sc-2021-ccu-lanl-statistics_paper_functions

May 26, 2021

1 Paper Functions

Nathan DeBardeleben, ndebard@lanl.gov, HPC-DES, ~March, 2021

These are Python functions which produce the plots for the SC2021 LANL/CCU statistics paper on acceptance testing.

[1]:
```
from scipy.stats import poisson
from scipy.stats import gamma
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.optimize import fmin
from numpy import array
import pandas as pd
import seaborn as sns
from scipy.optimize import minimize
```

[2]:
```
def pff_calculator(x, F, pff):
    # in R
    # abs(1-ppois(q=q,lambda=x)-pff)
    return abs(1 - poisson.cdf(F, x) - pff)
def one_minus_poisson_cfd(x, F):
    return abs(1 - poisson.cdf(F, x))
def pfp_calculator(x, F, pfp):
    # in R
    # abs(ppois(q=q,lambda=x)-pfp)
    return abs(poisson.cdf(F, x) - pfp)
def poisson_cfd(x, F):
    return abs(poisson.cdf(F, x))
def two_party_optimization(x, F, factor, pfp, pff):
    # in R
    # abs(1-ppois(q=quse,lambda=x)-pff) + abs(ppois(q=quse,lambda=fac*x)-pfp)
    return ( abs(1 - poisson.cdf(F, x) - pff) + abs(poisson.cdf(F, x * factor)␣
 ↪- pfp) )
```

2 Plotting Parameters

Here we define some parameters for plotting.

3 Parameters of Interest

Here we set our initial constraints.

[3]:
```
pff = 0.1 # probability of false failure
pfp = 0.1 # probability of false pass
F = 1 # failures
mu = 24 # hours
factor = 2
```

4 Probability of False Fail

This is related to the vendor (producer) failing a test that they should have passed. They want to minimize that happening.

4.1 Find the Min Theta

First, we need to find the min theta for these above constraints.

[4]:
```
func = pff_calculator
ret = fmin(func, F/2, args=(F, pff), xtol = 0.00001, full_output=True)
the_min_x = ret[0][0]
the_y_val = ret[1]
print(f"The min of function '{func.__name__}' is at {the_min_x} with result␣
 ↪{the_y_val}.")
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
         Function evaluations: 26
The min of function 'pff_calculator' is at 0.5318115234374999 with result
2.6544168779674138e-08.
```

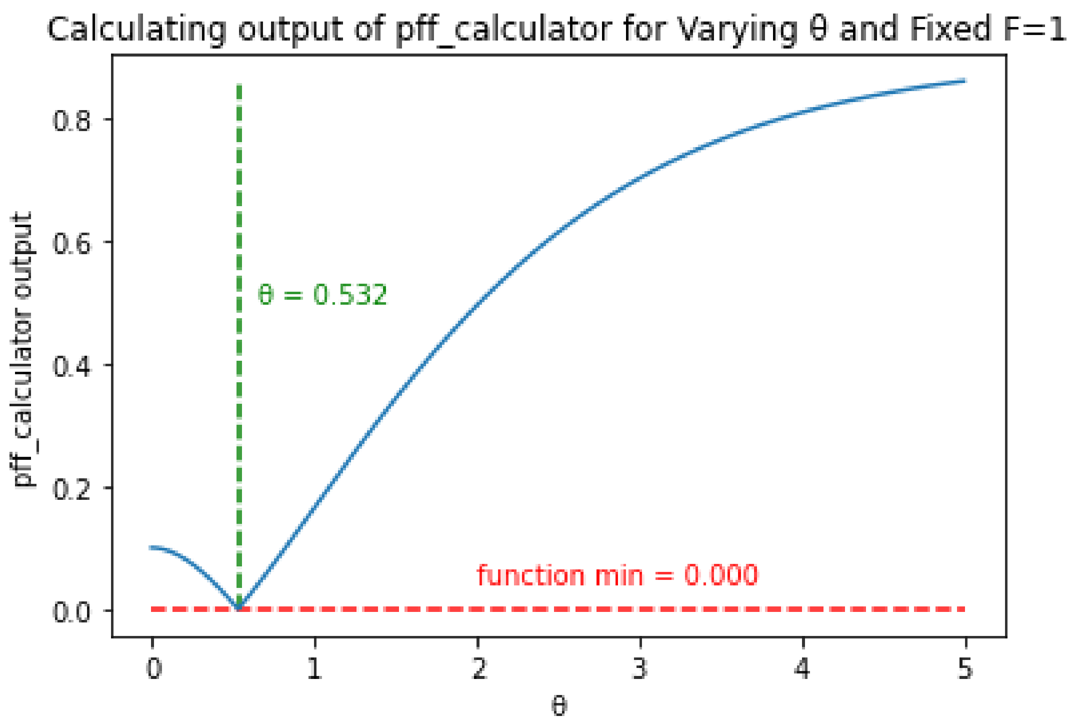
Set up our range of x values, based on what we know the min is - this is just for pretty plotting.

[5]:
```
xmin = 0
xmax = math.floor(the_min_x) * 2
if xmax < 5:
    xmax = 5
points = 10000
# generate x points
xlist = np.linspace(xmin, xmax, points)
```

4.1.1 Plot the Min

This plots the function and annotates the min

[6]:
```python
func = pff_calculator

# map the x points to the values from the above function
ylist = list(map(lambda x: func(x, F=F, pff=pff), xlist))

# plot the results, w/ annotations
plt.plot(xlist, ylist)

# # draw horizontal line at min of function
plt.hlines(y=the_y_val, xmin=min(xlist), xmax=max(xlist), color='red', linestyles="--")
plt.annotate(f"function min = {the_y_val:.3f}", xy=(2, the_y_val + max(ylist) * 0.05), color='red')

# draw vertical line at min
plt.vlines(x=the_min_x, ymin=min(ylist), ymax=max(ylist), color='green', linestyles='--')
plt.annotate(f"  = {the_min_x:.3f}", xy=(the_min_x + max(xlist) * 0.025, 0.5), color='green')

# labels and title
plt.xlabel(" ")
plt.ylabel(f"{func.__name__} output")
plt.title(f"Calculating output of {func.__name__} for Varying   and Fixed F={F}")
plt.savefig(f"NOT_USED_IN_PAPER_pff_calculator_f{F}_pff{pff}.png")
plt.savefig(f"NOT_USED_IN_PAPER_pff_calculator_f{F}_pff{pff}.pdf")
```



4.1.2 Show the Min With Constraints

Going back to our initial distribution, show this point crossing the constraint

[7]:
```
func = one_minus_poisson_cfd

# map the x points to the values from the above function
ylist = list(map(lambda x: func(x, F), xlist))
```

[8]:
```
df = pd.DataFrame()
df['x'] = xlist
df['pff'] = ylist
df.columns = ['theta', '$P_{FF}$']
df.head()
```

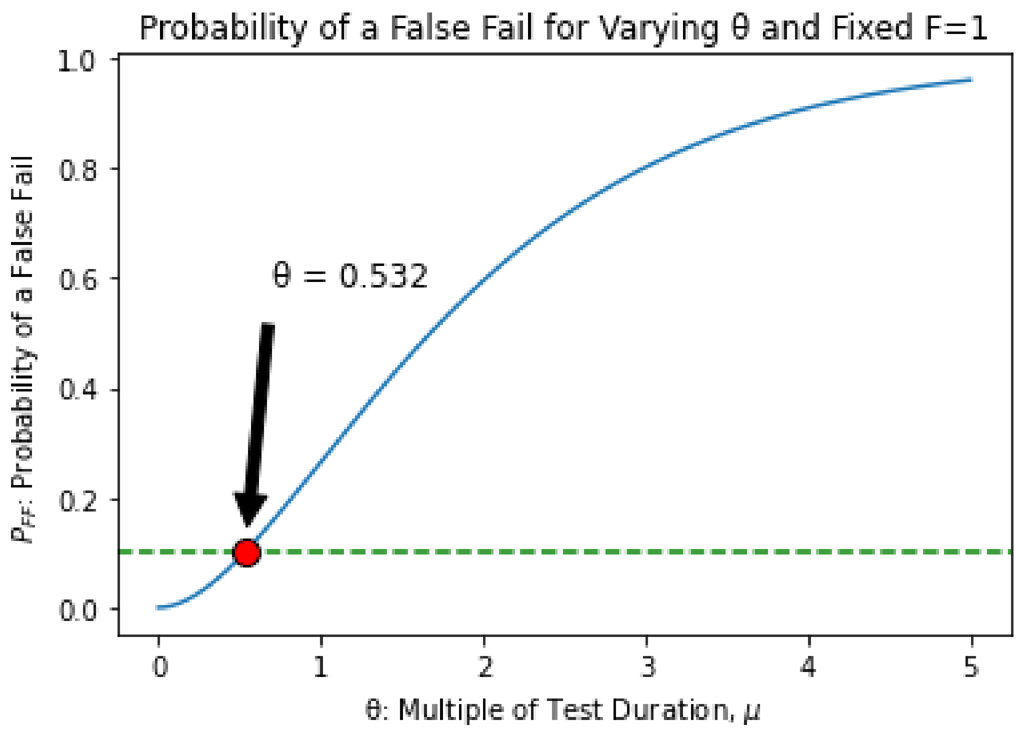
[8]:

	theta	P_{FF}
0	0.0000	0.000000e+00
1	0.0005	1.249833e-07
2	0.0010	4.997667e-07
3	0.0015	1.124100e-06
4	0.0020	1.997735e-06

[9]:
```
fig, ax = plt.subplots(figsize=(6, 4))
ax = sns.lineplot(data=df, x='theta', y='$P_{FF}$')
```

```
ax.axhline(pfp, ls='--', color='green')
ax.annotate(text=f"  = {the_min_x:.3f}",
            xy=(the_min_x, the_y_val + pfp),
            xycoords='data',
            xytext=(10, 100),
            fontsize=12,
            textcoords='offset points',
            arrowprops=dict(facecolor='black', shrink=0.1)
           )
ax.plot([the_min_x],[the_y_val + pfp], 'ro', mec='black', ms=10, linewidth=5)
plt.xlabel(" : Multiple of Test Duration, $\mu$") # matplotlib doesn't like $\theta$ for some reason, wtf?
plt.ylabel("$P_{FF}$: Probability of a False Fail")
plt.title("Probability of a False Fail for Varying   and Fixed F=1")
fn = f"FIGURE_2_prob_false_fail_{F}_pff{pff}"
plt.savefig(f"{fn}.pdf")
plt.savefig(f"{fn}.png")
print(f"{fn}.png and .pdf created")
```

```
FIGURE_2_prob_false_fail_1_pff0.1.png and .pdf created
```



4.2 Result

These are the end results for this section:

```
[10]: print(f"The vendor / producer:\n" \
      f"  To minimize false failure, with acceptable probability of false␣
 ↪failure = {pff}\n" \
      f"  With mu (MTBF) of {mu} (hours)\n" \
      f"  And testing until <= {F} failures are observed\n" \
      f"  Wants to test for {the_min_x:.3f} * {mu} or {(the_min_x * mu):.3f}␣
 ↪hours")
```

```
The vendor / producer:
  To minimize false failure, with acceptable probability of false failure = 0.1
  With mu (MTBF) of 24 (hours)
  And testing until <= 1 failures are observed
  Wants to test for 0.532 * 24 or 12.763 hours
```

5 Probability of False Pass

This is related to the buyer (consumer) failing to reject a system that is actually bad. They want to minimize that happening.

5.1 Find the Min Theta

First, we need to find the min theta for these above constraints.

```
[11]: func = pfp_calculator
ret = fmin(func, F/2, args=(F, pfp), xtol = 0.00001, full_output=True)
the_min_x = ret[0][0]
the_y_val = ret[1]
print(f"The min of function '{func.__name__}' is at {the_min_x} with result␣
 ↪{the_y_val}.")
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 25
         Function evaluations: 50
The min of function 'pfp_calculator' is at 3.889721679687507 with result
1.2010450728405786e-07.
```

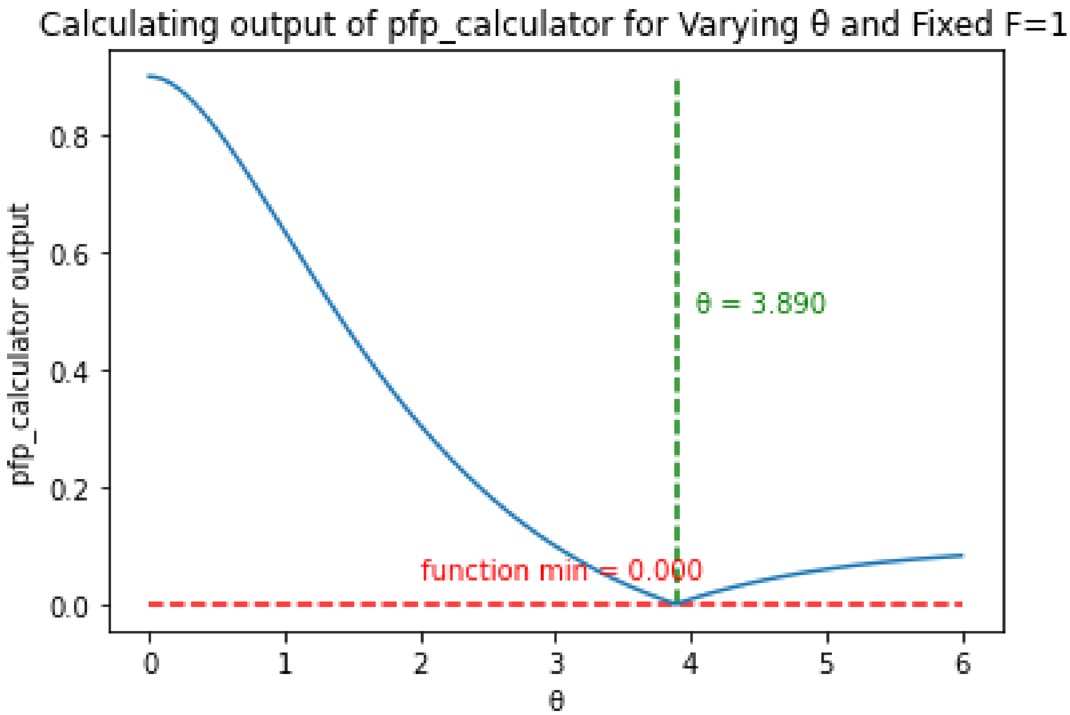
Set up our range of x values, based on what we know the min is - this is just for pretty plotting.

```
[12]: xmin = 0
xmax = math.floor(the_min_x) * 2
if xmax < 5:
    xmax = 5
points = 10000
# generate x points
xlist = np.linspace(xmin, xmax, points)
```

5.1.1 Plot the Min

This plots the function and annotates the min

[13]:
```python
func = pfp_calculator

# map the x points to the values from the above function
ylist = list(map(lambda x: func(x, F=F, pfp=pfp), xlist))

# plot the results, w/ annotations
plt.plot(xlist, ylist)

# # draw horizontal line at min of function
plt.hlines(y=the_y_val, xmin=min(xlist), xmax=max(xlist), color='red', linestyles="--")
plt.annotate(f"function min = {the_y_val:.3f}", xy=(2, the_y_val + max(ylist) * 0.05), color='red')

# draw vertical line at min
plt.vlines(x=the_min_x, ymin=min(ylist), ymax=max(ylist), color='green', linestyles='--')
plt.annotate(f"  = {the_min_x:.3f}", xy=(the_min_x + max(xlist) * 0.025, 0.5), color='green')

# labels and title
plt.xlabel(" ")
plt.ylabel(f"{func.__name__} output")
plt.title(f"Calculating output of {func.__name__} for Varying   and Fixed F={F}")
plt.savefig(f"NOT_USED_IN_PAPER_pfp_calculator_f{F}_pfp{pfp}.png")
plt.savefig(f"NOT_USED_IN_PAPER_pfp_calculator_f{F}_pfp{pfp}.pdf")
```



5.1.2 Show the Min With Constraints

Going back to our initial distribution, show this point crossing the constraint

[14]:
```
func = poisson_cfd

# map the x points to the values from the above function
ylist = list(map(lambda x: func(x, F), xlist))
```

[15]:
```
df = pd.DataFrame()
df['x'] = xlist
df['pff'] = ylist
df.columns = ['theta', '$P_{FP}$']
df.head()
```

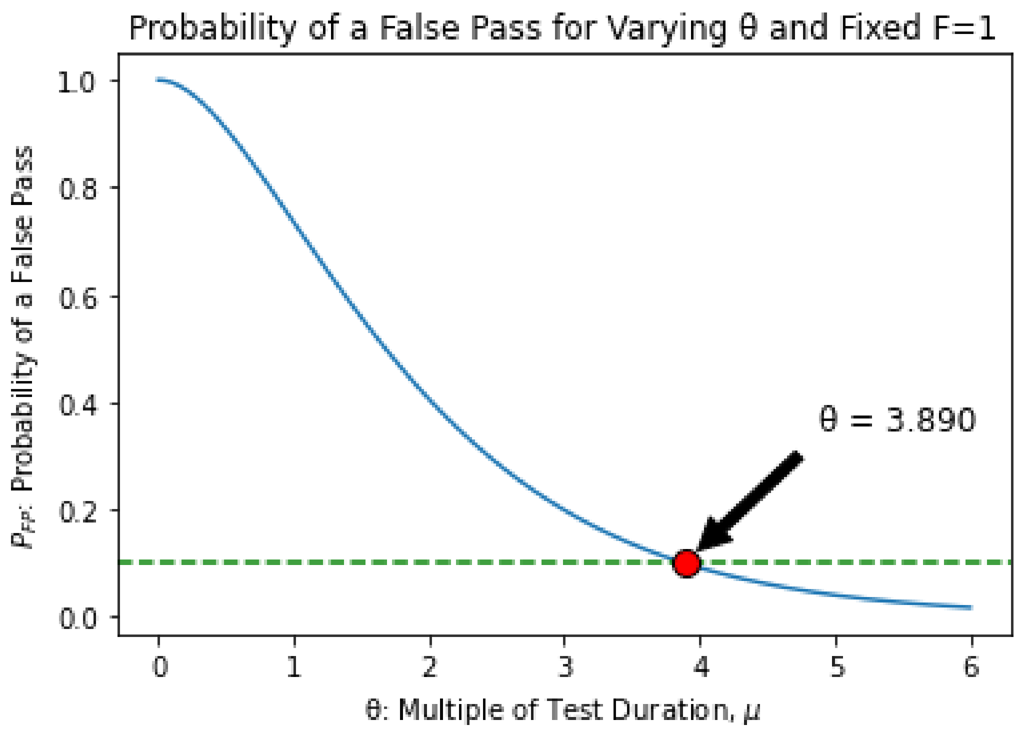
[15]:

	theta	P_{FP}
0	0.0000	1.000000
1	0.0006	1.000000
2	0.0012	0.999999
3	0.0018	0.999998
4	0.0024	0.999997

[16]:
```
fig, ax = plt.subplots(figsize=(6, 4))
ax = sns.lineplot(data=df, x='theta', y='$P_{FP}$')
```

```
ax.axhline(pfp, ls='--', color='green')
ax.annotate(text=f" = {the_min_x:.3f}",
            xy=(the_min_x, the_y_val + pfp),
            xycoords='data',
            xytext=(50, 50),
            fontsize=12,
            textcoords='offset points',
            arrowprops=dict(facecolor='black', shrink=0.1)
           )
ax.plot([the_min_x],[the_y_val + pfp], 'ro', mec='black', ms=10, linewidth=5)
plt.xlabel(" : Multiple of Test Duration, $\mu$") # matplotlib doesn't like␣
↪$\theta$ for some reason, wtf?
plt.ylabel("$P_{FP}$: Probability of a False Pass")
plt.title("Probability of a False Pass for Varying  and Fixed F=1")
fn = f"FIGURE_1_prob_false_pass_{F}_pfp{pfp}"
plt.savefig(f"{fn}.pdf")
plt.savefig(f"{fn}.png")
print(f"{fn}.png and .pdf created")
```

```
FIGURE_1_prob_false_pass_1_pfp0.1.png and .pdf created
```



5.2 Result

These are the end results for this section:

[17]:
```
print(f"The buyer / consumer:\n" \
      f"  To minimize false pass, with acceptable probability of false pass =␣
 ↪{pfp}\n" \
      f"  With mu (MTBF) of {mu} (hours)\n" \
      f"  And testing until <= {F} failures are observed\n" \
      f"  Wants to test for {the_min_x:.3f} * {mu} or {(the_min_x * mu):.3f}␣
 ↪hours")
```

```
The buyer / consumer:
  To minimize false pass, with acceptable probability of false pass = 0.1
  With mu (MTBF) of 24 (hours)
  And testing until <= 1 failures are observed
  Wants to test for 3.890 * 24 or 93.353 hours
```

6 Constrain PFF and Match PFP with `factor` Tolerance

This code locks PFF at the input value and then plots PFP and the intersection of these curves

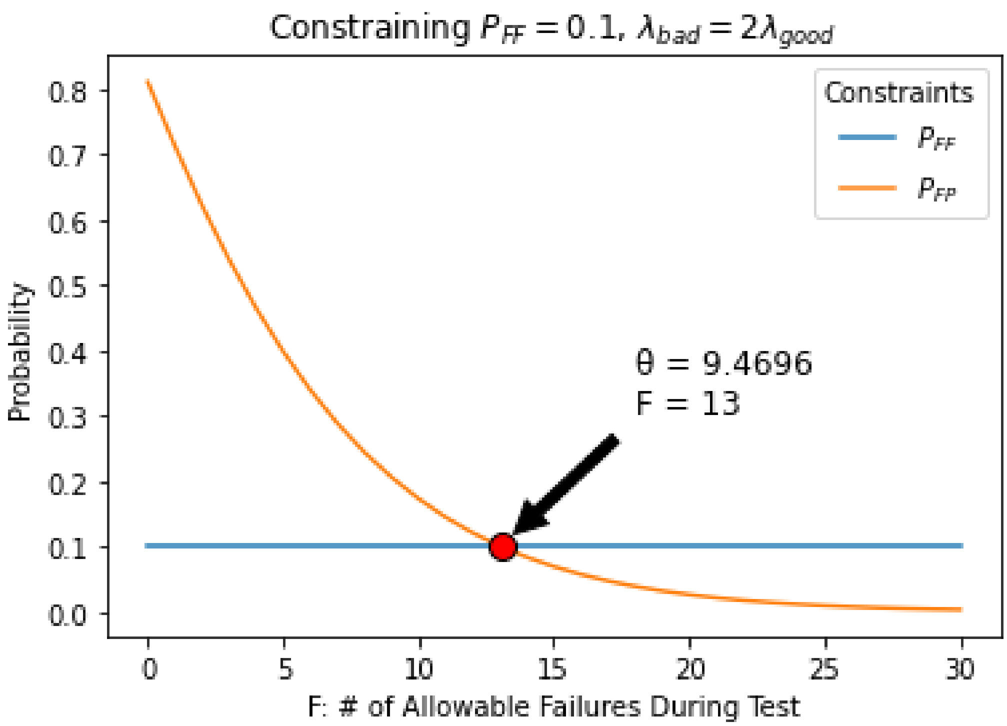
[18]:
```
# some helper code to find intersection of columns from the dataframe
def find_intersection_of_two_curves(x, y1, y2, deg):
    y1_fit = np.polyfit(x, y1, deg)
    y1_func = np.poly1d(y1_fit)

    y2_fit = np.polyfit(x, y2, deg)
    y2_func = np.poly1d(y2_fit)

    from scipy.optimize import fsolve
    def findIntersection(fun1,fun2,x0):
        return fsolve(lambda x : fun1(x) - fun2(x), x0)

    result = findIntersection(y1_func, y2_func, 0.0)
    return (result[0],y1_func(result[0]))
```

[19]:
```
def constrain_pff_find_pfp(pff, factor, title_prepend):
    # generate the dataframe with F, theta, and values of PFF, PFP (where PFP␣
 ↪has a `factor` tolerance)
    d = dict()
    for Ftemp in range(0, 31):
        d[Ftemp] = list()
        d[Ftemp].append(Ftemp)
        res = minimize(fun=pff_calculator, args=(Ftemp, pff), x0=Ftemp/2, tol=0.
 ↪0000001, method='Nelder-Mead')
        d[Ftemp].append(res['x'][0])
        d[Ftemp].append(one_minus_poisson_cfd(res['x'][0], Ftemp))
        d[Ftemp].append(poisson_cfd(res['x'] * factor, Ftemp)[0])
```

```
    df = pd.DataFrame.from_dict(d, orient='index',␣
↪columns=['F','theta','$P_{FF}$','$P_{FP}$'])
    intersection_x,intersection_y = find_intersection_of_two_curves(df['F'].
↪to_numpy(), df['$P_{FF}$'].to_numpy(), df['$P_{FP}$'].to_numpy(), 10)
    print(f"These curves intersect at x={intersection_x}, y={intersection_y}")
    idx = df['F'].sub(intersection_x).abs().idxmin()
    theta_at_intersection = df.iloc[idx]['theta']
    print(f"Theta value at this intersection is {theta_at_intersection}")

    # melt (transform) the dataframe for plotting
    df2 = df.melt(id_vars=['F','theta'], var_name='Constraints',␣
↪value_name='Probability')

    # and plot it
    fig, ax = plt.subplots(figsize=(6,4))
    ax = sns.lineplot(data=df2, x='F', y='Probability', hue='Constraints')
    ax.annotate(text=f"  = {theta_at_intersection:.4f}\nF = {df.iloc[idx]['F']:.
↪0f}",
                xy=(intersection_x, intersection_y),
                xycoords='data',
                xytext=(50, 50),
                fontsize=12,
                textcoords='offset points',
                arrowprops=dict(facecolor='black', shrink=0.1)
               )
    ax.plot([intersection_x],[intersection_y], 'ro', mec='black', ms=10,␣
↪linewidth=5)
    plt.xlabel("F: # of Allowable Failures During Test")
    plt.title("Constraining $P_{FF} = 0.1$, $\lambda_{bad} = %s\lambda_{good}$"␣
↪% factor)
    fn = f"{title_prepend}constrained_pff{pff}_W{factor}"
    plt.savefig(f"{fn}.png");
    plt.savefig(f"{fn}.pdf");
    print(f"{fn}.png and .pdf created")
```

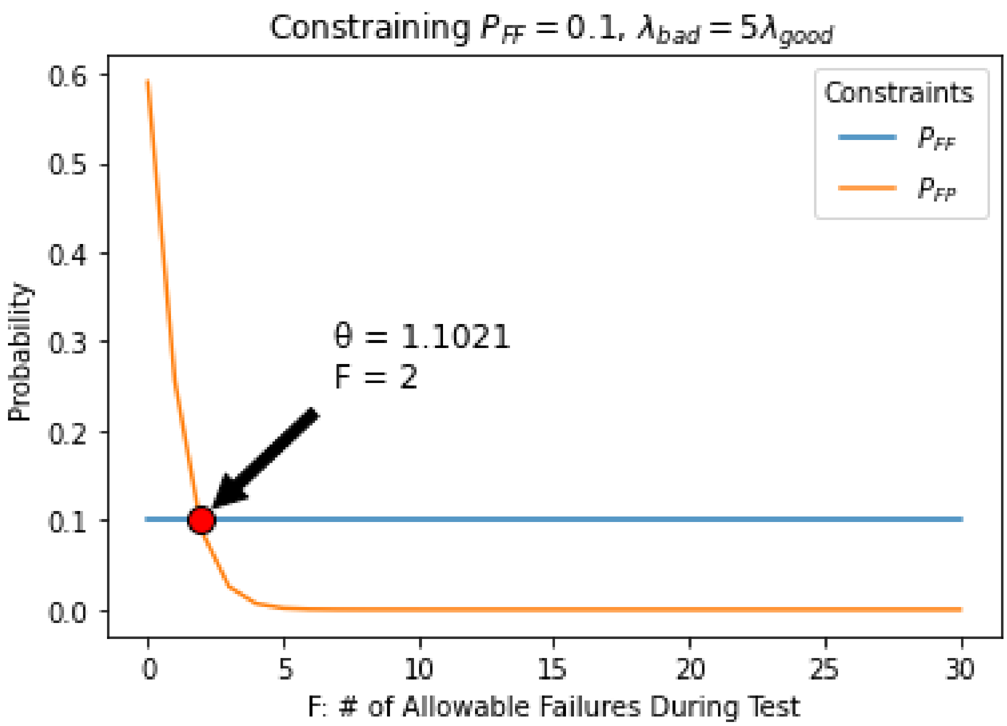
[20]:
```
constrain_pff_find_pfp(pff=0.1, factor=2, title_prepend="FIGURE_3_")
```

```
These curves intersect at x=13.038587393880091, y=0.0999999998808488
Theta value at this intersection is 9.469621187448507
FIGURE_3_constrained_pff0.1_W2.png and .pdf created
```



[21]:
```
constrain_pff_find_pfp(pff=0.1, factor=5, title_prepend="NOT_USED_IN_PAPER_")
```

```
These curves intersect at x=1.9284141502834966, y=0.10000000532803094
Theta value at this intersection is 1.1020653724670408
NOT_USED_IN_PAPER_constrained_pff0.1_W5.png and .pdf created
```



7 Optimize to Both Parties' Goals

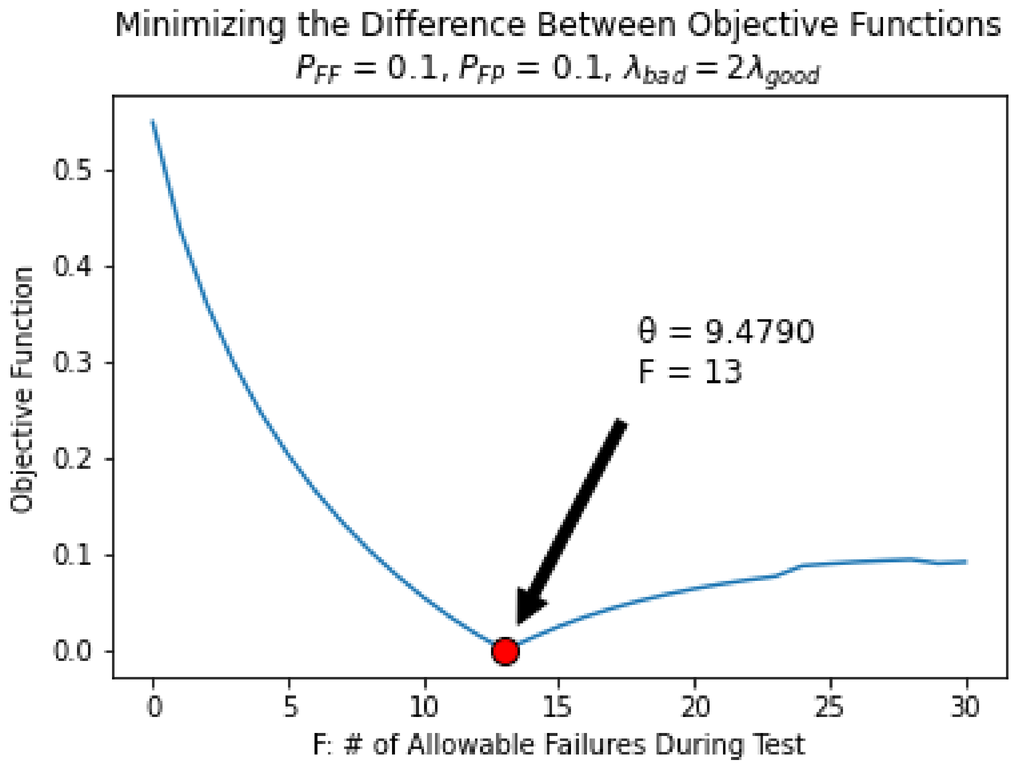
The previous section assumed that PFP would be interested in the same goal as PFF, here we allow them to vary

[22]:
```
def constraint_pff_and_pfp_optimize(pff, pfp, factor, Fmax, degree_poly,␣
↪title_prepend):
    # generate the dataframe with F, theta, and values of PFF, PFP (where PFP␣
↪has a `factor` tolerance)
    d = dict()
    for Ftemp in range(0, Fmax):
        d[Ftemp] = list()
        d[Ftemp].append(Ftemp)
        res = minimize(fun=two_party_optimization, args=(Ftemp, factor, pfp,␣
↪pff), x0=Ftemp/2, tol=0.0000001, method='Nelder-Mead')
        d[Ftemp].append(res['x'][0])
        d[Ftemp].append(res['fun'])

    df = pd.DataFrame.from_dict(d, orient='index',␣
↪columns=['F','theta','objective_function'])
    #display(df)
```

```python
    obj_fit = np.polyfit(df['F'].to_numpy(), df['objective_function'].to_numpy(), degree_poly)
    obj_func = np.poly1d(obj_fit)
    min_res = minimize(obj_func, x0=0, method='Nelder-Mead')
    min_value = min_res['x'][0]
    print(min_value)
    idx = df['F'].sub(min_value).abs().idxmin()
    theta_at_min = df.iloc[idx]['theta']
    f_at_min = df.iloc[idx]['F']
    obj_at_min = df.iloc[idx]['objective_function']
    print(f"F value at this min is {f_at_min}")
    print(f"Theta value at this min is {theta_at_min}")
    print(f"Objective value at this min is {obj_at_min}")

    fig, ax = plt.subplots(figsize=(6,4))
    ax = sns.lineplot(data=df, x='F', y='objective_function')

    ax.annotate(text=f"  = {theta_at_min:.4f}\nF = {f_at_min:.0f}",
                xy=(f_at_min, obj_at_min),
                xycoords='data',
                xytext=(50, 100),
                fontsize=12,
                textcoords='offset points',
                arrowprops=dict(facecolor='black', shrink=0.1)
               )
    ax.plot([f_at_min],[obj_at_min], 'ro', mec='black', ms=10, linewidth=5)

    #plt.xticks(range(0, Fmax))
    plt.ylabel("Objective Function")
    plt.xlabel("F: # of Allowable Failures During Test")

    plt.title("Minimizing the Difference Between Objective Functions\n$P_{FF}$ = %s, $P_{FP}$ = %s, $\lambda_{bad} = %s\lambda_{good}$" % (pff, pfp, factor));
    fn = f"{title_prepend}two_party_pff{pff}_pfp{pfp}_W{factor}"
    plt.savefig(f"{fn}.png");
    plt.savefig(f"{fn}.pdf");
    print(f"{fn}.png and .pdf created")
```

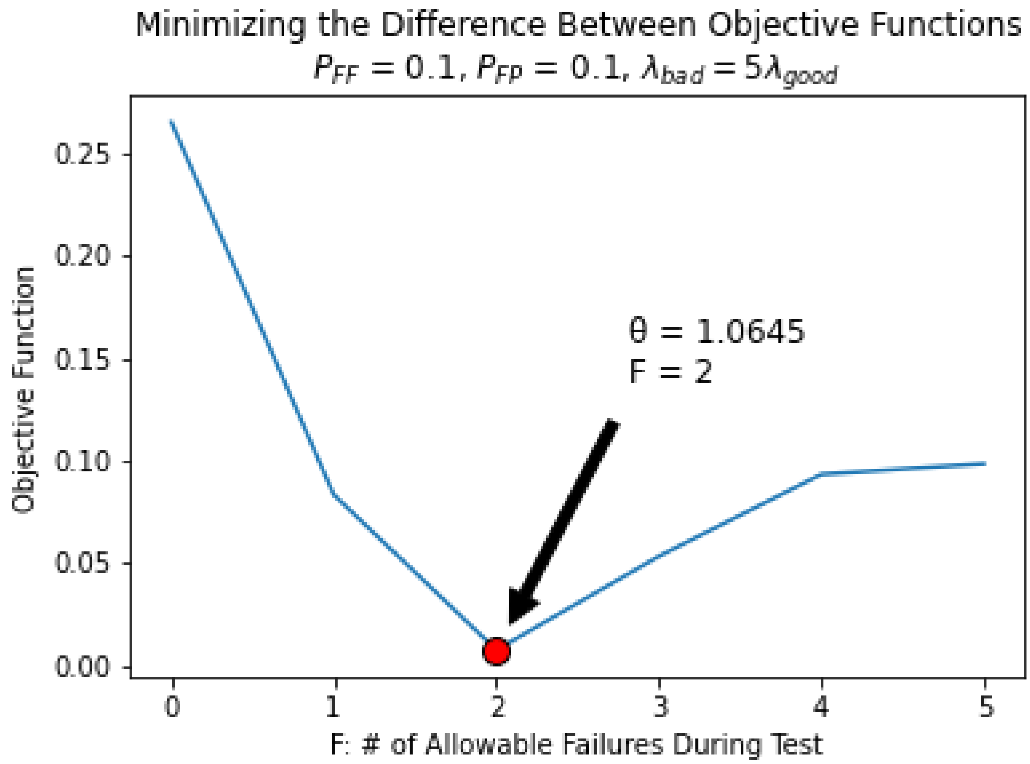
[23]:
```python
constraint_pff_and_pfp_optimize(pff=0.1, pfp=0.1, factor=2, Fmax=31, degree_poly=10, title_prepend="FIGURE_4_")
```

```
13.316687500000013
F value at this min is 13.0
Theta value at this min is 9.478980571031574
Objective value at this min is 0.0005720677095657073
FIGURE_4_two_party_pff0.1_pfp0.1_W2.png and .pdf created
```




```
[24]: constraint_pff_and_pfp_optimize(pff=0.1, pfp=0.1, factor=5, Fmax=6,␣
       ↪degree_poly=4, title_prepend="FIGURE_7_")
```

```
2.1060000000000025
F value at this min is 2.0
Theta value at this min is 1.0644640922546391
Objective value at this min is 0.007467249196865769
FIGURE_7_two_party_pff0.1_pfp0.1_W5.png and .pdf created
```



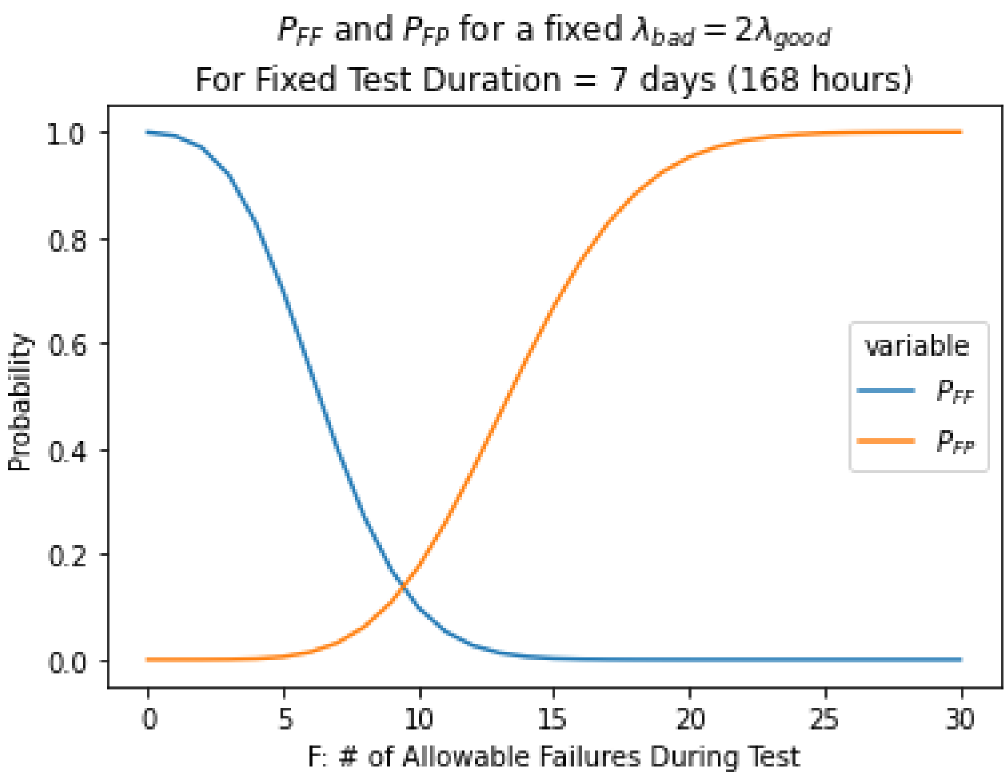
8 Constraining T

Try and fix T and see if we can optimize

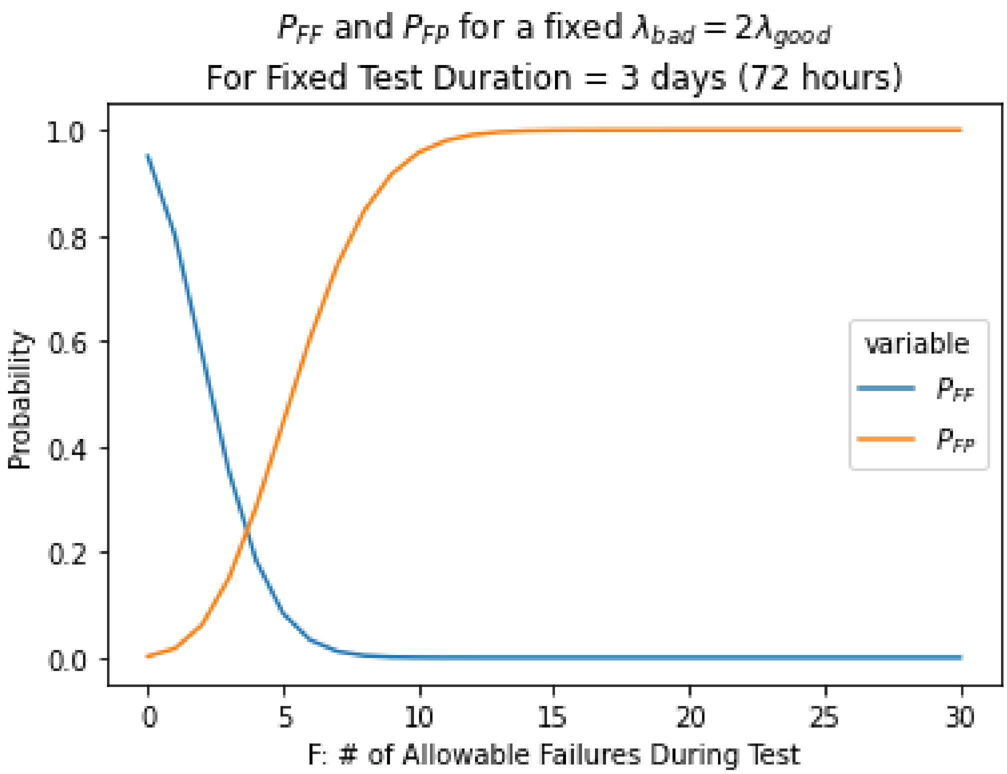
[25]:
```
def constrained_time(T, W, title_prepend):
    T_hours = T * 24
    # generate the dataframe with F, theta, and values of PFF, PFP (where PFP␣
↪has a `factor` tolerance)
    d = dict()
    for Ftemp in range(0, 31):
        d[Ftemp] = list()
        d[Ftemp].append(Ftemp)
        pff = 1 - poisson.cdf(Ftemp, T_hours / 24)
        pfp = poisson.cdf(Ftemp, W * T_hours / 24)
        d[Ftemp].append(pff)
        d[Ftemp].append(pfp)

    df = pd.DataFrame.from_dict(d, orient='index',␣
↪columns=['F','$P_{FF}$','$P_{FP}$'])
    df.head()
    df_melted = df.melt(id_vars=['F'])
```

```
    df_melted.head()
    fig, ax = plt.subplots(figsize=(6,4))
    ax = sns.lineplot(data=df_melted, x='F', y='value', hue='variable')
    plt.ylabel("Probability")
    plt.xlabel("F: # of Allowable Failures During Test")
    plt.title("$P_{FF}$ and $P_{FP}$ for a fixed $\lambda_{bad} =␣
↪%s\lambda_{good}$\n" \
              f"For Fixed Test Duration = {T} days ({T_hours} hours)" % factor)

    plt.
↪savefig(f"{title_prepend}fixed_time_interval_THours{T_hours}_F{F}_W{factor}.
↪png");
    plt.
↪savefig(f"{title_prepend}fixed_time_interval_THours{T_hours}_F{F}_W{factor}.
↪pdf");
```

[26]:
```
constrained_time(T=7, W=2, title_prepend="FIGURE_5_")
```



[27]:
```
constrained_time(T=3, W=2, title_prepend="FIGURE_6_")
```



9 FIGURE 8 - TEST DURATION WITH VARYING W

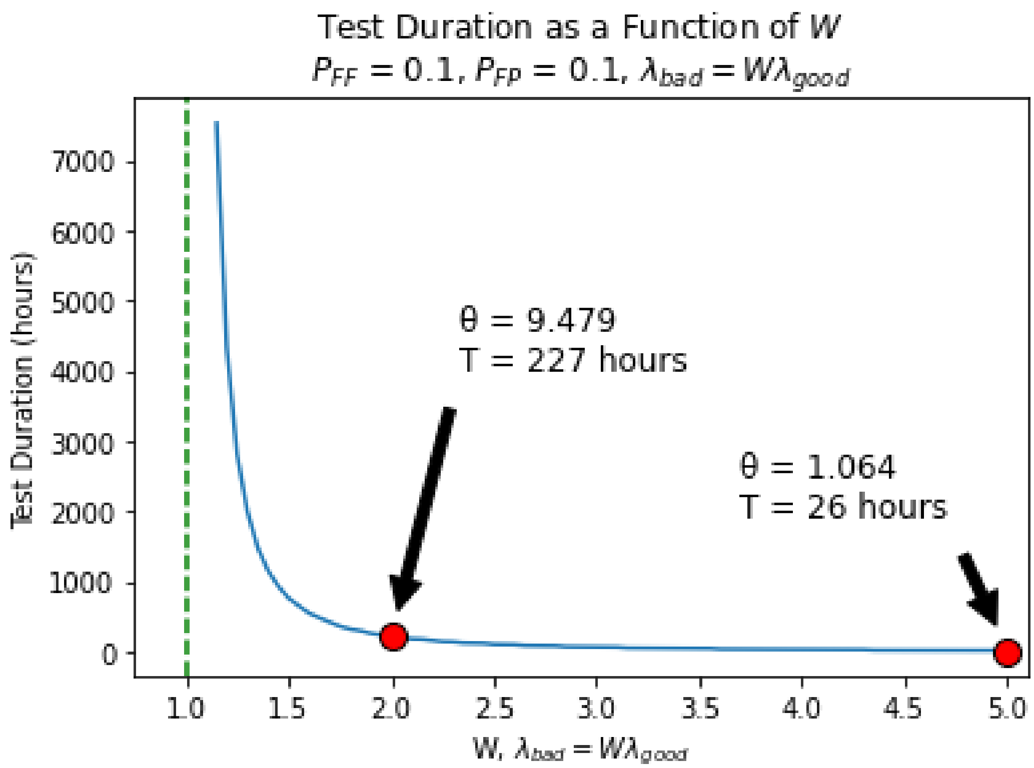
[28]:
```
print("Calculating optimal test duration while sweeping W.")
print("This step takes a moment.  Please be patient . . .")
factor_range = np.linspace(1.15, 5, 78)

pff = 0.1
pfp = 0.1
Fmax = 350

factor_sweep_d = dict()

for factor in factor_range:
    # generate the dataframe with F, theta, and values of PFF, PFP (where PFP␣
 ↪has a `factor` tolerance)
    d = dict()
    for Ftemp in range(0, Fmax):
        d[Ftemp] = list()
        d[Ftemp].append(Ftemp)
        res = minimize(fun=two_party_optimization, args=(Ftemp, factor, pfp,␣
 ↪pff), x0=Ftemp, tol=0.000000001, method='Nelder-Mead')
```

```
        d[Ftemp].append(res['x'][0])
        d[Ftemp].append(res['fun'])

    df = pd.DataFrame.from_dict(d, orient='index', columns=['F','theta','objective_function'])

    factor_sweep_d[factor] = df.query(f"objective_function == {df['objective_function'].min()}")['theta'].values[0]
    print(f"factor = {factor}, theta = {factor_sweep_d[factor]}")

factor_sweep_df = pd.DataFrame.from_dict(factor_sweep_d, orient='index').reset_index()
factor_sweep_df.columns = ['W', 'theta']
factor_sweep_df['time'] = factor_sweep_df['theta'] * (mu) # hours
factor_sweep_df

fig, ax = plt.subplots(figsize=(6,4))
ax = sns.lineplot(data=factor_sweep_df, x='W', y='time')



theta_at_2 = factor_sweep_df[factor_sweep_df['W'] == 2.0]['time'].values[0]
theta_at_5 = factor_sweep_df[factor_sweep_df['W'] == 5.0]['time'].values[0]

ax.annotate(text=f"  = {theta_at_2 / 24:.3f}\nT = {theta_at_2:.0f} hours",
            xy=(2, theta_at_2),
            xycoords='data',
            xytext=(25, 100),
            fontsize=12,
            textcoords='offset points',
            arrowprops=dict(facecolor='black', shrink=0.1)
           )
ax.plot([2.0],[theta_at_2], 'ro', mec='black',  ms=10, linewidth=5)

ax.annotate(text=f"  = {theta_at_5 / 24:.3f}\nT = {theta_at_5:.0f} hours",
            xy=(5, theta_at_5),
            xycoords='data',
            xytext=(-100, 50),
            fontsize=12,
            textcoords='offset points',
            arrowprops=dict(facecolor='black', shrink=0.2)
           )
ax.plot([5.0],[theta_at_5], 'ro', mec='black', ms=10, linewidth=5)

# #plt.xticks(range(0, Fmax))

ax.axvline(1.0, ls='--', color='green')
```

```
plt.xlim((0.75, 5.1))
plt.ylabel("Test Duration (hours)")
plt.xlabel("W, $\lambda_{bad} = W\lambda_{good}$")
plt.title("Test Duration as a Function of $W$\n$P_{FF}$ = %s, $P_{FP}$ = %s,␣
 ↪$\lambda_{bad} = W\lambda_{good}$" % (pff, pfp));
fn = f"test_duration_varying_W_pff{pff}_pfp{pfp}"
plt.savefig(f"FIGURE_8_{fn}.png");
plt.savefig(f"FIGURE_8_{fn}.pdf");
print(f"{fn}.png and .pdf created")
```

```
Calculating optimal test duration while sweeping W.
This step takes a moment.  Please be patient . . .
factor = 1.15, theta = 313.6781580199487
factor = 1.2, theta = 180.19464774465888
factor = 1.25, theta = 117.53711794638077
factor = 1.2999999999999998, theta = 83.65477133373497
factor = 1.3499999999999999, theta = 62.32690597102042
factor = 1.4, theta = 48.53500800803301
factor = 1.45, theta = 39.3554163391469
factor = 1.5, theta = 32.19273453829808
factor = 1.5499999999999998, theta = 27.58936861846596
factor = 1.5999999999999999, theta = 23.24906428726389
factor = 1.65, theta = 20.506904896814355
factor = 1.7, theta = 17.91370794940739
factor = 1.75, theta = 15.454343557357788
factor = 1.7999999999999998, theta = 13.753494395501907
factor = 1.85, theta = 12.76004699971527
factor = 1.9, theta = 11.20651186304167
factor = 1.95, theta = 10.322057368606318
factor = 2.0, theta = 9.478980635805051
factor = 2.05, theta = 8.673944212496274
factor = 2.1, theta = 7.9038676876574705
factor = 2.15, theta = 7.720056811533864
factor = 2.2, theta = 7.003018714487553
factor = 2.25, theta = 6.3137734626419775
factor = 2.3, theta = 6.176517519168548
factor = 2.35, theta = 5.52966448515653
factor = 2.4, theta = 5.414463143050666
factor = 2.45, theta = 5.303963895142074
factor = 2.5, theta = 4.708365784585474
factor = 2.55, theta = 4.616044887155291
factor = 2.6, theta = 4.5272747935727224
factor = 2.65, theta = 3.974366832152004
factor = 2.7, theta = 3.9007674464955877
factor = 2.75, theta = 3.8298444025218448
factor = 2.8, theta = 3.7614543244242604
factor = 2.85, theta = 3.2542715407907963
```

```
factor = 2.9, theta = 3.1981634106487036
factor = 2.95, theta = 3.1439572526142
factor = 3.0, theta = 3.091557964682579
factor = 3.05, theta = 3.0408766865730286
factor = 3.1, theta = 2.9918302884325385
factor = 3.15, theta = 2.5376474872231474
factor = 3.2, theta = 2.497996745258568
factor = 3.25, theta = 2.4595660261809815
factor = 3.3, theta = 2.4222998753190015
factor = 3.35, theta = 2.3861461453139765
factor = 3.4, theta = 2.3510557606816285
factor = 3.45, theta = 2.3169824890792357
factor = 3.5, theta = 2.2838827393949
factor = 3.5500000000000003, theta = 2.2517153769731486
factor = 3.6, theta = 1.8557730741798855
factor = 3.65, theta = 1.8303515251725881
factor = 3.7, theta = 1.8056170452386109
factor = 3.75, theta = 1.7815421512350413
factor = 3.8000000000000003, theta = 1.7581008072942468
factor = 3.85, theta = 1.7352683298289737
factor = 3.9, theta = 1.7130212998017638
factor = 3.95, theta = 1.6913374856114352
factor = 4.0, theta = 1.6701957676559647
factor = 4.050000000000001, theta = 1.6495760668069088
factor = 4.1, theta = 1.6294592851772873
factor = 4.15, theta = 1.6098272457718816
factor = 4.2, theta = 1.5906626354902962
factor = 4.25, theta = 1.5719489576294978
factor = 4.300000000000001, theta = 1.55367048103362
factor = 4.35, theta = 1.223521916568278
factor = 4.4, theta = 1.2096182584762565
factor = 4.45, theta = 1.1960270419716825
factor = 4.5, theta = 1.1827378526329988
factor = 4.550000000000001, theta = 1.1697407335042946
factor = 4.6, theta = 1.1570261597633351
factor = 4.65, theta = 1.144585018604993
factor = 4.7, theta = 1.1324085816740976
factor = 4.75, theta = 1.1204884916543951
factor = 4.800000000000001, theta = 1.1088167369365682
factor = 4.85, theta = 1.097385637462138
factor = 4.9, theta = 1.086187824606895
factor = 4.95, theta = 1.0752162300050256
factor = 5.0, theta = 1.0644640676677224
test_duration_varying_W_pff0.1_pfp0.1.png and .pdf created
```



10 Bayesian

10.1 Confidence Intervals On Determinging MTBF

11 THIS IS TABLE 2

[29]:
```
def get_gamma_pi_95(alpha_prior, beta_prior, mu, X, T_Days):
    alpha_post = alpha_prior + X
    beta_post = beta_prior + 1
    T_Hours = T_Days * 24
    dist = gamma(alpha_post, scale=1.0/beta_post)
    # can do this if you want to sample it, without analytical results
    #random_variables = T_Hours / (dist.rvs(size=pow(10,7))) # pull 10^7 random␣
 ↪variables
    #np.count_nonzero(random_variables > mu) / len(random_variables))))

    # PI and posterior prob that measured mu >= mu
    return (tuple((T_Hours / dist.ppf(0.975),
                   T_Hours / dist.ppf(0.025),
                   dist.cdf(T_Hours / mu))))
```

[30]:
```
mu = 24
T_Days = 7
alpha_prior = pow(10, -6)
beta_prior = pow(10, -4)
```

[31]:
```
get_gamma_pi_95(alpha_prior, beta_prior, mu, 1, 7)
```

[31]: (45.54681341170713, 6636.281526794217, 0.9990887537137954)

[32]:
```
get_gamma_pi_95(alpha_prior, beta_prior, mu, 2, 7)
```

[32]: (30.155689426283086, 693.6835160314615, 0.9927093990251339)

[33]:
```
get_gamma_pi_95(alpha_prior, beta_prior, mu, 3, 7)
```

[33]: (23.255920643711523, 271.5762949453369, 0.9703794360056592)

[34]:
```
get_gamma_pi_95(alpha_prior, beta_prior, mu, 4, 7)
```

[34]: (19.164085553491788, 154.1628223358991, 0.9182709960049941)

[35]:
```
get_gamma_pi_95(alpha_prior, beta_prior, mu, 5, 7)
```

[35]: (16.405343113241006, 103.49131551158374, 0.8270721298265464)

[36]:
```
get_gamma_pi_95(alpha_prior, beta_prior, mu, 6, 7)
```

[36]: (14.399382756009041, 76.30555109109329, 0.6993809748736854)

[37]:
```
get_gamma_pi_95(alpha_prior, beta_prior, mu, 7, 7)
```

[37]: (12.865509131958868, 59.69974715291811, 0.5503930882919177)

[38]:
```
get_gamma_pi_95(alpha_prior, beta_prior, mu, 8, 7)
```

[38]: (11.64948804895228, 48.646477223965505, 0.40139032420760085)

[39]:
```
get_gamma_pi_95(alpha_prior, beta_prior, mu, 9, 7)
```

[39]: (10.658806633048547, 40.82661975420877, 0.27099988419077115)

[40]:
```
get_gamma_pi_95(alpha_prior, beta_prior, mu, 10, 7)
```

[40]: (9.83428270413837, 35.03715476074176, 0.1695749673017822)

[ ]:
```

```