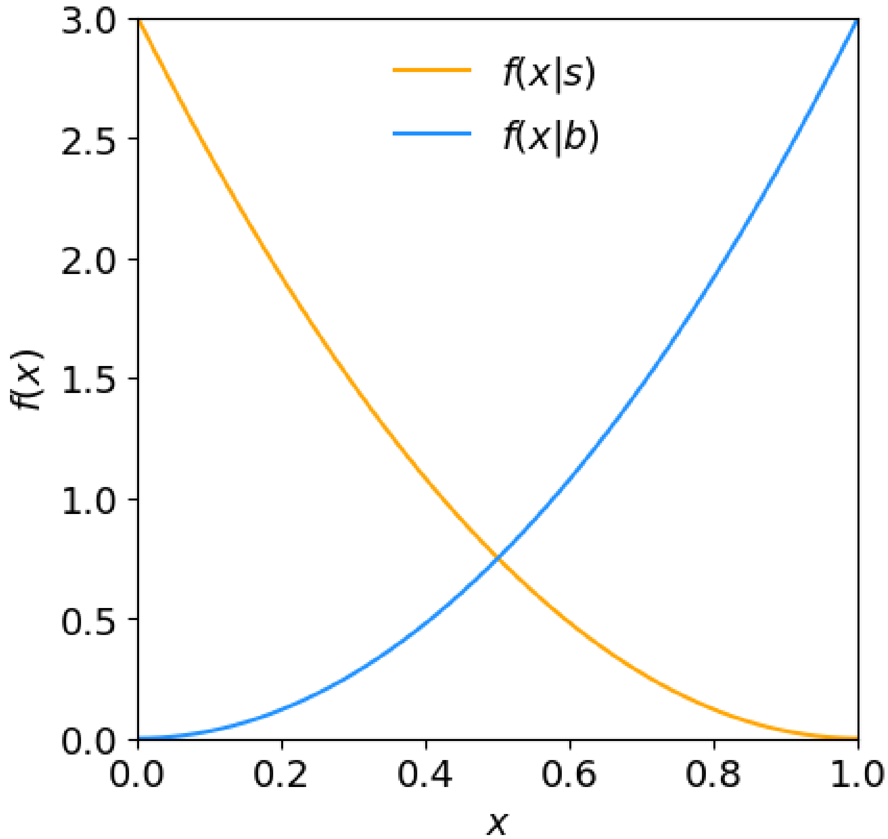
hypTest

January 10, 2025

[2]:
```
# Program to find measure of expected significance as a function
# of a cut value x_cut applied to measured variable x.
# G. Cowan / RHUL Physics / December 2022

import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
plt.rcParams["font.size"] = 14
```

[3]:
```
#  Plot the pdfs
def f_s(x):
    return 3.*(1-x)**2
def f_b(x):
    return 3.*x**2
x = np.linspace(0., 1., 201)
fs = f_s(x)
fb = f_b(x)
fig = plt.figure(figsize=(5,5))
plt.plot(x, fs, color='orange', label=r'$f(x|s)$')
plt.plot(x, fb, color='dodgerblue', label=r'$f(x|b)$')
plt.xlabel(r'$x$')
plt.ylabel(r'$f(x)$')
plt.xlim(0., 1.)
plt.ylim(0., 3.)
plt.legend(loc='upper center', frameon=False)
plt.subplots_adjust(left=0.15, right=0.9, top=0.9, bottom=0.15)
plt.show()
```



[4]:
```
# Add code here:

# Find x_cut for size alpha = 0.05


# Find power with respect to s hypothesis for this x_cut


# Calculate s, b, signficance for x_cut=0.1, s_tot=10, b_tot=100


# Find s, b, significance versus x_cut


# Plot s, b versus x_cut


# Plot Z_A versus x_cut

```

```
# Find x_cut that maximizes Z_A


# Repeat for case where b is uncertain
```

[5]:
```
from scipy.stats import rv_continuous
class my_dist(rv_continuous):
    "f(x)=f_b(x) distribution"
    def _pdf(self, x):
        return 3.*x**2                  # creating the pdf for background
my_Dist_back = my_dist(name='background',a=0,b=1)
```

[6]:
```
class my_dist2(rv_continuous):
    "f(x)=f_b(x) distribution"
    def _pdf(self, x):
        return 3.*(1-x)**2              # creating the pdf for signal
my_Dist_sig = my_dist2(name='signal',a=0,b=1)
```

[7]:
```
# class my_dist2(rv_continuous):
#     "f(x)=f_b(x) distribution"
#     def _pdf(self, x):
#         return (np.sin(np.pi*x))**2           # creating the pdf for signal
# my_Dist = my_dist2(name='signal',a=0,b=1)
```

Value of x_{cut} such that $P(x < x_{cut}|b)$ is equal to 0.05. I have used the ppf (Percent point function) from scipy to calculate the inverse of cdf.

[8]:
```
x_cut=my_Dist_back.ppf(0.05)
x_cut
```

[8]:
```
np.float64(0.36840314986403866)
```

Just to verify my answer I have integrated the area under the curve from 0 to x_{cut} of $f(x|b)$ function using "scipy.integrate. quad" package.

[9]:
```
from scipy.integrate import quad

def f(x):
  return 3.0*x*x

I, err = quad(f, 0, x_cut)
# I,tot, err_tot = quad(f, 0, 1)
print(I)
# print(err)
```

```
0.05
```

$P(x < x_{cut}|s)$

[10]:
```
cdf=my_Dist_sig.cdf(x_cut)
cdf
```

[10]:
```
np.float64(0.74804680710288)
```

Again for verification

[11]:
```
def g(x):
  return 3.*(1-x)**2

I2, err2 = quad(g, 0, x_cut)
# I,tot, err_tot = quad(f, 0, 1)
print(I2)
```

```
0.74804680710288
```

Expected number of background and signal events in $x < x_{cut}$ range.

[12]:
```
X_cut = 0.1
b_tot = 100
s_tot = 10

CDF_b = my_Dist_back.cdf(X_cut)
CDF_s = my_Dist_sig.cdf(X_cut)

s = s_tot * CDF_s
b = b_tot * CDF_b
```

Expected number of signal events in $x < x_{cut}(0.1)$ region.

[13]:
```
s
```

[13]:
```
np.float64(2.710000000000001)
```

Expected number of background events in $x < x_{cut}(0.1)$ region.

[11]:
```
b
```

[11]:
```
np.float64(0.10000000000000002)
```

Priors

[12]:
```
pi_s = s_tot/(s_tot+b_tot)
pi_b = b_tot/(s_tot+b_tot)
```

[13]:
```
pi_s
```

[13]:
```
0.09090909090909091
```

[14]:
```
pi_b
```

[14]: 0.9090909090909091

$P(s|x < x_{cut}) = \frac{P(x<x_{cut}|s)\pi(s)}{P(x<x_{cut}|s)\pi(s)+P(x<x_{cut}|b)\pi(b)}$

[15]:
```
P_s = (CDF_s * pi_s)/(CDF_s * pi_s + CDF_b * pi_b)
```

[16]:
```
P_s
```

[16]: np.float64(0.9644128113879004)

Calculating the p-value

[17]:
```
import math
def p_value(b, n_obs):
    term_list=[]
    for n in range(n_obs):
        term = np.exp(-b) * b**n / math.factorial(n)
        term_list.append(term)
    return (1-(np.sum(term_list)))
```

[20]:
```
p_val=p_value(0.5,3)
```

[21]:
```
p_val
```

[21]: np.float64(0.014387677966970713)

Calculating the significance $Z = \Phi^{-1}(1-p)$

[22]:
```
from statistics import NormalDist

sig = NormalDist(mu=0, sigma=1).inv_cdf(1-p_val)
sig
```

[22]: 2.186550477435837

[18]:
```
# Writing the expression for expected(median) significance
term1 = (s+b)
term2 = np.log(1+(s/b))
Ex_Sig = np.sqrt(2*((term1*term2)-s))
Ex_Sig
```

[18]: 3.650619812994654

[14]:
```
# Defining the function for expected(median) significance with x_c as parameter
def ex_sig_gen(x_c):
    b_tot = 100
    s_tot = 10

    CDF_b = my_Dist_back.cdf(x_c)
```

```
    CDF_s = my_Dist_sig.cdf(x_c)

    s = s_tot * CDF_s
    b = b_tot * CDF_b

    term1 = (s+b)
    term2 = np.log(1+(s/b))
    return np.sqrt(2*((term1*term2)-s))
```

Find the value of x_c for which Expected(median) significance has maximum value.

[26]:
```
from scipy.optimize import minimize_scalar
result = minimize_scalar(
    lambda x: -ex_sig_gen(x),  # Negative for maximization
    bounds=(0,1),  # Search range
    method='bounded'
)
xc_max = result["x"]
print(xc_max)
```

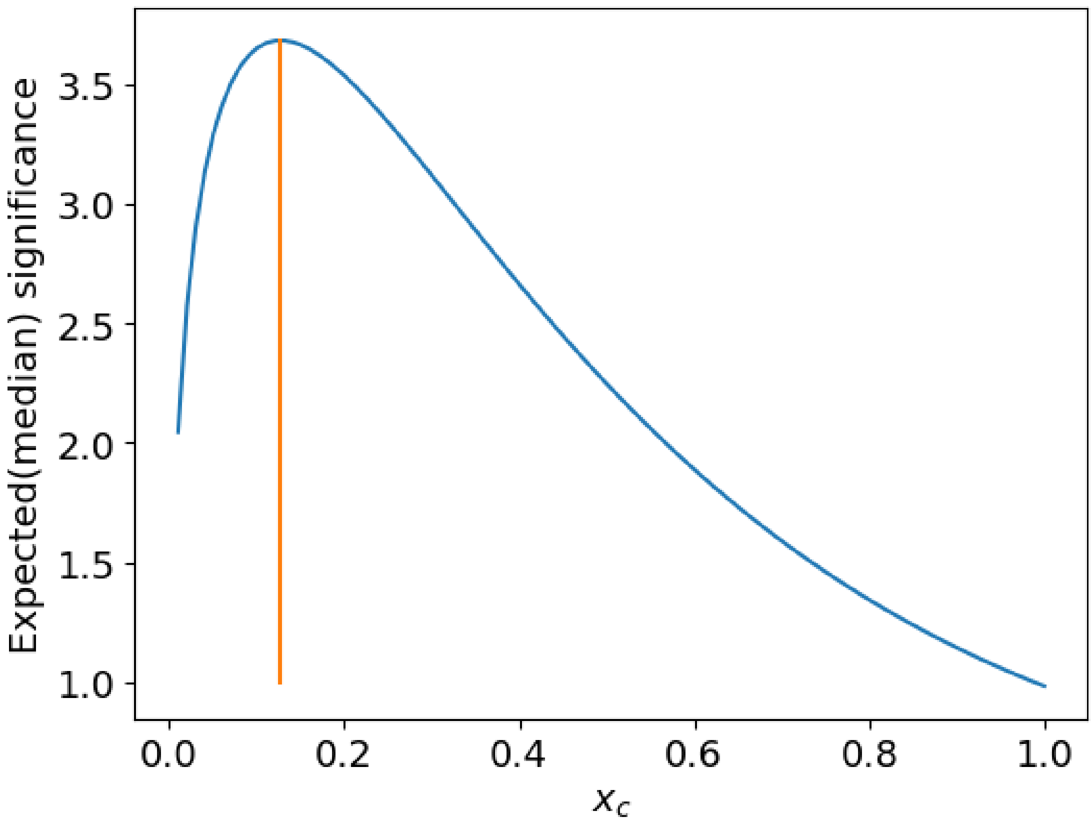
```
0.12692547022457892
```

Verify

[49]:
```
xc = np.linspace(0,1,100)
plt.plot(xc, ex_sig_gen(xc))
plt.plot([xc_max,xc_max],[1,ex_sig_gen(xc_max)])
plt.xlabel(r'$x_c$')
plt.ylabel("Expected(median) significance")
```

```
/tmp/ipykernel_7550/2766341654.py:13: RuntimeWarning: invalid value encountered
in divide
  term2 = np.log(1+(s/b))
```

[49]:
```
Text(0, 0.5, 'Expected(median) significance')
```


Expected(median) significance
3.5
3.0
2.5
2.0
1.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
x_c