete( np.delete( cov_inv, argdrop, axis=0), argdrop, axis=1 )
        cov1 = np.linalg.inv( cov1_inv )

        S22_inv = np.linalg.inv( S22 )

        cmean = m1 + S12 @ S22_inv @ ( condvals - m2 )
        ccov = S11 - S12 @ S22_inv @ S21

        return mvn( cmean, ccov )

    else:
        return mvn( mean, cov )

mu1 = 1
mu2 = 4
sg1 = 3
sg2 = 2

# 4d example
# mean = np.array([1, 4, 2, 3])
# err  = np.array([3, 2, 1, 4]).reshape((-1,1))
# corr = np.array([[1,0.5,0.2,0.2],[0.5,1,0.2,0.2],[0.2,0.2,1,0.6],[0.2,0.2,0.6,1]])
# cov  = err.T * corr * err
```

```
# get_mvn( mean, cov, conditional=[(2,3),(3,4)] )


# 2d example
mean = np.array( [mu1,mu2] )
err  = np.array( [sg1,sg2] ).reshape( (-1,1) )
corr = np.array( [[1,0.5],[0.5,1]] )
cov  = err.T * corr * err

dist_xy = get_mvn( mean, cov )
marg_x  = get_mvn( mean, cov, marginal=1 )
marg_y  = get_mvn( mean, cov, marginal=0 )

# Make some plots
xrange = [ mu1 - 3*sg1, mu1 + 3*sg1 ]
yrange = [ mu2 - 3*sg2, mu2 + 3*sg2 ]
x = np.linspace(*xrange,100)
y = np.linspace(*yrange,100)
X, Y = np.meshgrid(x,y)

# The 2D distribution and the 1D marginals
from matplotlib.gridspec import GridSpec
fig = plt.figure( figsize=(16,9) )
gs = GridSpec(2, 2, figure=fig)
ax1 = fig.add_subplot( gs[:,0] )
ax2 = fig.add_subplot( gs[0,1] )
ax3 = fig.add_subplot( gs[1,1] )
pos = np.dstack([X,Y])
Z = dist_xy.pdf(pos)
im = ax1.contourf( X, Y, Z )
cb = fig.colorbar(im, ax=ax1, location='top')
cb.set_label('Probability Density, $f(X,Y)$')
ax1.set_xlabel('$X$')
ax1.set_ylabel('$Y$')

# The 1D marginal distributions
ax2.plot( x, marg_x.pdf( x.reshape((-1,1)) ), label='Marginal over $Y$' )
ax3.plot( y, marg_y.pdf( y.reshape((-1,1)) ), label='Marginal over $X$' )
# add some 1d conditional distributions
cond_x1 = get_mvn(mean, cov, conditional=[(1,6)])
cond_x2 = get_mvn(mean, cov, conditional=[(1,0)])
ax2.plot( x, cond_x1.pdf( x.reshape((-1,1)) ), ls='--', label='
    Conditional on $Y=6$')
ax2.plot( x, cond_x2.pdf( x.reshape((-1,1)) ), ls='--', label='
    Conditional on $Y=0$')
cond_y1 = get_mvn(mean, cov, conditional=[(0,3)])
cond_y2 = get_mvn(mean, cov, conditional=[(0,-2)])
ax3.plot( y, cond_y1.pdf( y.reshape((-1,1)) ), ls='--', label='
    Conditional on $X=1$')
ax3.plot( y, cond_y2.pdf( y.reshape((-1,1)) ), ls='--', label='
    Conditional on $X=-2$')
ax2.set_xlabel('$X$')
ax2.set_ylabel('$g(X) = \int f(X,Y) dY$')
ax2.legend()
ax3.set_xlabel('$Y$')
ax3.set_ylabel('$h(Y) = \int f(X,Y) dX$')
```

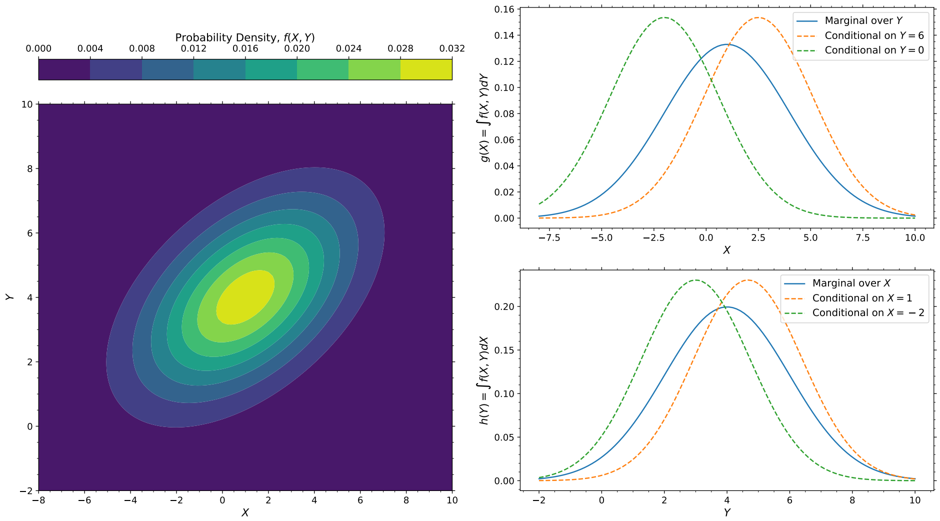
```
ax3.legend()
fig.savefig('figs/multivar.pdf')

# Now we do the 2D conditional
Z_XY = Z / marg_y.pdf( Y.reshape( (*Y.shape,1) ) )
Z_YX = Z / marg_x.pdf( X.reshape( (*X.shape,1) ) )
fig, ax = plt.subplots(1, 2, figsize=(12.8,4.8))
im = ax[0].contourf( X, Y, Z_XY )
cb = fig.colorbar(im, ax=ax[0])
cb.set_label('Conditional Probability, $p(X|Y)$')
ax[0].set_xlabel('$X$')
ax[0].set_ylabel('$Y$')
im = ax[1].contourf( X, Y, Z_YX )
cb = fig.colorbar(im, ax=ax[1])
cb.set_label('Conditional Probability, $p(Y|X)$')
ax[1].set_xlabel('$X$')
```

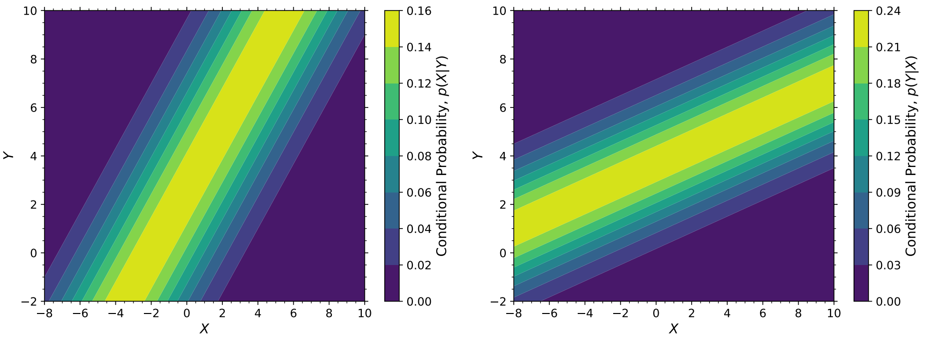
It produces the following plots. The first 2D distribution just shows the 2D distribution, $f(X, Y)$, alongside the two marginal distributions:

$$g(X) = \int f(X, Y)dx \quad \text{and} \quad h(Y) = \int f(X, Y)dX. \tag{1}$$

The bottom two 2D distributions show the conditional p.d.f.s for X given Y and Y given X:

$$p(X|Y) = f(X, Y)/g(Y) \quad \text{and} \quad p(Y|X) = f(X, Y)/h(X). \tag{2}$$





For a 3D Gaussian it becomes difficult, you could try and draw a blob of constant contour in 3D space (rather difficult to interpret). Another option is to make a gif which scans values in 1D and shows the 2D disitrbution in others. I made such a thing and which is at the bottom of `multivar_normal.py`.

2. Exponential disitrbution p.d.f. given by

$$p(x;\lambda) = \lambda e^{-\lambda x} \qquad \text{for} \qquad x \in [0, \infty] \tag{3}$$

We can prove or already assume it is normalised so that

$$\int_0^\infty \lambda e^{-\lambda x} dx = 1. \tag{4}$$

First the mean:

$$\mu = E[x] = \int_0^\infty x\lambda e^{-\lambda x} dx \tag{5}$$

which can be solved using integration by parts with

$$u = \lambda x \qquad u' = \lambda \tag{6}$$

$$v' = e^{-\lambda x} \qquad v' = -\frac{e^{-\lambda x}}{\lambda}. \tag{7}$$

So that

$$\mu = E[x] = \int_0^\infty x\lambda e^{-\lambda x} dx \tag{8}$$

$$= \left[-\lambda x e^{-\lambda x}\right]_0^\infty + \int_0^\infty e^{-\lambda x} dx \tag{9}$$

$$= 0 + \left[-\frac{e^{-\lambda x}}{\lambda}\right]_0^\infty = \frac{1}{\lambda}. \tag{10}$$

For the variance we need, $V(x) = E[x^2] - E[x]^2 = E[x^2] - \mu^2$, so let's compute $E[x^2]$:

$$E[x^2] = \int_0^\infty x^2 \lambda e^{-\lambda x} dx. \tag{11}$$

By parts with

$$u = \lambda x^2 \qquad u' = 2\lambda x \tag{12}$$

$$v' = e^{-\lambda x} \quad v = -\frac{e^{-\lambda x}}{\lambda} \tag{13}$$

gives

$$E[x^2] = \int_0^\infty x^2 \lambda e^{-\lambda x} dx \tag{14}$$

$$= \left[-x^2 e^{-\lambda x}\right]_0^\infty + \int_0^\infty 2x e^{-\lambda x} dx \tag{15}$$

$$= 0 + \frac{2}{\lambda} E[x] \tag{16}$$

$$= \frac{2}{\lambda^2}. \tag{17}$$

Therefore the variance is

$$V(x) = E[x^2] - E[x]^2 = \frac{2}{\lambda^2} - \frac{1}{\lambda^2} = \frac{1}{\lambda^2}. \tag{18}$$

3. χ^2 distribution p.d.f. given by

$$p(x;k) = \frac{1}{2^{k/2}\Gamma(\frac{k}{2})} x^{k/2-1} e^{-x/2} \tag{19}$$

and is only defined for positive values of x. Makes life easier if we define the normalisation constant (which does not depend on x) as

$$N = \frac{1}{2^{k/2}\Gamma(\frac{k}{2})}. \tag{20}$$

To compute the mean:

$$\mu = E[x] = \int_0^\infty x p(x) dx \tag{21}$$

$$= N \int_0^\infty x x^{k/2-1} e^{-x/2} dx \tag{22}$$

$$= N \int_0^\infty x^{k/2} e^{-x/2} dx. \tag{23}$$

Use integration by parts with

$$u = x^{k/2} \qquad \text{and} \qquad v' = e^{-x/2} \tag{24}$$

$$u' = \tfrac{k}{2} x^{k/2-1} \qquad \text{and} \qquad v = -2e^{-x/2} \tag{25}$$

Continuing from above:

$$
\begin{aligned}
\mu = E[x] &= N \int_0^\infty x^{k/2} e^{-x/2} dx && (26)\\
&= N \left[-2e^{-x/2} x^{k/2}\right]_0^\infty + N \int_0^\infty 2e^{-x/2} \frac{k}{2} x^{k/2-1} dx && (27)\\
&= N\,(0-0) + Nk \int_0^\infty x^{k/2-1} e^{-x/2} dx && (28)\\
&= k \underbrace{\int_0^\infty N x^{k/2-1} e^{-x/2}}_{=1} && (29)\\
&= k. && (30)
\end{aligned}
$$

To compute the variance:

$$
V(x) = E[x^2] - E[x]^2 \tag{31}
$$

We just did $E[x]$ so lets now do $E[x^2]$:

$$
\begin{aligned}
E[x^2] &= \int_0^\infty x^2 p(x) dx && (32)\\
&= N \int_0^\infty x^2 x^{k/2-1} e^{-x/2} dx && (33)\\
&= N \int_0^\infty x^{k/2+1} e^{-x/2} dx. && (34)
\end{aligned}
$$

Use integration by parts with

$$
\begin{aligned}
u &= x^{k/2+1} && \text{and} && v' = e^{-x/2} && (35)\\
u' &= \left(\tfrac{k}{2}+1\right) x^{k/2} && \text{and} && v = -2e^{-x/2} && (36)
\end{aligned}
$$

Continuing from above:

$$
\begin{aligned}
E[x^2] &= N \int_0^\infty x^{k/2+1} e^{-x/2} dx && (37)\\
&= N \left[-2e^{-x/2} x^{k/2+1}\right]_0^\infty + N \int_0^\infty 2e^{-x/2} \left(\frac{k}{2}+1\right) x^{k/2} dx && (38)\\
&= N\,(0-0) + N(k+2) \int_0^\infty x^{k/2} e^{-x/2} dx && (39)\\
&= N(k+2) \int_0^\infty x^{k/2} e^{-x/2} dx. && (40)
\end{aligned}
$$

Using integration by parts again (which we actually already did with the mean) with

$$
\begin{aligned}
u &= x^{k/2} && \text{and} && v' = e^{-x/2} && (41)\\
u' &= \tfrac{k}{2} x^{k/2-1} && \text{and} && v = -2e^{-x/2} && (42)
\end{aligned}
$$

And continuing from above:

$$
\begin{aligned}
E[x^2] &= N(k+2)\int_0^\infty x^{k/2}e^{-x/2}dx \quad (43)\\
&= N(k+2)\left[-2e^{-x/2}x^{k/2}\right]_0^\infty + N(k+2)\int_0^\infty 2e^{-x/2}\frac{k}{2}x^{k/2-1}dx \quad (44)\\
&= N(k+2)\,(0-0) + N(k+2)k\int_0^\infty e^{-x/2}x^{k/2-1}dx \quad (45)\\
&= (k+2)k\,\underbrace{N\int_0^\infty e^{-x/2}x^{k/2-1}dx}_{=1} \quad (46)\\
&= k^2+2k. \quad (47)
\end{aligned}
$$

Therefore the variance is:

$$
\begin{aligned}
V(x) &= E[x^2]-E[x]^2 \quad (48)\\
&= k^2+2k-k^2 \quad (49)\\
&= 2k. \quad (50)
\end{aligned}
$$

It should be noted that there is an alternative solution based on the fact that if X is χ^2 distributed then X is also the sum of squared normally distributed variables. This is arguably mathematically easier but requires a bit of a deeper intuition.

4. Below is some code (`accept_reject.py`) that should do the job.

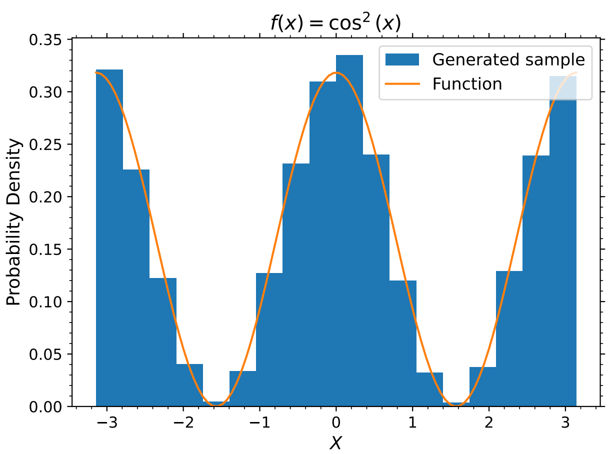
```python
## Solution to Problem Sheet 2 Question 3
## once again more optimal solutions will be available

import numpy as np
from scipy.optimize import brute, minimize
from scipy.integrate import quad
import matplotlib.pyplot as plt
plt.style.use('code/mphil.mplstyle')

def accept_reject_1d( func, xrange, size=1, fmax=None, ret_fmax=False,
    seed=None, stats=False ):
    """A simple function to run accept-reject generation.

    Parameters
    ----------
    func : callable
        A callable function which accepts one argument x and
        returns the function which you want to generate from.
        Note that for accept-reject there is no requirement
        that is normalised
    xrange : tuple or list
        Must be a two element tuple or list containing the
        range of x values to generate in
    size : int, optional
        Number of events to generate
    fmax : float, optional
        If already known you can provide the maximum value of
        the function. This speeds up generation a bit because
```

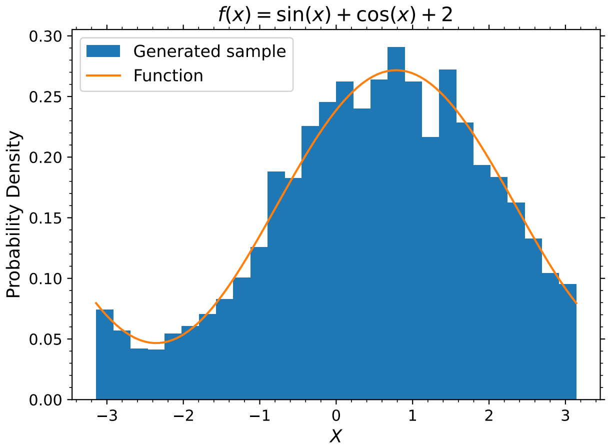
```python
        the algorithm can skip finding the maximum
    stats : bool, optional
        If True then track the generation efficiency statistics
        and return them

    Returns
    -------
    x : array
        an array of generated x values
    fmax : float, optional
        if ret_fmax=True then will also return the fmax found
    stats : tuple, optional
        if stats=True will also return the accept_eff and reject_eff

    """

    if fmax is None:
        # global
        f_to_min = lambda x: -func(x)
        x0 = brute( f_to_min, [xrange] )[0]
        # local
        x0 = minimize( f_to_min, x0=x0 ).x[0]
        fmax = func(x0)

    if seed is not None:
        np.random.seed(seed)

    res = []
    nrej = 0
    while len(res)<size:

        # generate uniform random in xrange
        x = np.random.uniform(*xrange)

        # compute the value of the func at this x
        fval = func(x)

        # check it's not negative
        if fval < 0:
            raise RuntimeError(f'Accept-reject found a negative function value. p.d.f.s cannot be negative. Bailing out')

        # check it's not bigger than fmax
        if fval > fmax:
            fmax = fval
            raise RuntimeError(f'Accept-reject found a larger function value than was provided {fval}. Updating fmax and bailing out')

        # now generate random between [0, fmax]
        ftest = np.random.uniform(0, fmax )

        # if ftest larger than fval reject else accept
        if ftest <= fval:
            res.append(x)
            print(f'Accept-rejecting {len(res)} / {size}', end='\r')
        # this just counts the number of rejections
```

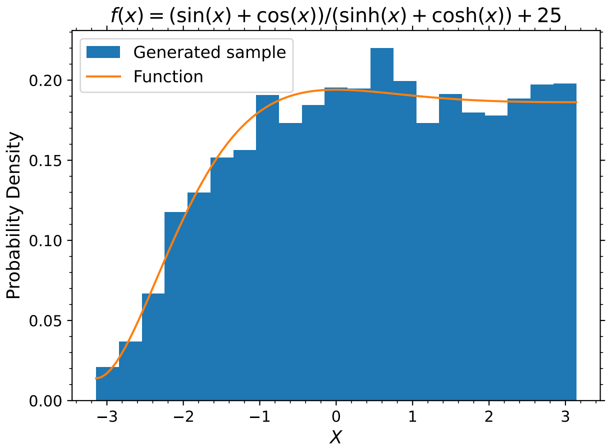
```
        else:
            nrej += 1

    if stats:
        ntot = size + nrej
        rej_eff = nrej / ntot
        acc_eff = size / ntot

    print('Accept-rejecting DONE         ')
    if ret_fmax:
        if stats:
            return res, fmax, acc_eff, rej_eff
        else:
            return res, fmax
    else:
        if stats:
            return res, acc_eff, rej_eff
        else:
            return res

def check_ok_plot( func, xrange, dset, save=None, title=None ):

    # going to normalise the function numerically
    N = 1/quad( func, *xrange )[0]

    x = np.linspace(*xrange, 100)

    fig, ax = plt.subplots()
    ax.hist( dset, density=True, bins='auto', label='Generated sample')
    ax.plot(x, N*func(x), label='Function')
    ax.set_xlabel('$X$')
    ax.set_ylabel('Probability Density')
    ax.legend()

    if title is not None:
        ax.set_title( title )

    if save is not None:
        fig.savefig(save)

# Run some examples
if __name__=="__main__":

    xrange = [-np.pi, np.pi]

    functions = [
        lambda x: np.cos(x)**2,
        lambda x: np.sin(x) + np.cos(x) + 2,
        lambda x: ( np.sin(x) + np.cos(x) ) / ( np.sinh(x) + np.cosh(x) ) + 25
    ]

    names = [ r'$f(x) = \cos^2(x)$',
              r'$f(x) = \sin(x) + \cos(x) + 2 $',
              r'$f(x) = ( \sin(x) + \cos(x) ) / ( \sinh(x) + \cosh(x) ) + 25$' ]
```

```python

files = [ 'cos2x', 'sinxcosx', 'sinxcosxosinhxcoshx' ]

for func, name, file in zip(functions, names, files):

    dset = accept_reject_1d( func, xrange=xrange, size=5000 )
    check_ok_plot( func, xrange, dset, save=f'figs/accept_reject_{file}.pdf', title=name )

# now check the efficiency for a normal distribution
func = lambda x: (1/np.sqrt(2*np.pi)) * np.exp( -x**2 / 2 )

acc_effs = []
rej_effs = []
sigmas = np.linspace(1,8,25)
for sigma in sigmas:
    xrange = [ -sigma, sigma ]
    dset, acc_eff, rej_eff = accept_reject_1d( func, xrange, size=10000, stats=True )
    acc_effs.append( acc_eff )
    rej_effs.append( rej_eff )

print('Accept efficiency at 8 sigma', acc_effs[-1])
fig, ax = plt.subplots()
ax.plot( sigmas, acc_effs, label='Accept Efficiency')
ax.plot( sigmas, rej_effs, label='Reject Overhead')
ax.set_xlabel('Width of generation range in standard deviations')
ax.set_ylabel('Efficiency')
ax.legend()
fig.savefig('figs/accept_reject_eff.pdf')
plt.show()
```

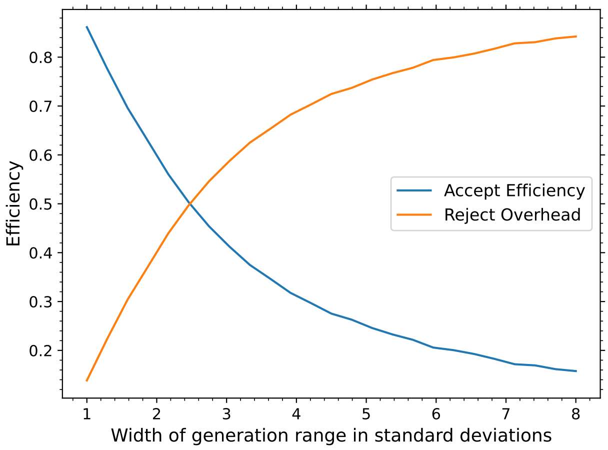
This will produce the following plots for the required functions







In terms of the efficiency for the normal distribution I get the following



which equates to about 16% accept efficiency at 8σ.

5. This should be quite straight-forward. If we have $f(x, y)$ then can propagate the uncertainty using

$$\sigma_f^2 = \left(\frac{\partial f}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial f}{\partial y}\right)^2 \sigma_y^2 + 2\left(\frac{\partial f}{\partial x}\right)\left(\frac{\partial f}{\partial y}\right)\rho\sigma_x\sigma_y. \tag{51}$$

In this case x and y are independent, therefore $\rho = 0$, so we can ignore the last term.

(a) $f = x + y$

$$\frac{\partial f}{\partial x} = 1 \quad \text{and} \quad \frac{\partial f}{\partial y} = 1 \tag{52}$$

$$\Rightarrow \sigma_f^2 = \sigma_x^2 + \sigma_y^2 \tag{53}$$

(b) $f = xy$

$$\frac{\partial f}{\partial x} = y \quad \text{and} \quad \frac{\partial f}{\partial y} = x \tag{54}$$

$$\Rightarrow \sigma_f^2 = y^2\sigma_x^2 + x^2\sigma_y^2 \tag{55}$$

$$\Rightarrow \left(\frac{\sigma_f}{f}\right)^2 = \left(\frac{\sigma_x}{x}\right)^2 + \left(\frac{\sigma_y}{y}\right)^2. \tag{56}$$

(c) $f = x/y$

$$\frac{\partial f}{\partial x} = \frac{1}{y} \quad \text{and} \quad \frac{\partial f}{\partial y} = -\frac{x}{y^2} \tag{57}$$

$$\Rightarrow \sigma_f^2 = \frac{\sigma_x^2}{y^2} + \frac{x^2\sigma_y^2}{y^4} \tag{58}$$

$$\Rightarrow \left(\frac{\sigma_f}{f}\right)^2 = \left(\frac{\sigma_x}{x}\right)^2 + \left(\frac{\sigma_y}{y}\right)^2. \tag{59}$$

(d) $f = \cos(x)$

$$\frac{\partial f}{\partial x} = -\sin(x) \tag{60}$$

$$\Rightarrow \sigma_f^2 = \sin^2(x)\sigma_x^2. \tag{61}$$

(e) $f = \sin(x)$

$$\frac{\partial f}{\partial x} = \cos(x) \tag{62}$$

$$\Rightarrow \sigma_f^2 = \cos^2(x)\sigma_x^2. \tag{63}$$

6. Recall from the lectures that an estimator is said to be efficient if the variance of that estimator is the minimum variance bound. The minimum variance bound is the negative inverse of the expectation of the double differential of the log-likelihood, thus if we can show that the following holds

$$V(\hat{\theta}) = -E\left[\frac{\partial^2 \ln L}{\partial \theta^2}\right]^{-1}, \tag{64}$$

then our estimator $\hat{\theta}$ for θ is efficient.

For the normal distribution we have a p.d.f. given by

$$p(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{65}$$

Thus we can write the log-likelihood over some number, N, observations as

$$\ln L = \ln\left[\prod_i^N \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)\right] \tag{66}$$

$$= \sum_i^N \ln\left[\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right)\right] \tag{67}$$

$$= -N\ln(\sigma\sqrt{2\pi}) - \sum_i^N \frac{(x_i-\mu)^2}{2\sigma^2}. \tag{68}$$

Differentiating this twice with respect to the mean μ gives

$$\frac{\partial}{\partial\mu}\left(\frac{\partial \ln L}{\partial\mu}\right) = \frac{\partial}{\partial\mu}\left(\frac{\partial}{\partial\mu}\left(-N\ln(\sigma\sqrt{2\pi}) - \sum_i^N \frac{(x_i-\mu)^2}{2\sigma^2}\right)\right) \tag{69}$$

$$= \frac{\partial}{\partial\mu}\left(\sum_i^N \frac{x_i-\mu}{\sigma^2}\right) \tag{70}$$

$$= -\sum_i^N \frac{\mu}{\sigma^2} \tag{71}$$

$$= -\frac{N}{\sigma^2}. \tag{72}$$

This has no dependence on x so the expectation is just the same, therefore we can write the minimum variance bound as

$$-E\left[\frac{\partial^2 \ln L}{\partial\theta^2}\right]^{-1} = \frac{\sigma^2}{N}. \tag{73}$$

Recall from the central limit theorem discussion and our sample estimates of variance in the first few lectures that the variance on the sample estimate of the mean is given by exactly this, *i.e.*

$$V(\hat{\mu}) = \frac{\sigma^2}{N}. \tag{74}$$

Thus we see it is an efficienct estimate.

7. You should stress how nice this result is, *i.e.* for an exponential decay distribution you don't need to fit anything to get the slope, just average the sample.

   The p.d.f. for an exponential we saw in the lectures was in terms of the parameter λ but often this is written in terms of the average lifetime $\tau = 1/\lambda$,

$$p(t;\lambda) = \lambda e^{-\lambda t} \quad \Leftrightarrow \quad p(t;\tau) = \frac{1}{\tau}e^{-t/\tau}, \tag{75}$$

   where decay times must be postive, thus the p.d.f. is only valid for $t \geq 0$. If you want you can spend 2 minutes convincing yourself these are properly normalised.

Lets first compute the log-likelihood over N observations:

$$\ln L = \ln \left[\prod_i^N \frac{1}{\tau} e^{-t_i/\tau} \right] \tag{76}$$

$$= \sum_i^N \left[\ln \frac{1}{\tau} - \frac{t_i}{\tau} \right] \tag{77}$$

$$= \sum_i^N \left[-\ln \tau - \frac{t_i}{\tau} \right] \tag{78}$$

Now differentiate with respect to the parameter τ to give

$$\frac{d \ln L}{d\tau} = \sum_i^N \left[\frac{t_i}{\tau^2} - \frac{1}{\tau} \right]. \tag{79}$$

To find the maximum of the log-likelihood we can set the differential to zero and solve for τ to give an estimate for τ, $\hat{\tau}$:

$$\sum_i^N \left[\frac{t_i}{\hat{\tau}^2} - \frac{1}{\hat{\tau}} \right] = 0 \tag{80}$$

$$\Rightarrow \frac{1}{\hat{\tau}^2} \sum_i^N t_i = \frac{N}{\hat{\tau}} \tag{81}$$

$$\Rightarrow \hat{\tau} = \frac{1}{N} \sum_i^N t_i, \tag{82}$$

which is simply the average of the measured decay times.

8. We have no uncertainties in this case so can set all $\sigma_i = 1$ thus the χ^2 in this case is given as

$$\chi^2 = \sum_i^N \frac{(y_i - m x_i - c)^2}{\cancelto{1}{\sigma_i^2}} \tag{83}$$

$$= \sum_i^N (y_i - m x_i - c)^2. \tag{84}$$

To find the minimum (*i.e.* least-squares estimate) of the parameter m or c we differentiate w.r.t m or c and set to zero which will give us two equations:

$$\frac{\partial \chi^2}{\partial m} = -2 \sum_i^N x_i (y_i - m x_i - c) \tag{85}$$

$$= -2(\overline{xy} - m\overline{x^2} - c\bar{x}) \tag{86}$$

$$\frac{\partial \chi^2}{\partial c} = -2 \sum_i^N (y_i - m x_i - c) \tag{87}$$

$$= -2(\bar{y} - m\bar{x} - c). \tag{88}$$

Now set differentials to zero and solve to find the estimates, $\hat{m}$ and $\hat{c}$:

$$1: \quad \overline{xy} - \hat{m}\overline{x^2} - \hat{c}\bar{x} = 0 \tag{89}$$

$$2: \quad \bar{y} - \hat{m}\bar{x} - \hat{c} = 0. \tag{90}$$

The second equation above we can rearrange in terms of $\hat{c}$ to give us the first part of the answer

$$\hat{c} = \bar{y} - \hat{m}\bar{x}. \tag{91}$$

We can then plug this into the first equation and solve for $\hat{m}$:

$$\begin{aligned}
0 &= \overline{xy} - \hat{m}\overline{x^2} - \hat{c}\bar{x} && (92)\\
&= \overline{xy} - \hat{m}\overline{x^2} - \bar{x}\bar{y} + \hat{m}\bar{x}^2 && (93)\\
&= \overline{xy} - \bar{x}\bar{y} - \hat{m}(\overline{x^2} - \bar{x}^2) && (94)\\
\Rightarrow \hat{m} &= \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2} && (95)\\
&= \frac{\mathrm{cov}(x,y)}{V(x)} && (96)
\end{aligned}$$

We can also plug this expression for $\hat{m}$ back into the equation for $\hat{c}$ to provide an expression for $\hat{c}$ without dependence on $\hat{m}$:

$$\hat{c} = \frac{\overline{x^2}\bar{y} - \bar{x}\overline{xy}}{\overline{x^2} - \bar{x}^2}. \tag{97}$$