Theoratical Generation of Data

- Strain Life (Morrow equation):
$$\frac{\Delta\epsilon}{2} = \frac{\sigma'_f}{E}(2N_f)^b + \epsilon'_f(2N_f)^c$$
- Reversal Stress Ratio:
$$\frac{\Delta\sigma}{2} = \sigma_{max} = \sigma'_f(2N_f)^b$$
- SWT(Smith,Watson and Topper fatigue damage parameter):
$$SWT = \sigma_{max}\frac{\Delta\epsilon}{2} = \frac{(\sigma'^2_f)(2N_f)^{2b}}{E} + \sigma'_f\epsilon'_f(2N_f)^{b+c}$$

  where,
  σ^*_f: fatigue strenght coefficient,
  b: fatigue strength exponent,
  ϵ^*_f: fatigue ductility coefficient,
  b: fatigue ductility exponent and
  E: Young modulus

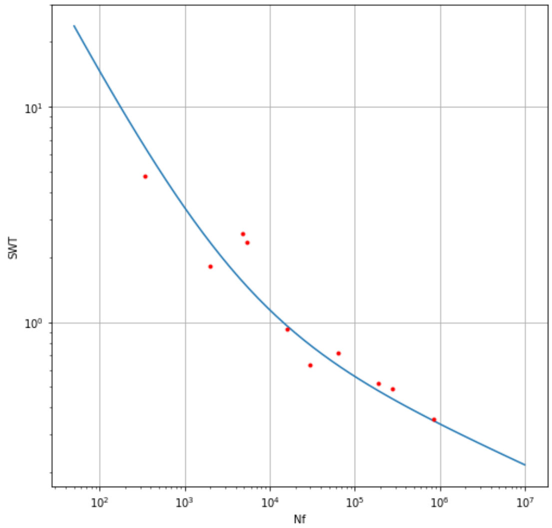
The values of the constants for S355 Mild Steel are:

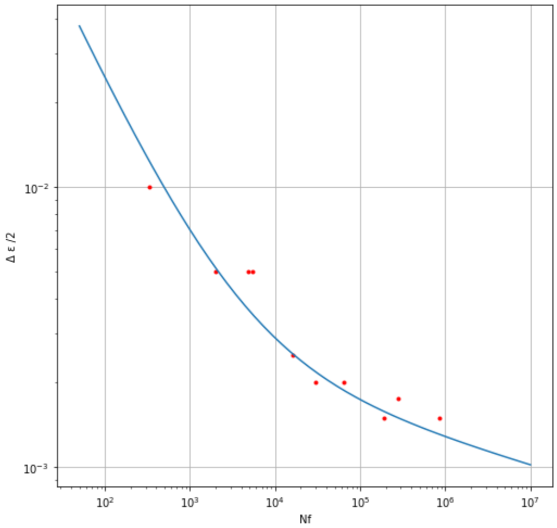
E GPa	σ'_f MPa	b	ϵ'_f	c
211.60	952.20	-0.0890	0.7371	-0.6640

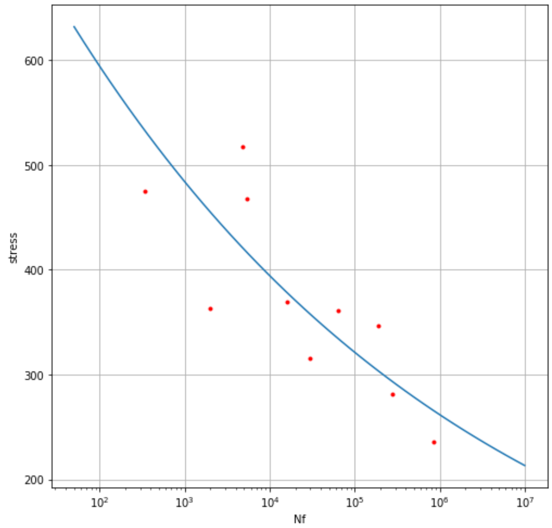
In [32]:

```
import numpy as np
import matplotlib.pyplot as plt

def generate_data(
        Nf=np.linspace(10,10000000,10000),
        E = 211600,
        sigma_dash = 952.20,
        b = -0.0890,
        epsilon_dash = 0.7371,
        c = -0.6640
    ):
    '''
        generate_data(Nf,E,sigma_dash,b,epsilon_dash,c)->
            SWT,strain,stress,Nf (default for S355)
    '''

    strain =  (sigma_dash/E)*((2*Nf)**b) + epsilon_dash*((2*Nf)**c)
    stress = sigma_dash*((2*Nf)**b)
    SWT = strain*stress
    return SWT,strain,stress,Nf

Nf = 10**np.linspace(1.7,7,50)

SWT,strain,stress,Nf = generate_data(Nf)

# Experimental Data

Exp_strain = np.array([1.00,0.50,2.00,0.40,0.30,0.35,0.30,0.40,1.00,2.00])*(5e-3
)
Exp_stress = np.array([817.39,669.54,775.47,615.40,536.34,581.51,646.56,661.21,6
63.57,768.21])-300
Exp_Nf     = np.array([4805,16175,336,29501,861304,278243,191940,64244,2009,5420
])

plt.figure(figsize=(18,18))

plt.subplot(2,2,1)

plt.ylabel('SWT')
plt.xlabel('Nf')
plt.xscale('log')
plt.yscale('log')

plt.plot(Nf,SWT,'-')
plt.plot(Exp_Nf,Exp_strain*Exp_stress,'r.')
plt.grid()

plt.subplot(2,2,2)

plt.ylabel('\u0394 \u03B5 /2')
plt.xlabel('Nf')
plt.xscale('log')
plt.yscale('log')
plt.plot(Exp_Nf,Exp_strain,'r.')
plt.plot(Nf,strain,'-')
plt.grid()
plt.subplot(2,2,3)

plt.ylabel('stress')
```

```
plt.xlabel('Nf')
plt.xscale('log')
plt.plot(Nf,stress)
plt.plot(Exp_Nf,Exp_stress,'r.')
plt.grid()
```


SWT
10^1
10^0
10^2 10^3 10^4 10^5 10^6 10^7
Nf


Δ ε /2
10^{-2}
10^{-3}
10^2 10^3 10^4 10^5 10^6 10^7
Nf


stress
600
500
400
300
200
10^2 10^3 10^4 10^5 10^6 10^7
Nf

The Model

General information on Weibul Distribution

If X is a random variable denoting the *time to failure*, the **Weibull distribution** gives a distribution for which the *failure rate* is proportional to a power of time.

$$f_X(x) = \frac{\beta}{\delta}(\frac{x-\lambda}{\delta})^{\beta-1}e^{-(\frac{x-\lambda}{\delta})^\beta}$$
$$F_X(x; \lambda, \delta, \sigma) = 1 - e^{-(\frac{x-\lambda}{\delta})^\beta}$$

where $\beta > 0$ is the **shape parameter**,

$\delta > 0$ is the **scale parameter**,

$\lambda > x$ is the **location parameter** (the minimum value of X).

Percentile points,

$$x_p = \lambda + \delta(-log(1-p))^{\frac{1}{\beta}}$$

where $0 \leq p \leq 1$

Important Properties of Weibull Distribution

- Stable with respect to location and scale
$$X \sim W(\lambda, \delta, \beta) \iff \frac{X-a}{b} \sim W(\frac{\lambda-a}{b}, \frac{\delta}{b}, \beta)$$
- It is stable with respect to Minimum Operations.i.e., if $X_1, X_2, X_3, \ldots\ldots X_m$ are independent and identical distribution,then
$$X_i \sim W(\lambda, \delta, \beta) \iff min(X_1, X_2, \ldots, X_m) \sim W(\lambda, \delta m^{\frac{1}{\beta}}, \beta)$$
if a set of independent and identical distribution is weibull then its minimum is also a Weibull Random Variable

Relevant Variable involved for modeling:

P:Probability of fatigue faliure
N:Number of stress cycles to failure
N_0:Threshold value of N (min lifetime)
SWT:Smith,Watson and Topper fatigue damage parameter
SWT_0:Endurance limit

Putting Related variables together we have three varaibles(based on II Theorem)

$$\frac{N}{N_0}, \frac{SWT}{SWT_0}, P$$
$$P = q(\frac{N}{N_0}, \frac{SWT}{SWT_0})$$

where $q()$ is a function we are to determine
so P can be any monotone function of $\frac{N}{N_0}, \frac{SWT}{SWT_0}$, as $h(\frac{N}{N_0})$ & $g(\frac{SWT}{SWT_0})$

We denote them as

$$N^* = h(\frac{N}{N_0})$$
$$SWT^* = g(\frac{SWT}{SWT_0})$$

Justification of Weibull for S-N fields

Considerations:

- **Weakest Link**: Fatigue lifetime of a longitudinal element is the minimum of its constituting particles.Thus we need minimum model for a longitudinal element $L = ml$
- **Stability**: The distribution function must hold for different lengths.
- **Limit Behaviour**: Need Asymptotic family of Distribution
- **Limited Range**: N^* & SWT^* has finite lower bound,coincide with theoretical end of CDF
$$N \geq N_0$$
$$SWT \geq SWT_0$$
- **Compatibility**:
$$E(N^*; SWT^*) = F(SWT^*; N^*)$$
i.e., Distribution of N^* can be determined based on given SWT^* and similarly SWT^* from N^*.

All these are Statisfied by Weibull Distribution

$$E(N^*; SWT^*) = F(SWT^*; N^*)$$

becomes

$$[\frac{SWT^* - \lambda(N^*)}{\delta(N^*)}]^{\beta(N^*)} = [\frac{N^* - \lambda(SWT^*)}{\delta(SWT^*)}]^{\beta(SWT^*)}$$

Best fitted Solution:

$$\lambda(N^*) = \frac{\lambda}{N^* - B}$$
$$\delta(N^*) = \frac{\delta}{N^* - B}$$
$$\beta(N^*) = \beta$$

and

$$\lambda(SWT^*) = \frac{\lambda}{SWT^* - C}$$
$$\delta(SWT^*) = \frac{\delta}{SWT^* - C}$$
$$\beta(SWT^*) = \beta$$

results in,

$$E[N^*; SWT^*] = F[SWT^*; N^*]$$
$$= 1 - exp\{-(\frac{(N^* - B)(SWT^* - C) - \lambda}{\delta})^{\beta}\}$$

since $SWT^* \longrightarrow \infty$ a lower end for $N^* = h(\frac{N}{N_0)}) = h(1)$ must exists such that $B = h(1)$, similarly for $N^* \longrightarrow \infty$, $C = g(1)$

The percentile curve is constant.

$$\frac{N^*SWT^* - \lambda}{\delta} = constant$$

- The Zero-percentile curve represents the minimum possible N_f for different values of SWT and is a hyperbola As log is used for N_f and SWT we choose,
$$h(x) = g(x) = log(x)$$

therefore,

$$N^* = log(\frac{N}{N_0})$$
$$SWT^* = log(\frac{SWT}{SWT_0})$$

$B = C = log(1) = 0$
$$E(N^*; SWT^*) = F(SWT^*; N^*)$$
$$= 1 - exp\{-(\frac{N^*SWT^*}{\delta})^{\beta}\}$$

p-curves

$$log(\frac{SWT}{SWT^*}) = \frac{\lambda + \delta[-log(1-p)]^{\frac{1}{\beta}}}{log(\frac{N}{N_0})}$$

Final Distribution $$ N^*SWT*^ \sim W(\lambda,\delta,\beta) \ log(\frac{N}{N_0)})log(\frac{SWT}{SWT_0}) \sim W(\lambda,\delta,\beta) \ log(\frac{N}{N_0)})\sim W(\frac{\lambda}{log(\frac{SWT}{SWT_0}) },\frac{\delta}{log(\frac{SWT}{SWT_0}) },\beta) \

$$

The values for this model are:

$logN_0$	$logSWT_0$	λ	δ	β
-4.1079	-4.4317	53.8423	7.2698	3.6226

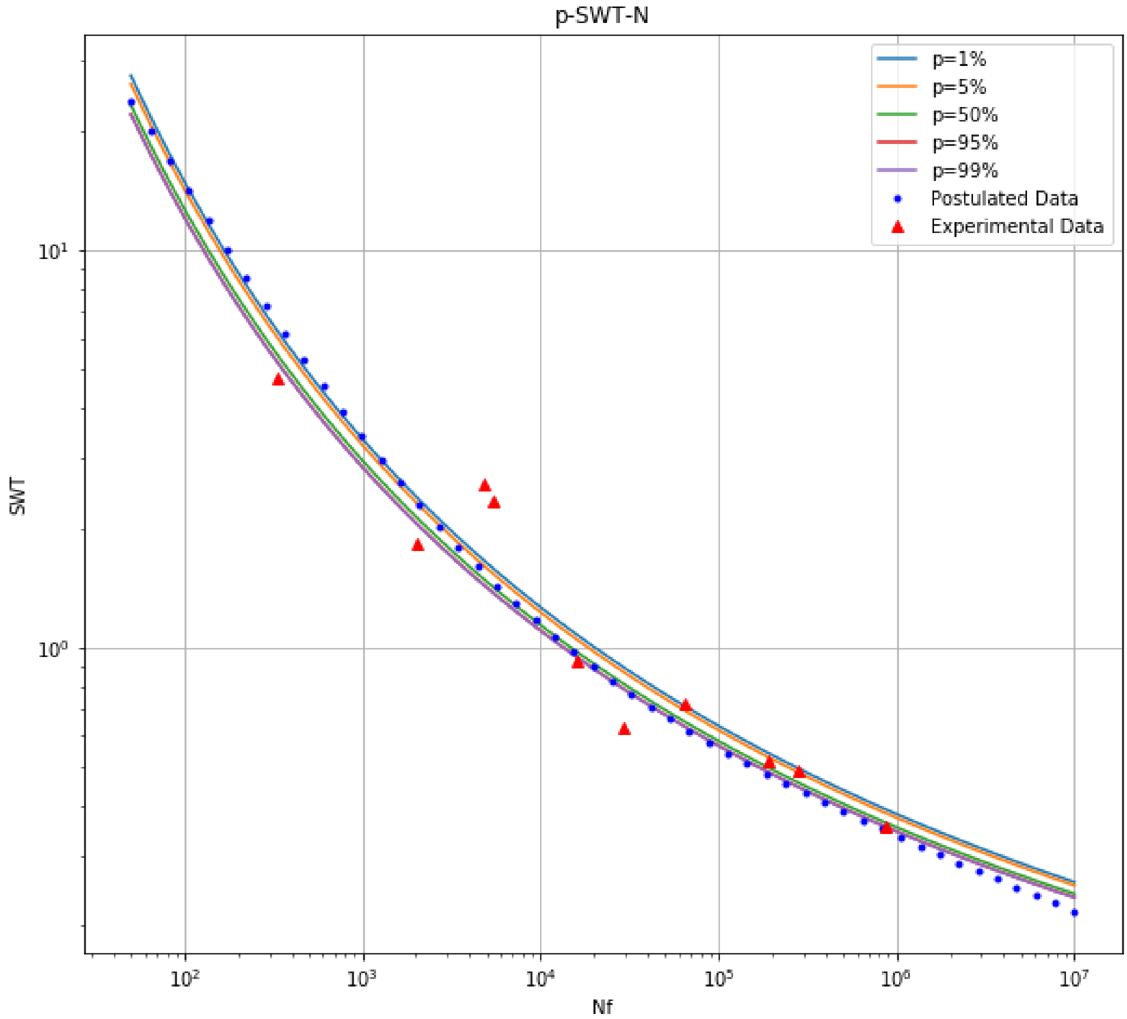
$logN_0$	$log\epsilon_{a0}$	λ	δ	β
-3.2593	-9.1053	36.6676	5.8941	4.6952

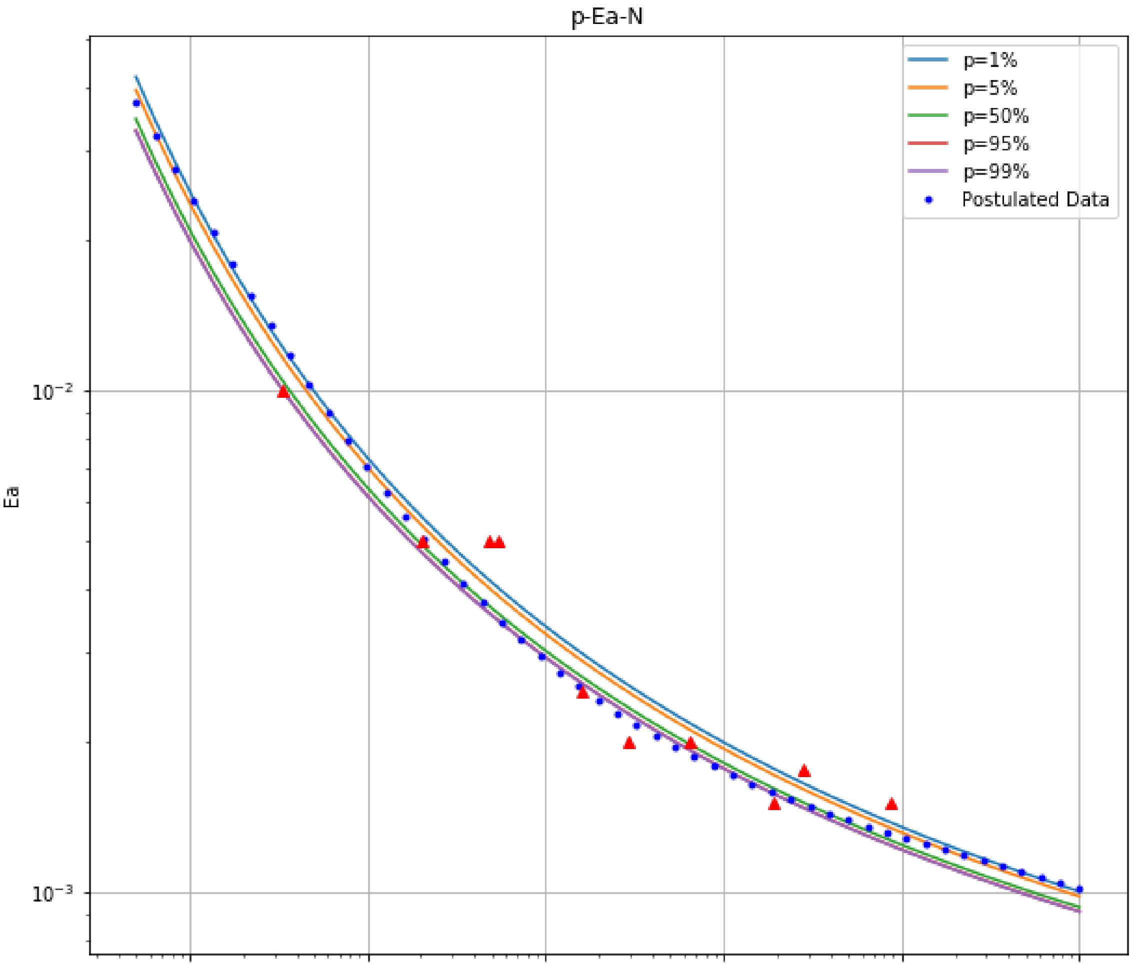
In [33]:

```
import math
class Weibull:
    '''
w = Weibull(shape:beta,scale:theta,position:lambda)

    '''
    def __init__(self,shape,scale,loc,x):
        self.shape = shape
        self.scale = scale
        self.loc = loc
        self.x = x
        _,self.bins  = np.histogram(self.x,100,density=True)


    def pdf(self):
        x = self.bins
        shape = self.shape
        scale = self.scale
        loc =  self.loc
        return ((shape/scale)*((x-loc)/scale)**(shape-1))*(np.exp(-((x-loc)/scal
e)**shape))

    def cdf(self):
        x = self.bins
        shape = self.shape
        scale = self.scale
        loc = self.loc
        return 1- np.exp(-((x-loc)/scale)**shape)

    def failure_rate(self):
        x = self.x
        shape = self.shape
        scale = self.scale
        return (shape/scale)*((x/scale)**(shape-1))

    def E_x(self):
        shape = self.shape
        scale = self.scale
        return np.real(scale*(gamma(1+1/shape)))

    def var_x(self):
        shape = self.shape
        scale = self.scale
        return (scale**2)*(gamma(1+(2/shape))-((gamma(1+(1/shape)))**2))

    def plot_pdf(self):
        plt.plot(self.bins,self.pdf())
        plt.grid()

    def plot_cdf(self):
        plt.plot(self.bins,self.cdf())
        plt.grid()

    def plot_fr(self):
        plt.plot(self.bins,self.failure_rate())
        plt.grid()

    def plot_hist(self):
        plt.hist(self.x)
```

```
        plt.grid()

    def get_xp(self,F_x):
        return  np.real(self.loc+(self.scale)*(-(math.log(1-F_x))**(1/self.shape
)))

logSWT0 = -4.4317
logN01 = -4.1079

loc1 = 53.8423+9
scale1 = 7.2698
shape1 = 3.6226

X1 = (np.log(Nf) - logN01)*(np.log(SWT) - logSWT0)

#loc1,scale1,shape1 = PWM(np.sort(X1))
w1 = Weibull(shape1,scale1,loc1,X1)

xp101 = w1.get_xp(0.001)
xp105 = w1.get_xp(0.005)
xp150 = w1.get_xp(0.050)
xp195 = w1.get_xp(0.095)
xp199 = w1.get_xp(0.099)

pSWT01 = np.exp(logSWT0 + (xp101/(np.log(Nf)-logN01)))
pSWT05 = np.exp(logSWT0 + (xp105/(np.log(Nf)-logN01)))
pSWT50 = np.exp(logSWT0 + (xp150/(np.log(Nf)-logN01)))
pSWT95 = np.exp(logSWT0 + (xp195/(np.log(Nf)-logN01)))
pSWT99 = np.exp(logSWT0 + (xp199/(np.log(Nf)-logN01)))


plt.figure(figsize=(10,20))
plt.subplot(2,1,1)
plt.title("p-SWT-N")
plt.xlabel("Nf")
plt.ylabel("SWT")
plt.xscale('log')
plt.yscale('log')

# percentile curves
plt.plot(Nf,pSWT01,label="p=1%")
plt.plot(Nf,pSWT05,label="p=5%")
plt.plot(Nf,pSWT50,label="p=50%")
plt.plot(Nf,pSWT95,label="p=95%")
plt.plot(Nf,pSWT99,label="p=99%")
# theoretical data
plt.plot(Nf,SWT,'b.',label="Postulated Data")

# Experimental Data
plt.plot(Exp_Nf,Exp_strain*Exp_stress,'r^',label="Experimental Data")
plt.grid()

plt.legend()


# plt.savefig("images/p_s_n_model.png")
# Strain p-E_a-N

logEa0 = -9.1053
logN02 = -3.2593
loc2 = 36.6676+7
```

```
scale2 = 5.8941
shape2 = 4.6952
X2 = (np.log(Nf) - logN02)*(np.log(strain) - logEa0)

w2 = Weibull(shape2,scale2,loc2,X2)

xp201 = w2.get_xp(0.001)
xp205 = w2.get_xp(0.005)
xp250 = w2.get_xp(0.050)
xp295 = w2.get_xp(0.095)
xp299 = w2.get_xp(0.099)

pEa01 = np.exp(logEa0 + (xp201/(np.log(Nf)-logN02)))
pEa05 = np.exp(logEa0 + (xp205/(np.log(Nf)-logN02)))
pEa50 = np.exp(logEa0 + (xp250/(np.log(Nf)-logN02)))
pEa95 = np.exp(logEa0 + (xp295/(np.log(Nf)-logN02)))
pEa99 = np.exp(logEa0 + (xp299/(np.log(Nf)-logN02)))

plt.subplot(2,1,2)
plt.title("p-Ea-N")
plt.xlabel("Nf")
plt.ylabel("Ea")
plt.xscale('log')
plt.yscale('log')

# percentile curves
plt.plot(Nf,pEa01,label="p=1%")
plt.plot(Nf,pEa05,label="p=5%")
plt.plot(Nf,pEa50,label="p=50%")
plt.plot(Nf,pEa95,label="p=95%")
plt.plot(Nf,pEa99,label="p=99%")
# theoretical data
plt.plot(Nf,strain,'b.',label="Postulated Data")
# Experimental Data
plt.plot(Exp_Nf,Exp_strain,'r^')
plt.grid()

plt.legend()
plt.savefig("images/model.png")
```


p-SWT-N
p=1%
p=5%
p=50%
p=95%
p=99%
Postulated Data
Experimental Data
SWT
Nf
10^1
10^0
10^2
10^3
10^4
10^5
10^6
10^7


p-Ea-N
p=1%
p=5%
p=50%
p=95%
p=99%
Postulated Data
Ea
10^{-2}
10^{-3}

10^2 10^3 10^4 10^5 10^6 10^7
Nf

In [ ]:

p-Nf given SWT

$$N^* SWT^* \sim W(\lambda, \delta, \beta)$$
$$log(\frac{N}{N_0}) log(\frac{SWT}{SWT_0}) \sim W(\lambda, \delta, \beta)$$
$$log(\frac{N}{N_0}) \sim W(\frac{\lambda}{log(\frac{SWT}{SWT_0})}, \frac{\delta}{log(\frac{SWT}{SWT_0})}, \beta)$$
$$log(\frac{N}{N_0}) = x_p$$
$$log(N_f) = x_p + log(N_0)$$
$$N_f = e^{x_p + log(N_0)}$$

Determining the fatigue crack propogation curves

$$\frac{\rho^*}{N_f} = \frac{da}{dN} = C \Delta K^m$$

$\rho^* = 5.5 \mu m$

- Determining p-$\frac{da}{dN}$-ΔK curves:

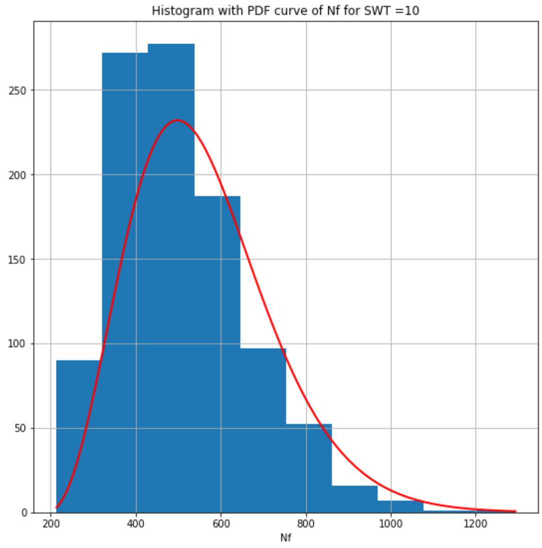
$$log(\frac{N}{N_0}) \sim W(\frac{\lambda}{log(\frac{SWT}{SWT_0})}, \frac{\delta}{log(\frac{SWT}{SWT_0})}, \beta)$$

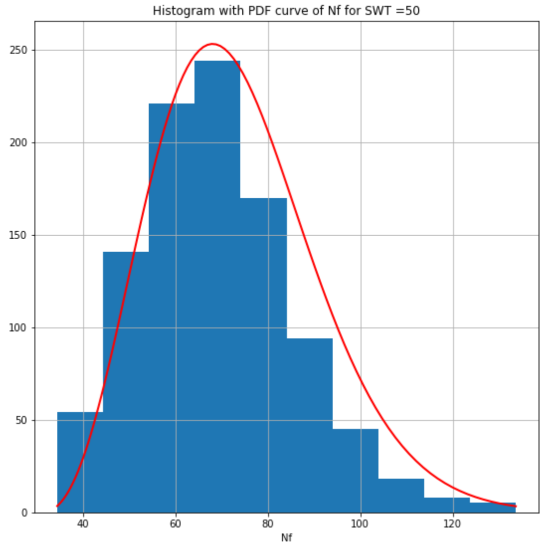
Generating Crack Propogation value:

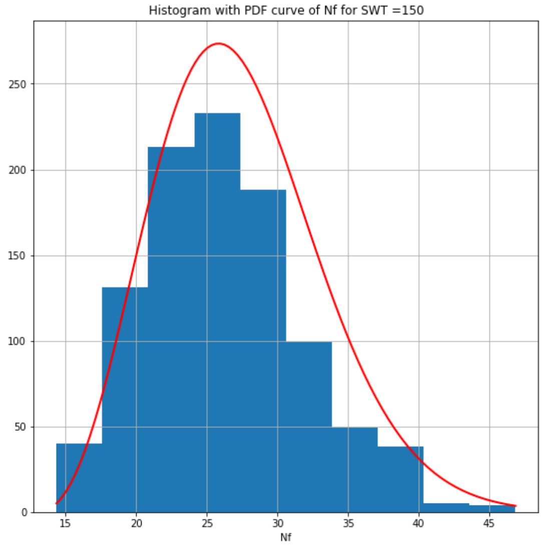
	C	m
R=0.0	$7.195x10^{-15}$	3.499
R=0.5	$6.281x10^{-15}$	3.555
R=0.7	$2.037x10^{-13}$	3.003

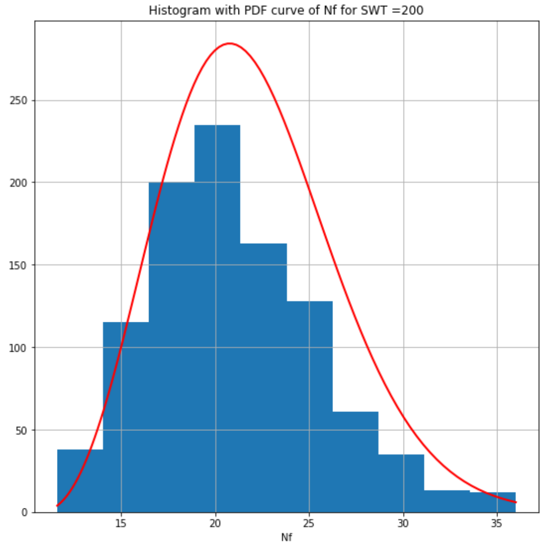
In [4]:

```
import math

# coping values of p-SWT-N curve

# determing p-Nf for a particular SWT
# say SWT = 9
def plt_pNf(SWT):
    logSWT0 = -4.4317
    lgSWT_SWT0 = math.log(SWT) - logSWT0

    logN01 = -4.1079
    loc3 = (53.8423+9)/lgSWT_SWT0
    scale3 = (7.2698)/lgSWT_SWT0
    shape1 = 3.6226
    #print(loc3,scale3,shape1)
    # X3 Weibull random variable (Nf) given SWT = 9
    X3 = (np.random.weibull(shape1,1000)*scale3) + loc3
    _,bins  = np.histogram(X3,100,density=True)
    pdf = ((shape1/scale3)*((bins-loc3)/scale3)**(shape1-1))*(np.exp(-((bins-loc
3)/scale3)**shape1))

    plt.xlabel("Nf")
    plt.title("Histogram with PDF curve of Nf for SWT ="+str(SWT))
    h,_,_ = plt.hist(np.exp(X3+logN01),label='histograph')
    plt.plot(np.exp(bins+logN01),pdf*(h.max()*(0.65)),'r-',label='PDF',linewidth
="2")
    plt.grid()

plt.figure(figsize=(20,20))
plt.subplot(2,2,1)
plt_pNf(10)

plt.subplot(2,2,2)
plt_pNf(50)

plt.subplot(2,2,3)
plt_pNf(150)

plt.subplot(2,2,4)
plt_pNf(200)




plt.savefig("images/nfswtpdf.png")
```


Histogram with PDF curve of Nf for SWT =10
Nf
0
50
100
150
200
250
200
400
600
800
1000
1200


Histogram with PDF curve of Nf for SWT =50
Nf
0
50
100
150
200
250
40
60
80
100
120


Histogram with PDF curve of Nf for SWT =150
Nf
0
50
100
150
200
250
15
20
25
30
35
40
45


Histogram with PDF curve of Nf for SWT =200
Nf
0
50
100
150
200
250
15
20
25
30
35

Probability Weighted Moments(PWM) method to determine Weibull parameters

$$\frac{3M_2 - M_0}{2M1 - M_0} = \frac{2 - 3.2^{\frac{-1}{\beta}} + 3^{\frac{-1}{\beta}}}{1 - 2^{\frac{-1}{\beta}}}$$
$$\delta = \frac{2M_1 - M_0}{(1 - -2^{\frac{-1}{\beta}})\Gamma_\beta}$$
$$\lambda = M_0 - \delta\Gamma_\beta$$

where $M_r = M_{1,r,0}, r = 0, 1, 2$ and $\Gamma_\beta = \Gamma(1 + \frac{1}{\beta})$ and

$$\hat{M}_0 = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\hat{M}_1 = \frac{1}{n(n-1)}\sum_{i=1}^{n}(i-1)x_i$$

$$\hat{M}_2 = \frac{1}{n(n-1)(n-2)}\sum_{i=1}^{n}(i-1)(i-2)x_i$$

Since β cannot be determined metamatically from the equation Numerical Method is used,

- Newton's Method for Mathematical Approximation

  $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

  where,

  $$f(x_n) = 2^x(C-3) + 3^x - (C-2)$$
  $$f'(x_n) = (log2)(2^x)(C-3) + log3(3^x)$$

  where $x = \frac{-1}{\beta}$ and $C = \frac{3M_2 - M_0}{2M1 - M_0}$

In [22]:

```
def getMoments(X):
    n = X.size
    M0 = (1/n)*(X.sum())

    sum1 = 0
    for i in range(n):
        sum1 += i*X[i]
    M1 = (1/((n)*(n-1)))*sum1

    sum2 = 0
    for i in range(n):
        sum2 += i*(i-1)*X[i]
    M2 = (1/(n*(n-1)*(n-2)))*sum2

    return M0,M1,M2

def f(x,C):
    return (2**x)*(C - 3) + (3**x) - (C-2)

def f_dash(x,C):
    return math.log(2)*(2**x)*(C - 3) - math.log(3)*(3**x)

def newton_approx(C,f,f_dash,x=-1,iteration=1000):
    for _  in range(iteration):
        x = x - (f(x,C)/f_dash(x,C))
        #print(x)
    return x

def getpara(M0,M1,M2,newt_init,newt_iterations):
    shape = -1/ newton_approx((3*M2 - M0)/(2*M1 - M0),f,f_dash,x = newt_init,ite
ration = newt_iterations)
    scale = (2*M1 - M0)/((1 - 2**(-(1/shape)))*math.gamma(1+(1/shape)))
    loc = M0 - scale*(math.gamma(1+(1/shape)))

    return loc,scale,shape

def PWM(X,newt_init=-0.1,newt_iterations = 100):
    return getpara(*getMoments(X),newt_init,newt_iterations)
```

In [23]:

```
print('predicted values:',PWM(np.sort(X1)))
print('values in  paper:',loc1,scale1,shape1)
```

```
predicted values: (58.31011426806585, 2.8096569546850514, 2.30812349
98763445)
values in  paper: 62.8423 7.2698 3.6226
```

In [24]:

```
A=PWM(np.sort(X1))
B=loc1,scale1,shape1
```

In [25]:

```
A
```

Out[25]:

```
(58.31011426806585, 2.8096569546850514, 2.3081234998763445)
```

In [26]:

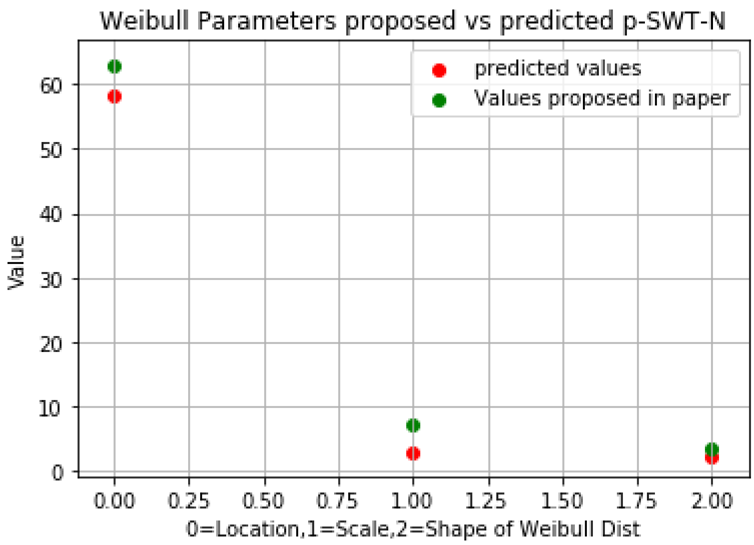
```
B
```

Out[26]:

```
(62.8423, 7.2698, 3.6226)
```

In [27]:

```
plt.title("Weibull Parameters proposed vs predicted p-SWT-N")
plt.scatter(np.arange(0,3),A,c="r",label="predicted values")
plt.scatter(np.arange(0,3),B,c="g",label="Values proposed in paper")

plt.ylabel('Value')
plt.xlabel('0=Location,1=Scale,2=Shape of Weibull Dist')
plt.legend()
plt.grid()
plt.savefig("images/PWMPlotswt.png")
```



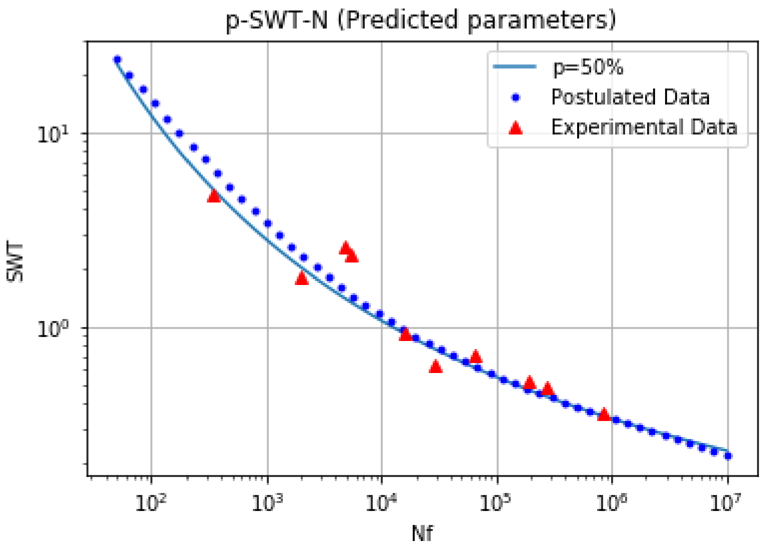
In [28]:

```
print("Predicted (Location, Scale,Shape) = " , A)
print("Proposed in Paper(Location, Scale, Shape) = ", B)
```

```
Predicted (Location, Scale,Shape) =  (58.31011426806585, 2.809656954
6850514, 2.3081234998763445)
Proposed in Paper(Location, Scale, Shape) =  (62.8423, 7.2698, 3.622
6)
```

In [39]:

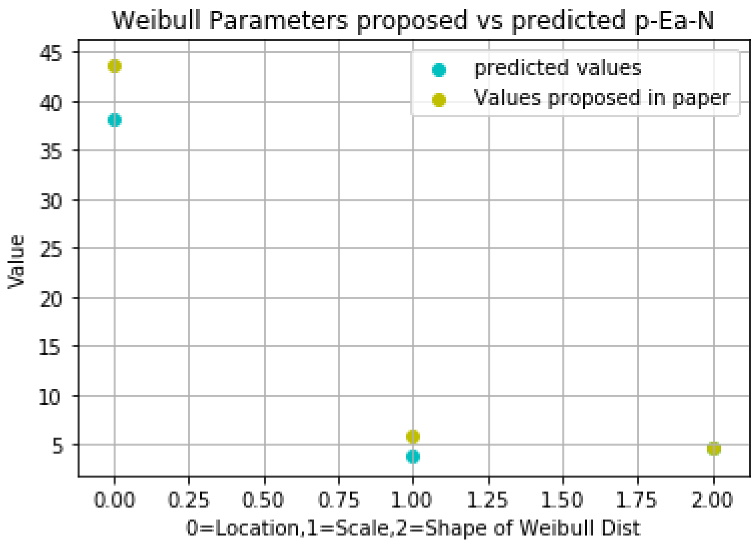
```
# added
X1_pred = (np.log(Nf) + 3.84135)*(np.log(SWT) + 4.3838)
# --
# added
w1_pred = Weibull(2.3081,2.8096,58.31011,X1_pred)
xp150_pred = w1_pred.get_xp(0.050)

pSWT50_pred = np.exp((-4.3838) + (xp150_pred/(np.log(Nf)-(-3.84135))))

# --

# percentile curve of predicted values

plt.title("p-SWT-N (Predicted parameters)")
plt.xlabel("Nf")
plt.ylabel("SWT")
plt.xscale('log')
plt.yscale('log')
plt.plot(Nf,pSWT50_pred,'-',label="p=50%")
# theoretical data
plt.plot(Nf,SWT,'b.',label="Postulated Data")

# Experimental Data
plt.plot(Exp_Nf,Exp_strain*Exp_stress,'r^',label="Experimental Data")
plt.grid()

plt.legend()

plt.savefig("images/p-SWT-Nf_predpara.png")
```



In [29]:

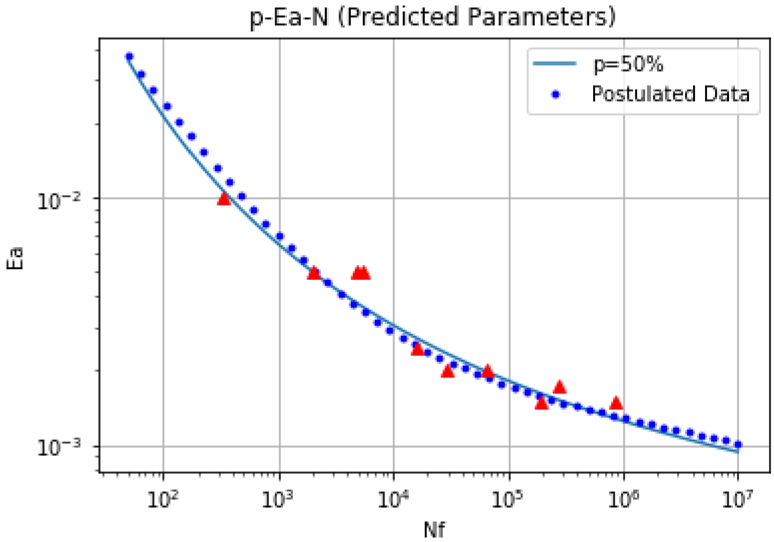
```
A=PWM(np.sort(X2))
B=loc2,scale2,shape2
plt.title("Weibull Parameters proposed vs predicted p-Ea-N")
plt.scatter(np.arange(0,3),A,c="c",label="predicted values")
plt.scatter(np.arange(0,3),B,c="y",label="Values proposed in paper")

plt.ylabel('Value')
plt.xlabel('0=Location,1=Scale,2=Shape of Weibull Dist')
plt.legend()
plt.grid()
plt.savefig("images/PWMPlotstrain.png")
print("Predicted (Location, Scale,Shape) = " , A)
print("Proposed in Paper(Location, Scale, Shape) = ", B)
```

```
Predicted (Location, Scale,Shape) =  (38.11514451850456, 3.869499492
983117, 4.674779199693778)
Proposed in Paper(Location, Scale, Shape) =  (43.6676, 5.8941, 4.695
2)
```



In [38]:

```
# added
X2_pred = (np.log(Nf) + 3.21691)*(np.log(strain) + 9.1053)
w2_pred = Weibull(4.67477,3.869499,38.115144,X2_pred)

xp250_pred = w2_pred.get_xp(0.050)

pEa50_pred = np.exp((-9.1053) + (xp250/(np.log(Nf) + 3.21691)))


plt.title("p-Ea-N (Predicted Parameters)")
plt.xlabel("Nf")
plt.ylabel("Ea")
plt.xscale('log')
plt.yscale('log')

plt.plot(Nf,pEa50_pred,label="p=50%")


plt.plot(Nf,strain,'b.',label="Postulated Data")
# Experimental Data
plt.plot(Exp_Nf,Exp_strain,'r^')
plt.grid()

plt.legend()
#--

plt.savefig("images/p-Ea-Nf_predpara.png")
```



Estimation of Threshold Value $(N_0, \Delta\sigma_0)$

Using the analysis done by Castillo, E. and Galambos, J. [20] for lifetime regression models we formulate the following to predict the $SWT_0 and N_0$

$$E[log(\frac{N}{N_0})|log(\frac{\Delta SWT}{\Delta SWT_0})] = \frac{E[N^* \Delta SWT^*]}{log(\frac{\Delta SWT}{\Delta SWT_0})}$$

$$E[log(N)|log(\frac{\Delta SWT}{\Delta SWT_0})] = log(N_0) + \frac{K}{log(\frac{\Delta SWT}{\Delta SWT_0})}$$

$$where, K = \lambda + \delta\Gamma(1 + \frac{1}{\beta})$$

Minimize Error Function Q

$$Q = \sum_{i=0}^{m}\sum_{j=1}^{n_i}(logN_{ij} - logN_0 - \frac{K}{log\Delta SWT_i - log\Delta SWT_0})^2$$

to get $logN_0$ and $log\Delta SWT_0$

We further minimise this cost function using a Gradient Descent algorithm to find optimal values of N_0 and SWT_0

The intial Estimation of Threshold

- The starting parameters provided to gradient descent are calculated mathematically using,

$$\mu_i = \frac{1}{n_i}\sum_{j=1}^{n_i}(logN_{ij} = logN_0 - \frac{K}{log\Delta SWT_i - log\Delta SWT_0})^2$$

Gradient Descent

Gradient Descent is used to find the minimun of a function known as cost function, here in our case is the error function.Gradient Descent is a iterative learning algorithm and with each step it moves closer to the minimum based on the gradient of the cost function and learning rate.

$$Cost function = J(\theta)$$

$$Gradient = \frac{\partial J(\theta)}{\partial \theta_i}$$

$$New parameter = \theta_i = \theta_i - \alpha\frac{\partial J(\theta)}{\partial \theta_i}$$

where θ are the parameters and α is the learning rate.Here,

$$Cost\,function = Q = \sum_{i=0}^{m} \sum_{j=1}^{n_i} (logN_{ij} - logN_0 - \frac{K}{log\Delta SWT_i - log\Delta SWT_0})^2$$

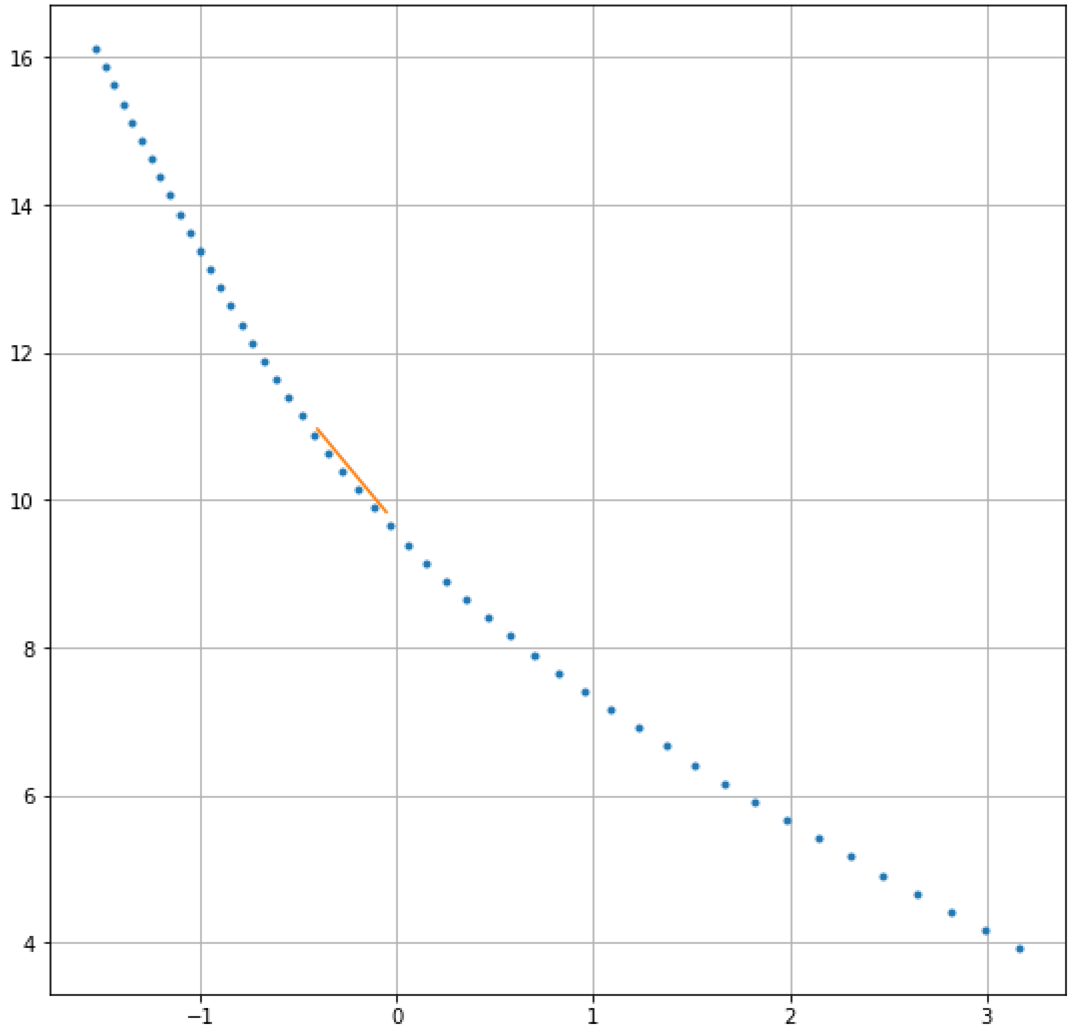
$$\frac{\partial Q}{\partial logN_0} = \frac{1}{M} \sum -2(logN - logN_0 - \frac{K}{log\Delta SWT_i - log\Delta SWT_0})$$

In [31]:

```
Nf_logged=np.log(Nf)
swt_logged=np.log(SWT)

plt.figure(figsize=(20,20))
plt.subplot(2,2,1)
# ploting the distribution
plt.plot(swt_logged,Nf_logged,'.')

# determine the initial estimation
def initial_est(Nf_logged,stress_logged):

    x1 = np.mean(stress_logged[0:16])
    x2 = np.mean(stress_logged[16:32])
    x3 = np.mean(stress_logged[32:48])
    y1 = np.mean(Nf_logged[0:16])
    y2 = np.mean(Nf_logged[16:32])
    y3 = np.mean(Nf_logged[32:48])

    c = (x2*(x3-x1)*(y1-y2) - x3*(x2-x1)*(y1-y3)) / ((x3-x1)*(y1-y2)-(x2-x1)*(y1
-y3))
    b = ((y1-y2)*(x2-c)*(x1-c))/(x2-x1)
    a = y1 - (b/(x1-c))
    return a,b,c

int_a,int_b,int_c = initial_est(Nf_logged,swt_logged)

# plot intial estimation
int_points = np.array([-0.4,-0.05]) # start and end points based on graph of exp
erimental values
int_val = int_a + (int_b/(int_points - int_c)) # values corresponind to that poi
nts based on initial parameters

plt.plot(int_points,int_val,label="initial estimation")
plt.grid()

print('initial estimation of parameters :',int_a,int_b,int_c)

# Better the estimation with gradient Descent

# Cost loss function
def cal_cost(a,b,c,X,Y):
    cost = (Y - a - (b/(X - c)))**2
    return np.mean(cost)/2

def predict(a,b,c,X,Y,alpha):

    G_a = (-2)*(Y - a - (b/(X-c)))
    G_b = (-2)*(Y - a - (b/(X-c)))*(1/X-c)
    G_c = (-2)*(Y - a - (b/(X-c)))*(b/((X-c)**2))

    a_new = a - (alpha*(np.mean(G_a)))
    b_new = b - (alpha*(np.mean(G_b)))
    c_new = c - (alpha*(np.mean(G_c)))
    #print(np.mean(G_a),np.mean(G_b),np.mean(G_c))
    return a_new,b_new,c_new

def gradient_desc(a,b,c,X,Y,alpha= 0.001,iterations=3000):
    cost_history = np.zeros(iterations)
    para_history = np.zeros((iterations,3))
```

```
    for i in range(iterations):
        para_history[i,:] = np.array([a,b,c])
        a,b,c = predict(a,b,c,X,Y,alpha)
        cost_history[i] = cal_cost(a,b,c,X,Y)
        #print(m,c)
    return [a,b,c],para_history,cost_history

pred=gradient_desc(int_a,int_b,int_c,Nf_logged,swt_logged)
```

```
initial estimation of parameters : -3.207753151127962 57.51059490872
167 -4.458919712990903
```



In [81]:

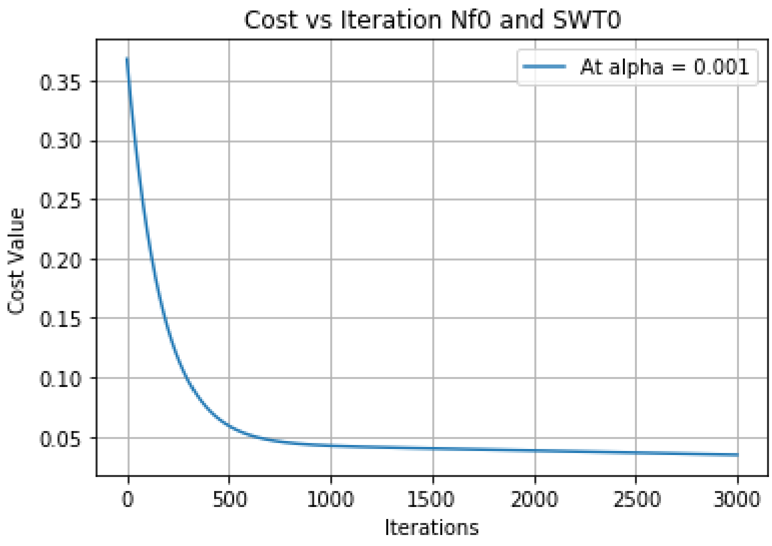
```
pred[0]
```

Out[81]:

```
[-3.8413545070878463, 54.65306079459594, -4.383809373097227]
```

In [82]:

```
plt.title("Cost vs Iteration Nf0 and SWT0")
plt.xlabel("Iterations")
plt.ylabel("Cost Value")
plt.grid()
plt.plot(pred[2],label="At alpha = 0.001")
plt.legend()
plt.savefig("images/costviterswt.png")
```



The values for this model are:

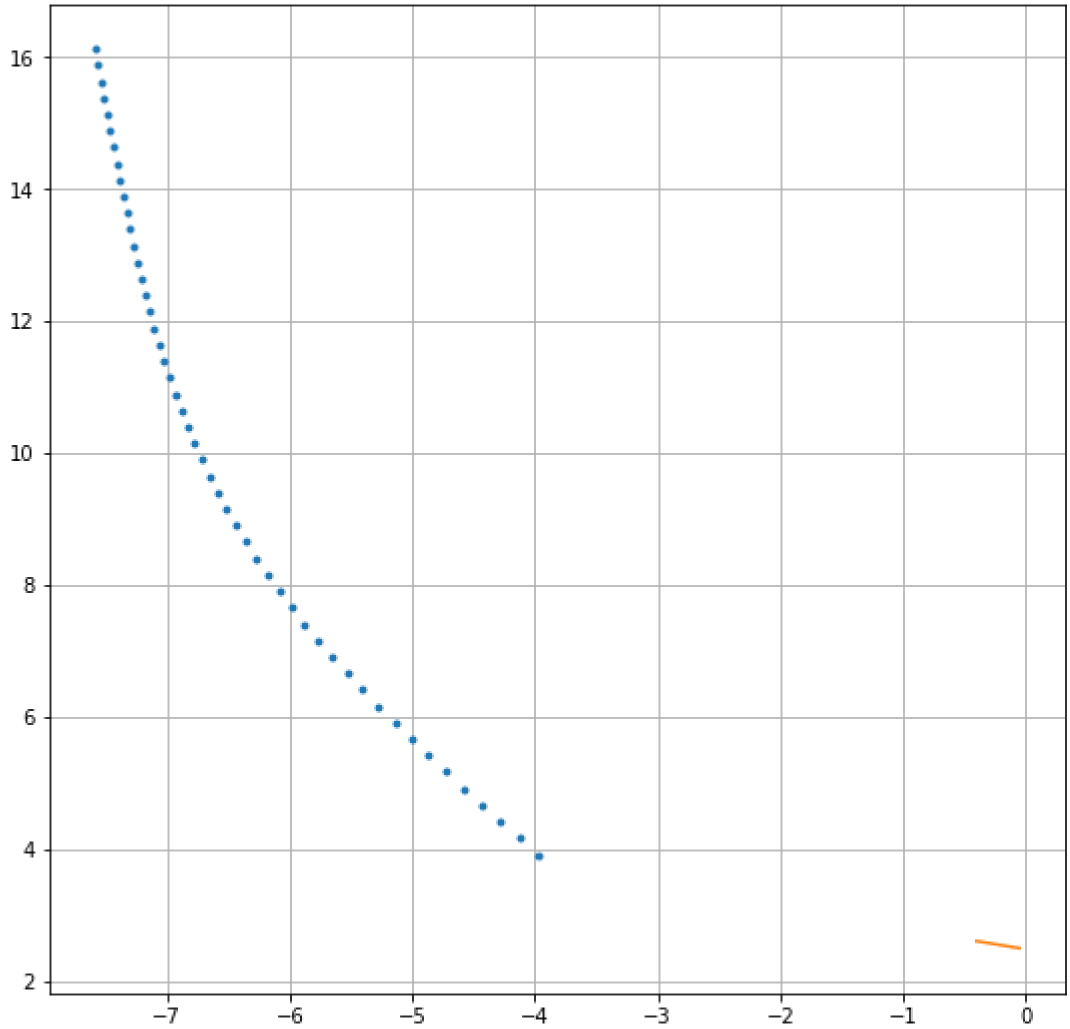
$logN_0$	$logSWT_0$	λ	δ	β
-4.1079	-4.4317	53.8423	7.2698	3.6226

In [84]:

```
Nf_logged=np.log(Nf)
strain_logged=np.log(strain/2)

plt.figure(figsize=(20,20))
plt.subplot(2,2,1)
# ploting the distribution
plt.plot(strain_logged,Nf_logged,'.')

# determine the initial estimation
def initial_est(Nf_logged,stress_logged):

    x1 = np.mean(stress_logged[0:16])
    x2 = np.mean(stress_logged[16:32])
    x3 = np.mean(stress_logged[32:48])
    y1 = np.mean(Nf_logged[0:16])
    y2 = np.mean(Nf_logged[16:32])
    y3 = np.mean(Nf_logged[32:48])

    c = (x2*(x3-x1)*(y1-y2) - x3*(x2-x1)*(y1-y3)) / ((x3-x1)*(y1-y2)-(x2-x1)*(y1
-y3))
    b = ((y1-y2)*(x2-c)*(x1-c))/(x2-x1)
    a = y1 - (b/(x1-c))
    return a,b,c

int_a,int_b,int_c = initial_est(Nf_logged,strain_logged)

# plot intial estimation
int_points = np.array([-0.4,-0.05]) # start and end points based on graph of exp
erimental values
int_val = int_a + (int_b/(int_points - int_c)) # values corresponind to that poi
nts based on initial parameters

plt.plot(int_points,int_val,label="initial estimation")
plt.grid()

print('initial estimation of parameters :',int_a,int_b,int_c)

# Better the estimation with gradient Descent

# Cost loss function
def cal_cost(a,b,c,X,Y):
    cost = (Y - a - (b/(X - c)))**2
    return np.mean(cost)/2

def predict(a,b,c,X,Y,alpha):

    G_a = (-2)*(Y - a - (b/(X-c)))
    G_b = (-2)*(Y - a - (b/(X-c)))*(1/X-c)
    G_c = (-2)*(Y - a - (b/(X-c)))*(b/((X-c)**2))

    a_new = a - (alpha*(np.mean(G_a)))
    b_new = b - (alpha*(np.mean(G_b)))
    c_new = c - (alpha*(np.mean(G_c)))
    #print(np.mean(G_a),np.mean(G_b),np.mean(G_c))
    return a_new,b_new,c_new

def gradient_desc(a,b,c,X,Y,alpha= 0.0003,iterations=1050):
    cost_history = np.zeros(iterations)
    para_history = np.zeros((iterations,3))
```

```
    for i in range(iterations):
        para_history[i,:] = np.array([a,b,c])
        a,b,c = predict(a,b,c,X,Y,alpha)
        cost_history[i] = cal_cost(a,b,c,X,Y)
        #print(m,c)
    return [a,b,c],para_history,cost_history

pred=gradient_desc(int_a,int_b,int_c,Nf_logged,strain_logged)
```

```
initial estimation of parameters : -0.14627170999350003 23.746380734
30922 -9.042803939998716
```



In [85]:

```
pred[0]
```

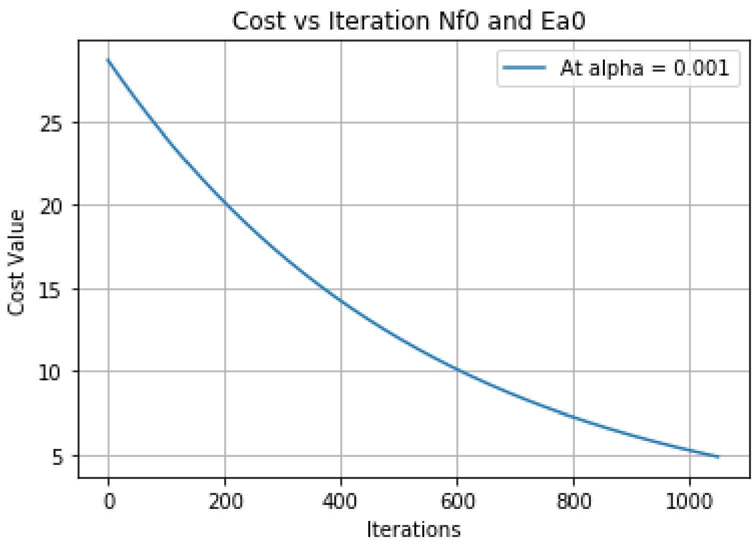
Out[85]:

```
[-3.2169137446018166, -4.544872577340627, -9.12850244974238]
```

$logN_0$	$log\epsilon_{a0}$	λ	δ	β
-3.2593	-9.1053	36.6676	5.8941	4.6952

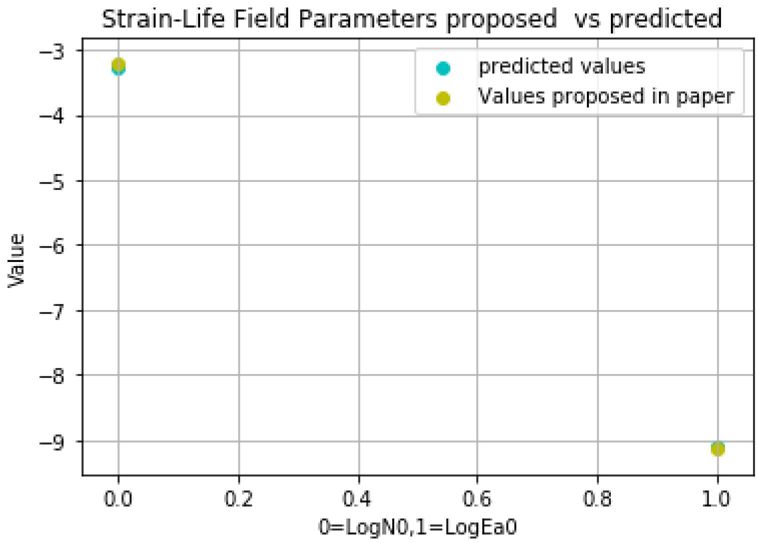
In [86]:

```
plt.title("Cost vs Iteration Nf0 and Ea0 ")
plt.xlabel("Iterations")
plt.ylabel("Cost Value")
plt.grid()
plt.plot(pred[2],label="At alpha = 0.001")
plt.legend()
plt.savefig("images/costviterEa.png")
```



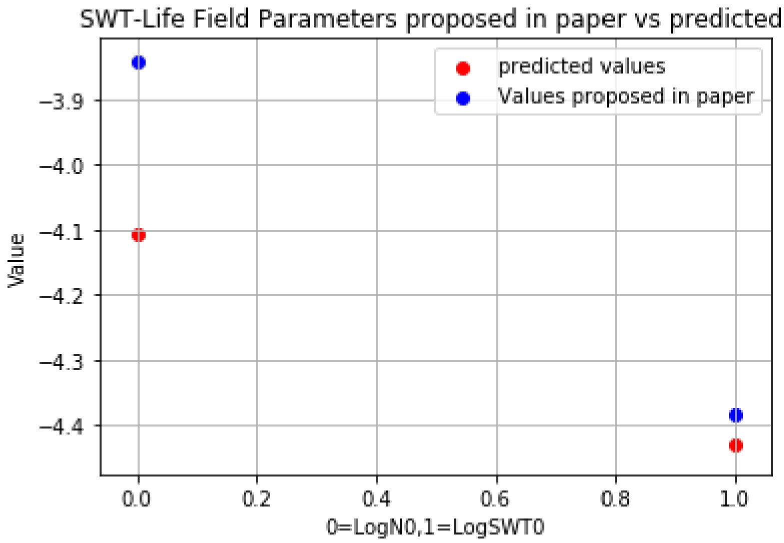
In [90]:

```
constlogN0=-3.2593
constlogEa0=-9.1053
predlogN0=-3.2169137446018166
predlogEa0=-9.12850244974238

A=constlogN0,constlogEa0
B=predlogN0,predlogEa0
plt.title("Strain-Life Field Parameters proposed  vs predicted ")
plt.scatter(np.arange(0,2),A,c="c",label="predicted values")
plt.scatter(np.arange(0,2),B,c="y",label="Values proposed in paper")

plt.ylabel('Value')
plt.xlabel('0=LogN0,1=LogEa0')
plt.legend()
plt.grid()
plt.savefig("images/StrainLifePropvPred.png")
```



In [93]:

```
constlogN0swt=-4.1079
constlogSWT=-4.4317
predlogN0swt=-3.8413545070878463
predlogSWT=-4.383809373097227

A=constlogN0swt,constlogSWT
B=predlogN0swt,predlogSWT
plt.title("SWT-Life Field Parameters proposed in paper vs predicted ")
plt.scatter(np.arange(0,2),A,c="r",label="predicted values")
plt.scatter(np.arange(0,2),B,c="b",label="Values proposed in paper")

plt.ylabel('Value')
plt.xlabel('0=LogN0,1=LogSWT0')
plt.legend()
plt.grid()
plt.savefig("images/SWTLifePropvPred.png")
```



In [ ]: