Appendix

April 10, 2016

In [1]:
```
#packages used

#packages to be downloaded
import numpy as np

import scipy.integrate as integrate
from scipy import linalg as la
import scipy.special as Cheby
from scipy.special import ellipe

import matplotlib.pyplot as plt
%matplotlib inline
font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 16}
plt.rc('font', **font)

#packages pre-installed
from math import pi
```

In [2]:
```
#constants
a = 1            #Crack Half-Length
nu = 0.25        #Poisson's Ratio
G = 26.2         #Shear Stress
E = 2*G*(1+nu)   #Young's Modulus
ks=(1-2*nu)/(2*(1-nu)) #Plain Strain Condition
```

In [3]:
```
degree = 80              #degree of Chebyshev's Approximation
order = degree + 1       #order of Chebyshev's Approximation

#values of x to use for plot
X = np.linspace(-a,a,order+1,False)
X = np.delete(X,0)
Y1 = np.linspace(-0.9999,X[0],10,False)
Y2 = np.linspace(0.9999,X[-1],10,False)
X = np.concatenate((X,Y1,Y2), axis=0)
X.sort()
order += 20
```

In [4]:
```
def First(x):          #For finding first term of LHS for given x
    first = np.zeros([order,])
    for n in xrange(order):
        first[n] = Cheby.eval_chebyu( n , x )
    return first
```

```
In [5]: def Second(x):         #For finding second term of LHS for given x
            second = np.zeros([order,])
            for n in range(order):
                func = lambda z:(Cheby.eval_chebyt(n + 1,z))/(np.sqrt(1-z**2))
                second[n] = integrate.quad(func,-1,x)[0]
            return second
```

```
In [6]: def Chebysol(alphas):   #To find Chebyshev Approximation Coefficients for given alpha*
            A = np.zeros([order,order])

            B = (np.sqrt(np.power(a,2)-np.power(X,2)))/(2*G*(1-ks))

            for x in range(order):
                A[x] = First(X[x])

            for x in range(order):
                A[x] -= alphas*Second(X[x])/(1-ks)

            b = la.solve(A,B)
            return b

        def Chebysol_Mat(x,b):      #To find value for given Chebyshev coefficients and x
            Sol_Mat = 0.0
            for i in range(order):
                    Sol_Mat += b[i]*Cheby.eval_chebyu( i , x )
            return Sol_Mat
```

```
In [7]: def Persol(alphas):     #To find Perturbation Solution for given alpha
            per = np.zeros([order,])
            for i in range(order):
                x = X[i]
                func = lambda t : (t * ellipe(t**2)/((t**2-x**2)**0.5))
                ans0=integrate.quad(func,abs(x),1)[0]
                ans = 4*alphas*(1-nu)*ans0/pi
                ans1 = 2*(1-nu**2)*((1-x**2)**0.5 - ans)/E
                per[i] = 2*ans1
            return per
```

```
In [8]: #Classical Solution i.e alpha* = 0
        b_Classical = Chebysol(0)
        Classical_Mat = np.zeros([order,])
        for i in range(order):
            Classical_Mat[i] = Chebysol_Mat(X[i],b_Classical)
```

```
In [9]: #Chebyshev Solution for alpha* = 0.1
        b_Cheby_1 = Chebysol(0.1)
        Cheby_Mat_1 = np.zeros([order,])
        for i in range(order):
            Cheby_Mat_1[i] = Chebysol_Mat(X[i],b_Cheby_1)
```

```
In [ ]: #Chebyshev Solution for alpha* = 0.2
        b_Cheby_2 = Chebysol(0.2)
```

```
Cheby_Mat_2 = np.zeros([order,])
for i in range(order):
    Cheby_Mat_2[i] = Chebysol_Mat(X[i],b_Cheby_2)
```

In [ ]:
```
#Chebyshev Solution for alpha* = 0.5
b_Cheby_3 = Chebysol(0.5)
Cheby_Mat_3 = np.zeros([order,])
for i in range(order):
    Cheby_Mat_3[i] = Chebysol_Mat(X[i],b_Cheby_3)
```

In [ ]:
```
#Perturbation Solutions for above 3 alpha*
Per_Mat_1 = Persol(0.1)
Per_Mat_2 = Persol(0.2)
Per_Mat_3 = Persol(0.5)
```

In [ ]:
```
#Graph Plot for Classical ,Cohesive and Perturbation
fig1 = plt.figure(num=None, figsize=(20, 10), dpi=300, facecolor='w', edgecolor='k')
ax1 = plt.subplot(111)
ax1.set_xlabel('x/a')
ax1.set_ylabel('normalised crack aperture 2*v(x)/a')
ax1.plot(X,2*Classical_Mat,'s',color='k',label="Classical")
ax1.plot(X,2*Cheby_Mat_1,'-',color='k',label="Cohesive")
ax1.plot(X,Per_Mat_1,'--',color='k',label="Perturbation")

ax1.plot(X,2*Cheby_Mat_2,'-',color='k')
ax1.plot(X,Per_Mat_2,'--',color='k')

ax1.plot(X,2*Cheby_Mat_3,'-',color='k')
ax1.plot(X,Per_Mat_3,'--',color='k')

ax1.annotate(r'$\alpha^* = 0.1$',xy=(X[30],Per_Mat_1[30]), xytext=(X[50],0.035),
             arrowprops=dict(width = 2,headwidth = 10,facecolor='black'))
ax1.annotate('',xy=(X[70],2*Cheby_Mat_1[70]), xytext=(X[54],0.0365),
             arrowprops=dict(width = 2,headwidth = 10,facecolor='black', shrink=0.01))

ax1.annotate(r'$\alpha^* = 0.2$',xy=(X[25],Per_Mat_2[25]), xytext=(X[50],0.025),
             arrowprops=dict(width = 2,headwidth = 10,facecolor='black'))
ax1.annotate('',xy=(X[75],2*Cheby_Mat_2[75]), xytext=(X[54],0.0265),
             arrowprops=dict(width = 2,headwidth = 10,facecolor='black', shrink=0.01))

ax1.annotate(r'$\alpha^* = 0.5$',xy=(X[20],Per_Mat_3[20]), xytext=(X[50],0.010),
             arrowprops=dict(width = 2,headwidth = 10,facecolor='black'))
ax1.annotate('',xy=(X[80],2*Cheby_Mat_3[80]), xytext=(X[54],0.0113),
             arrowprops=dict(width = 2,headwidth = 10,facecolor='black', shrink=0.01))

plt.legend(loc=1)
plt.show()
fig1.savefig("Result.png")
```

In [ ]:
```
#For plotting error graph
alp = np.zeros([100,])  #different alpha* matrix
C = Classical_Mat[50]

for i in range(1,51):
    alp[i-1] = 1.0/(i*2)
```

```
for i in range(51,101):
    alp[i-1] = float(i)

alp.sort()
```

In [ ]:
```
#Finding Error for given alpha*
err = np.zeros([100,])
for i in range(100):
    b_temp = Chebysol(alp[i])
    P = Persol(alp[i])[50]
    err[i] = (2*Chebysol_Mat(X[50],b_temp) - P)/(2*C - P)
    print alp[i],err[i]
```

In [ ]:
```
#Graph Plot for error vs 1/alpha*
fig2 = plt.figure(num=None, figsize=(20, 10), dpi=300, facecolor='w', edgecolor='k')
ax2 = plt.subplot(111)
ax2.set_xlabel(r'$1/\alpha^*$')
ax2.set_ylabel(r'$Error(Correction Terms)_(x=0)$')
ax2 = plt.subplot(111)
ax2.plot(1/alp,err)
plt.show()
fig2.savefig("Error.png")
```