1

We look these values up from pp. 136-143

a. 42 ksi

b. 48 ksi

c. 68 ksi

d. 70 ksi

2

These values we get from the charts on pp. 111-121

a. Room Temperature: $133 \text{ ksi}\sqrt{\text{in.}}$

b. Room Temperature: $140 \text{ ksi}\sqrt{\text{in.}}$

c. Room Temperature: $43 \text{ ksi}\sqrt{\text{in.}}$

d. Room Temperature: $60 \text{ ksi}\sqrt{\text{in.}}$

3

We now use the Fedderson approach to plot the residual strength as a function of crack length

a. For 2024-T351, we find

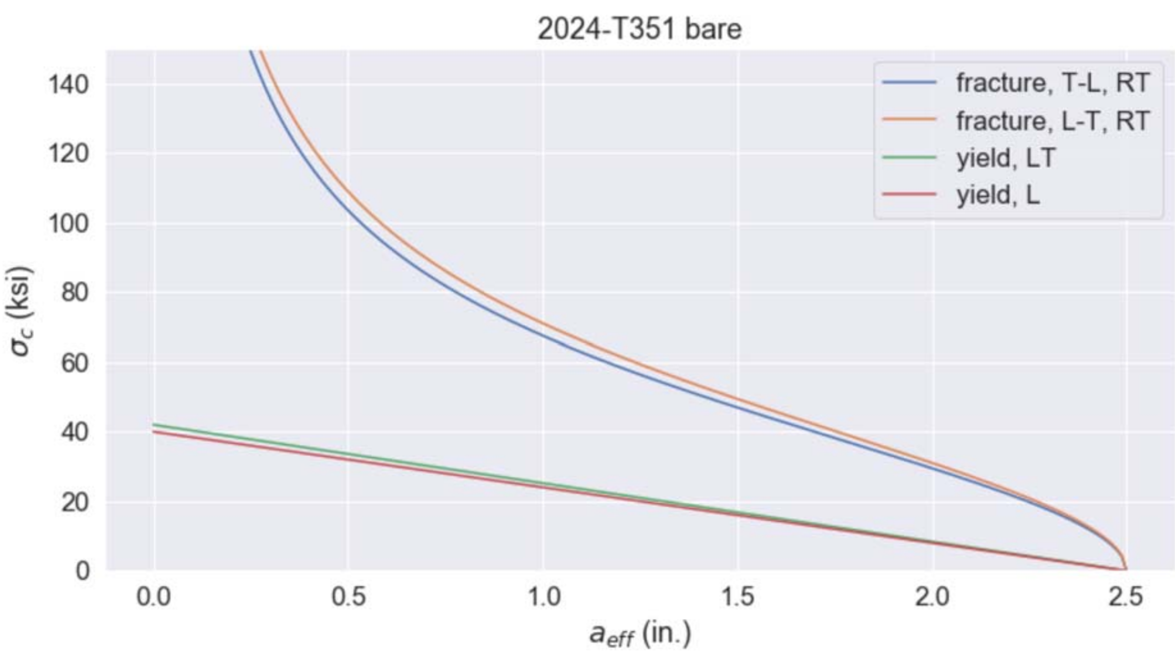
In [1]:
```
# load libraries
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sb
sb.set(font_scale=1.5)
%matplotlib inline

#interpolation library to go between discrete points
from scipy import interpolate
#optimization library
from scipy.optimize import minimize
```

We can now generate a preliminary plot for these conditions

In [2]:
```
#panel properties
W = 5.0 #in

#2024-T351
Kc_TL_RT = 133.0 #ksi sqrt(in)
Kc_LT_RT = 140.0 #ksi sqrt(in)

s_ys_LT = 42.0 #ksi
s_ys_L = 40.0 #ksi

#crack length array
a = np.linspace(0,W/2,200)

#fracture criteria
sc_f_TL_RT = Kc_TL_RT/(np.sqrt(np.pi*a)*np.sqrt(1./np.cos(np.pi*a/W)))
sc_f_LT_RT = Kc_LT_RT/(np.sqrt(np.pi*a)*np.sqrt(1./np.cos(np.pi*a/W)))

#net section yield criteria
sc_y_LT = s_ys_LT*(W-2*a)/W
sc_y_L = s_ys_L*(W-2*a)/W

#preliminary plot
plt.figure(figsize=(12,6))
plt.plot(a,sc_f_TL_RT,label='fracture, T-L, RT')
plt.plot(a,sc_f_LT_RT,label='fracture, L-T, RT')
plt.plot(a,sc_y_LT,label='yield, LT')
plt.plot(a,sc_y_L,label='yield, L')
plt.xlabel('$a_{eff}$ (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.legend(loc='best')
plt.ylim([0,150])
plt.title('2024-T351 bare')
```

```
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:15: RuntimeWar
ning: divide by zero encountered in true_divide
  from ipykernel import kernelapp as app
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:16: RuntimeWar
ning: divide by zero encountered in true_divide
  app.launch_new_instance()
```

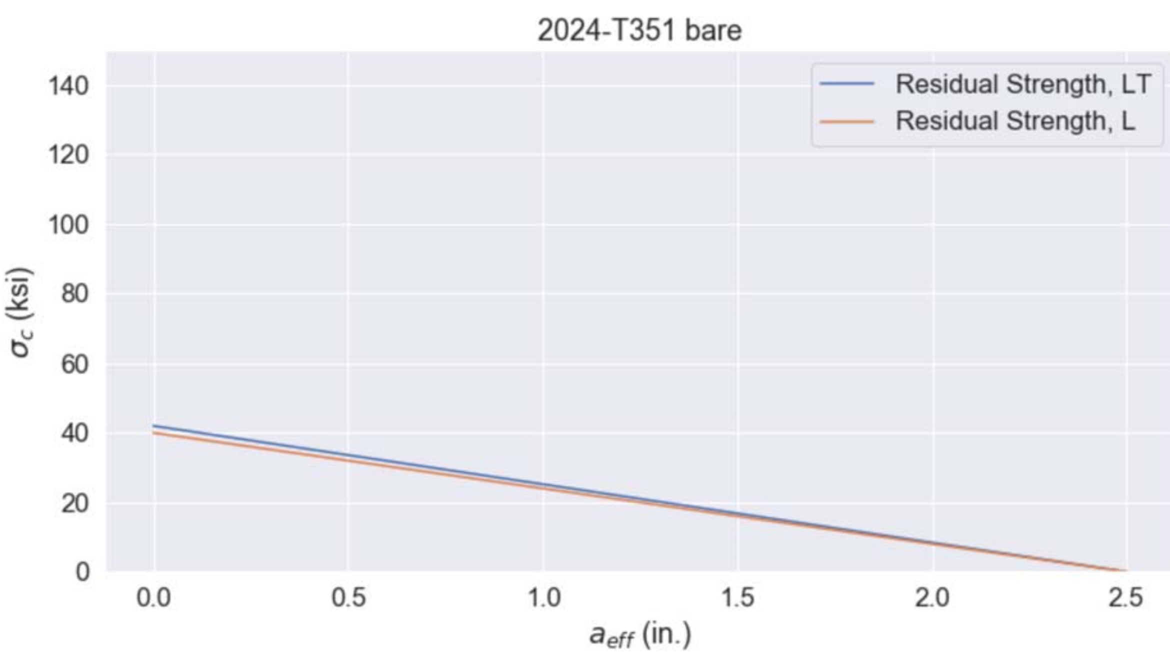
Out[2]:
```
Text(0.5, 1.0, '2024-T351 bare')
```



We can see that in this case, for any size of crack the panel will fail in yield, so the Fedderson approach is no different from any other. The panel will always fail due to net section yield (under the same stress in either grain orientation and temperature condition).

In [3]:
```
#preliminary plot
plt.figure(figsize=(12,6))
plt.plot(a,sc_y_LT,label='Residual Strength, LT')
plt.plot(a,sc_y_L,label='Residual Strength, L')
plt.xlabel('$a_{eff}$ (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.legend(loc='best')
plt.ylim([0,150])
plt.title('2024-T351 bare')
```

Out[3]:
```
Text(0.5, 1.0, '2024-T351 bare')
```

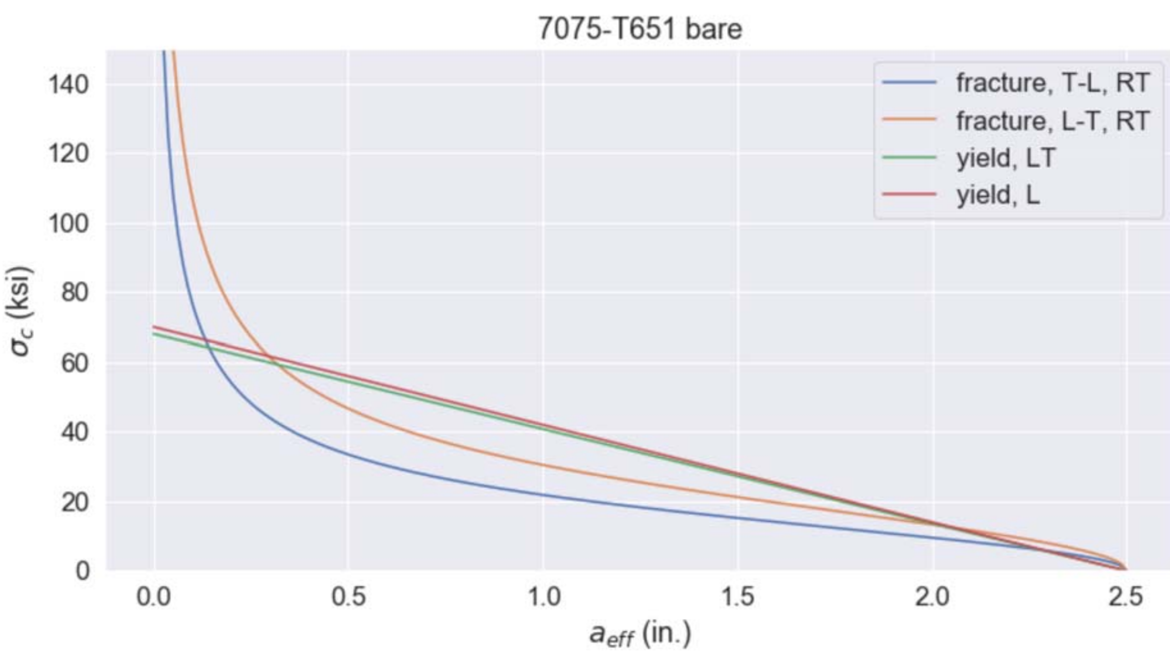


b. For the panel in part b we generate a preliminary plot

In [4]:
```
#panel properties
W = 5.0 #in

#7075-T651
Kc_TL_RT = 43.0 #ksi sqrt(in)
Kc_LT_RT = 60.0 #ksi sqrt(in)

s_ys_LT = 68.0 #ksi
s_ys_L = 70.0 #ksi

#crack length array
a = np.linspace(0,W/2,200)

#fracture criteria
sc_f_TL_RT = Kc_TL_RT/(np.sqrt(np.pi*a)*np.sqrt(1./np.cos(np.pi*a/W)))
sc_f_LT_RT = Kc_LT_RT/(np.sqrt(np.pi*a)*np.sqrt(1./np.cos(np.pi*a/W)))

#net section yield criteria
sc_y_LT = s_ys_LT*(W-2*a)/W
sc_y_L = s_ys_L*(W-2*a)/W

#preliminary plot
plt.figure(figsize=(12,6))
plt.plot(a,sc_f_TL_RT,label='fracture, T-L, RT')
plt.plot(a,sc_f_LT_RT,label='fracture, L-T, RT')
plt.plot(a,sc_y_LT,label='yield, LT')
plt.plot(a,sc_y_L,label='yield, L')
plt.xlabel('$a_{eff}$ (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.legend(loc='best')
plt.ylim([0,150])
plt.title('7075-T651 bare')
```

```
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:15: RuntimeWar
ning: divide by zero encountered in true_divide
  from ipykernel import kernelapp as app
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:16: RuntimeWar
ning: divide by zero encountered in true_divide
  app.launch_new_instance()
```

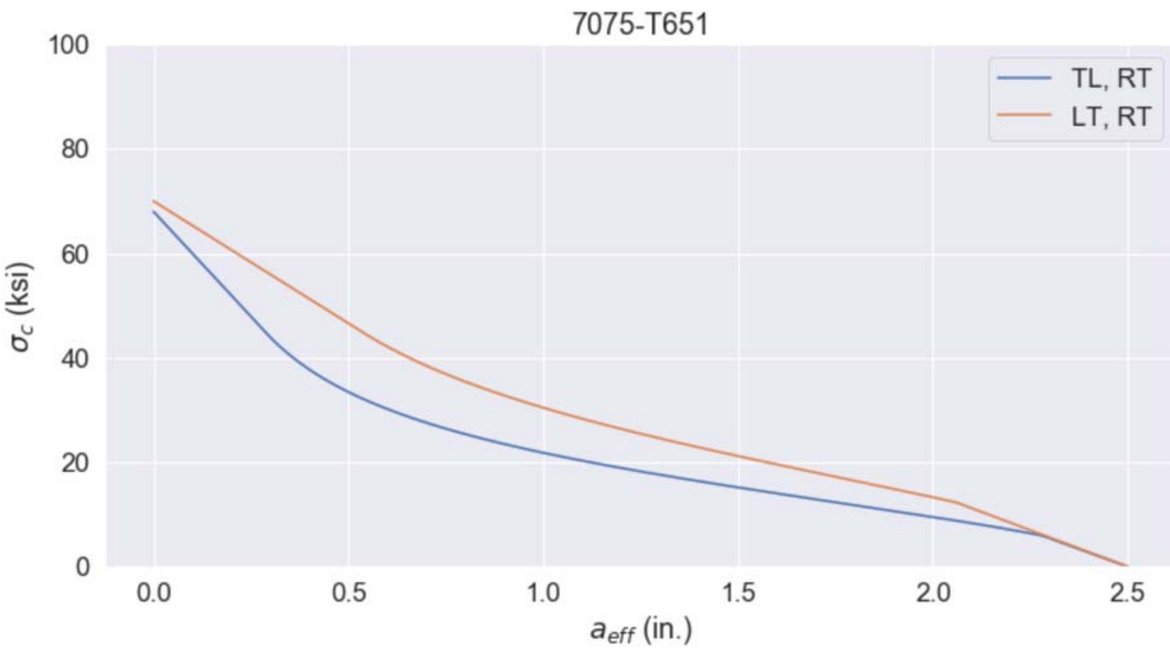
Out[4]:
```
Text(0.5, 1.0, '7075-T651 bare')
```



In this case we will need to use the Fedderson approach to plot tangent curves between the yield and fracture criteria

In [5]:
```
#interpolations of fracture condition
spl_TL_RT = interpolate.splrep(a[1:],sc_f_TL_RT[1:])
spl_LT_RT = interpolate.splrep(a[1:],sc_f_LT_RT[1:])
#guess point of intersection
a0 = 0.7

#objective function for optimization
def myobj(a,spl=spl_TL_RT,sc_y=sc_y_LT):
    fa = interpolate.splev(a,spl,der=0)
    fprime = interpolate.splev(a,spl,der=1)
    return (sc_y - (fa-fprime*a))**2

#optimize
res = minimize(myobj,a0,args=(spl_TL_RT,sc_y_LT[0]))
a_int_TL_RT = res.x[0]
res = minimize(myobj,a0,args=(spl_LT_RT,sc_y_L[0]))
a_int_LT_RT = res.x[0]

#array to plot tangent line
a_tan_TL_RT = np.linspace(0,a_int_TL_RT)
a_tan_LT_RT = np.linspace(0,a_int_LT_RT)

#generate tangent lines
fa_TL_RT = interpolate.splev(a_int_TL_RT,spl_TL_RT,der=0)
fprime_TL_RT = interpolate.splev(a_int_TL_RT,spl_TL_RT,der=1)
fa_LT_RT = interpolate.splev(a_int_LT_RT,spl_LT_RT,der=0)
fprime_LT_RT = interpolate.splev(a_int_LT_RT,spl_LT_RT,der=1)

def mymin(a,a_int,fa,fprime,Kc,s_ys):
    if a < a_int:
        return fa+fprime*(a-a_int)
    else:
        return min([Kc/(np.sqrt(np.pi*a)*np.sqrt(1./np.cos(np.pi*a/W))),s_ys*(W-2*
a)/W])

plt.figure(figsize=(12,6))
plt.plot(a,[mymin(i,a_int_TL_RT,fa_TL_RT,fprime_TL_RT,Kc_TL_RT,s_ys_LT) for i in
a],label='TL, RT')
plt.plot(a,[mymin(i,a_int_LT_RT,fa_LT_RT,fprime_LT_RT,Kc_LT_RT,s_ys_L) for i in a],
label='LT, RT')
plt.xlabel('$a_{eff}$ (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.legend(loc='best')
plt.ylim([0,100])
plt.title('7075-T651')
```

Out[5]: `Text(0.5, 1.0, '7075-T651')`

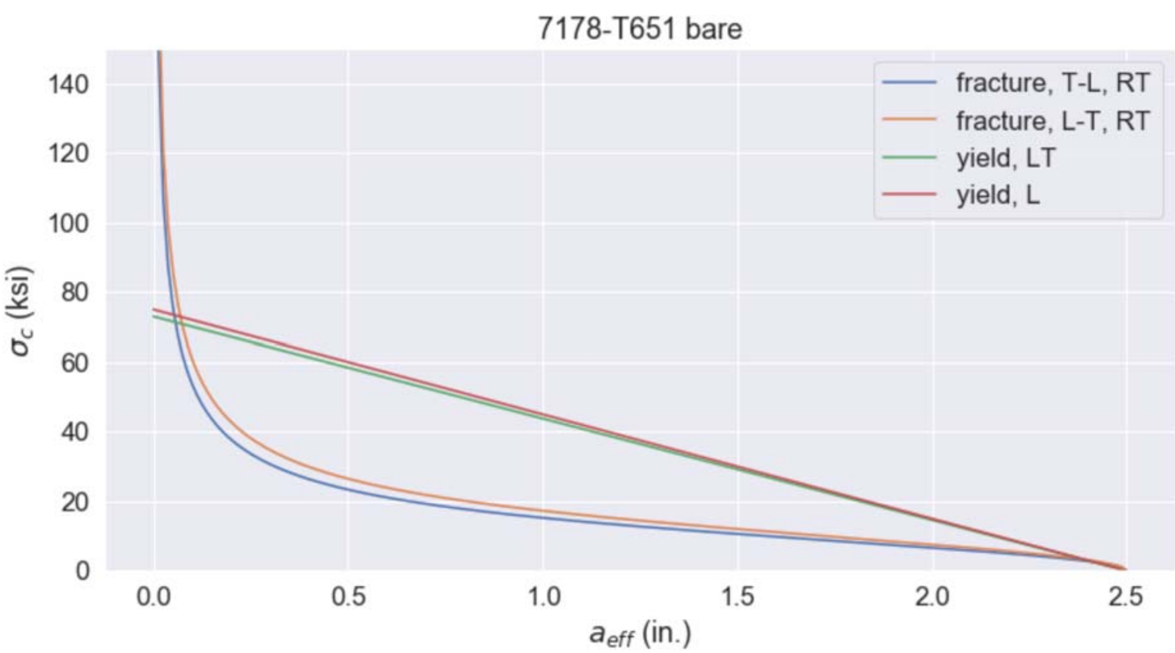


c. For part c we repeat for the values we found for 7178-T651

In [6]:
```
#panel properties
W = 5.0 #in

#7178-T651
Kc_TL_RT = 30.0 #ksi sqrt(in)
Kc_LT_RT = 34.0 #ksi sqrt(in)

s_ys_LT = 73.0 #ksi
s_ys_L = 75.0 #ksi

#crack length array
a = np.linspace(0,W/2,200)

#fracture criteria
sc_f_TL_RT = Kc_TL_RT/(np.sqrt(np.pi*a)*np.sqrt(1./np.cos(np.pi*a/W)))
sc_f_LT_RT = Kc_LT_RT/(np.sqrt(np.pi*a)*np.sqrt(1./np.cos(np.pi*a/W)))

#net section yield criteria
sc_y_LT = s_ys_LT*(W-2*a)/W
sc_y_L = s_ys_L*(W-2*a)/W

#preliminary plot
plt.figure(figsize=(12,6))
plt.plot(a,sc_f_TL_RT,label='fracture, T-L, RT')
plt.plot(a,sc_f_LT_RT,label='fracture, L-T, RT')
plt.plot(a,sc_y_LT,label='yield, LT')
plt.plot(a,sc_y_L,label='yield, L')
plt.xlabel('$a_{eff}$ (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.legend(loc='best')
plt.ylim([0,150])
plt.title('7178-T651 bare')
```

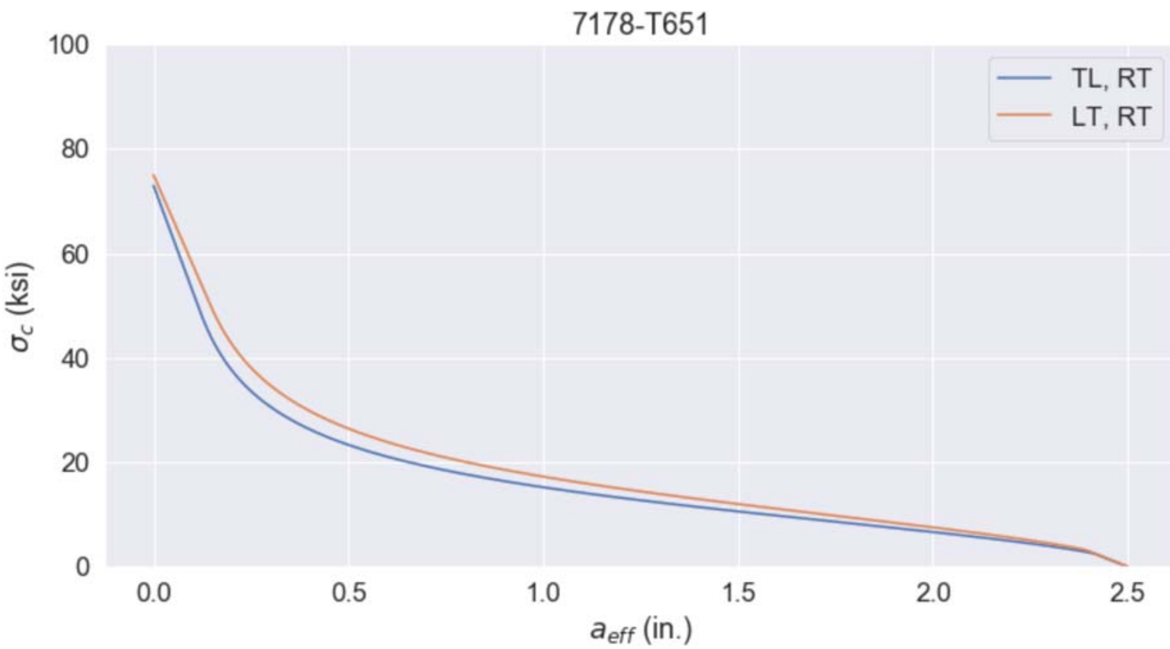
```
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:15: RuntimeWar
ning: divide by zero encountered in true_divide
  from ipykernel import kernelapp as app
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:16: RuntimeWar
ning: divide by zero encountered in true_divide
  app.launch_new_instance()
```

Out[6]: `Text(0.5, 1.0, '7178-T651 bare')`



In [7]:
```
#interpolations of fracture condition
spl_TL_RT = interpolate.splrep(a[1:],sc_f_TL_RT[1:])
spl_LT_RT = interpolate.splrep(a[1:],sc_f_LT_RT[1:])
#guess point of intersection
a0 = 0.1

#objective function for optimization
def myobj(a,spl=spl_TL_RT,sc_y=sc_y_LT):
    fa = interpolate.splev(a,spl,der=0)
    fprime = interpolate.splev(a,spl,der=1)
    return (sc_y - (fa-fprime*a))**2

#optimize
res = minimize(myobj,a0,args=(spl_TL_RT,sc_y_LT[0]))
a_int_TL_RT = res.x[0]
res = minimize(myobj,a0,args=(spl_LT_RT,sc_y_L[0]))
a_int_LT_RT = res.x[0]

#array to plot tangent line
a_tan_TL_RT = np.linspace(0,a_int_TL_RT)
a_tan_LT_RT = np.linspace(0,a_int_LT_RT)

#generate tangent lines
fa_TL_RT = interpolate.splev(a_int_TL_RT,spl_TL_RT,der=0)
fprime_TL_RT = interpolate.splev(a_int_TL_RT,spl_TL_RT,der=1)
fa_LT_RT = interpolate.splev(a_int_LT_RT,spl_LT_RT,der=0)
fprime_LT_RT = interpolate.splev(a_int_LT_RT,spl_LT_RT,der=1)

def mymin(a,a_int,fa,fprime,Kc,s_ys):
    if a < a_int:
        return fa+fprime*(a-a_int)
    else:
        return min([Kc/(np.sqrt(np.pi*a)*np.sqrt(1./np.cos(np.pi*a/W))),s_ys*(W-2*
a)/W])

plt.figure(figsize=(12,6))
plt.plot(a,[mymin(i,a_int_TL_RT,fa_TL_RT,fprime_TL_RT,Kc_TL_RT,s_ys_LT) for i in
a],label='TL, RT')
plt.plot(a,[mymin(i,a_int_LT_RT,fa_LT_RT,fprime_LT_RT,Kc_LT_RT,s_ys_L) for i in a],
label='LT, RT')
plt.xlabel('$a_{eff}$ (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.legend(loc='best')
plt.ylim([0,100])
plt.title('7178-T651')
```

Out[7]: Text(0.5, 1.0, '7178-T651')



4

We can find the proof load required for this case using $\sigma_c = \frac{K_c}{\sqrt{\pi a \beta}}$ with $2a = 0.25$

In [9]:
```
W = 8.0 #in
t = 0.4 #in.
a = .25/2 #in.

Kc_TL_RT = 62.5 #ksi sqrt(in)
Kc_LT_RT = 47.5 #ksi sqrt(in)

sc_TL_RT = Kc_TL_RT/(np.sqrt(np.pi*a)/np.cos(np.pi*a/W))
sc_LT_RT = Kc_LT_RT/(np.sqrt(np.pi*a)/np.cos(np.pi*a/W))
#print stress
print (sc_TL_RT)
print (sc_LT_RT)

#convert stress to load
print (sc_TL_RT*W*t)
print (sc_LT_RT*W*t)
```

```
99.6154342382702
75.70773002108535
318.76938956246465
242.26473606747314
```

So for the T-L grain direction at room temperature, a load of 153 k-lbs. would provide a proof test for a 0.25 inch crack in this panel. Similarly for L-T grain direction at room temperature, the proof load would be 173 k-lbs.

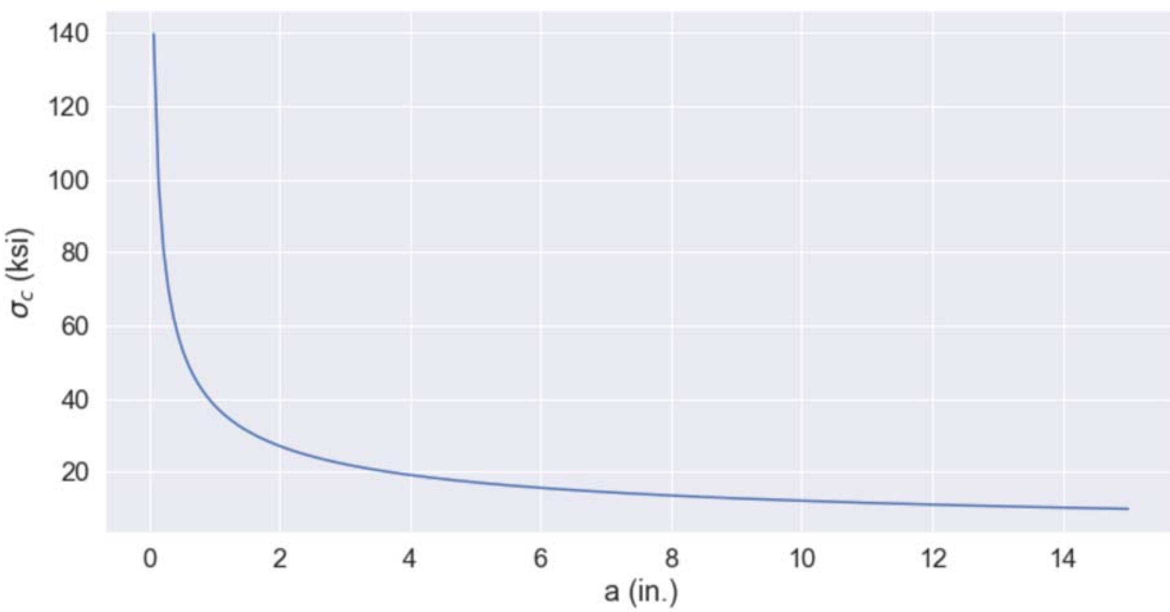
5

First we plot the residual strength of the skin without stiffeners

In [10]:
```
Kc = 68.0 #ksi sqrt(in)
b = 10.0 #inches (stiffener spacing)

#crack length array
a = np.linspace(0,1.5*b,200)

#fracture criteria
sc_f = Kc/(np.sqrt(np.pi*a))
#ignore net-section yield
#plot
plt.figure(figsize=(12,6))
plt.plot(a,sc_f,label='skin without stiffener')
plt.xlabel('a (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
```

```
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:8: RuntimeWarn
ing: divide by zero encountered in true_divide
```

Out[10]:
```
Text(0, 0.5, '$\\sigma_c$ (ksi)')
```



To find the residual strength with stiffeners we first need to find the parameter μ, which will indicate which carts to use. We find

In [11]:
```
t = 0.1875 #in
As = 0.3788 #in^2
Es = 23.4e3 #ksi
A = b*t #in^2
E = 11.0e3 #ksi

#parameter for charts
mu = As*Es/(As*Es+A*E)
print(mu)
```

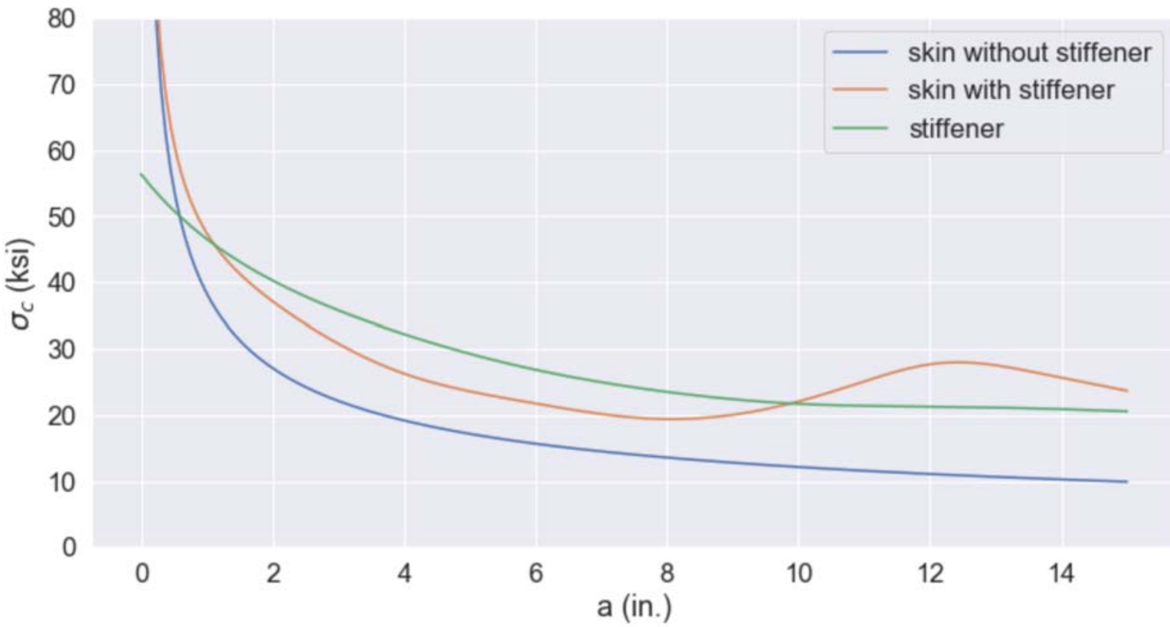
```
0.30058476200552614
```

We find $\mu = 0.30$, so we use that to look up the appropriate β and L values from pp. 167-178.

In [12]:
```
#beta values from chart
a_b = np.arange(0,3.2,0.2)
beta = np.array([1.0, 0.73, 0.73, 0.72, 0.70, 0.55, 0.4, 0.4, 0.45, 0.50, 0.32, 0.29, .30, 0.31, 0.38, 0.25])
L = np.array([1.0, 1.4, 1.75, 2.1, 2.4, 2.6, 2.65, 2.7, 2.8, 2.9, 2.90, 2.92, 2.94, 2.96, 2.98, 3.0])

#interpolate between data points
beta_i = interpolate.splrep(a_b*b,beta)
L_i = interpolate.splrep(a_b*b,L)

#stiffened skin
sc_f_s = Kc/(np.sqrt(np.pi*a)*interpolate.splev(a,beta_i))

#stiffener
s_ys = 120.0 #ksi
sc_st = s_ys*E/(Es*interpolate.splev(a,L_i))

#plot
plt.figure(figsize=(12,6))
plt.plot(a,sc_f,label='skin without stiffener')
plt.plot(a,sc_f_s,label='skin with stiffener')
plt.plot(a,sc_st,label='stiffener')
plt.xlabel('a (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.legend(loc='best')
plt.ylim([0,80])
```

```
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:11: RuntimeWarning: divide by zero encountered in true_divide
  # This is added back by InteractiveShellApp.init_path()
```

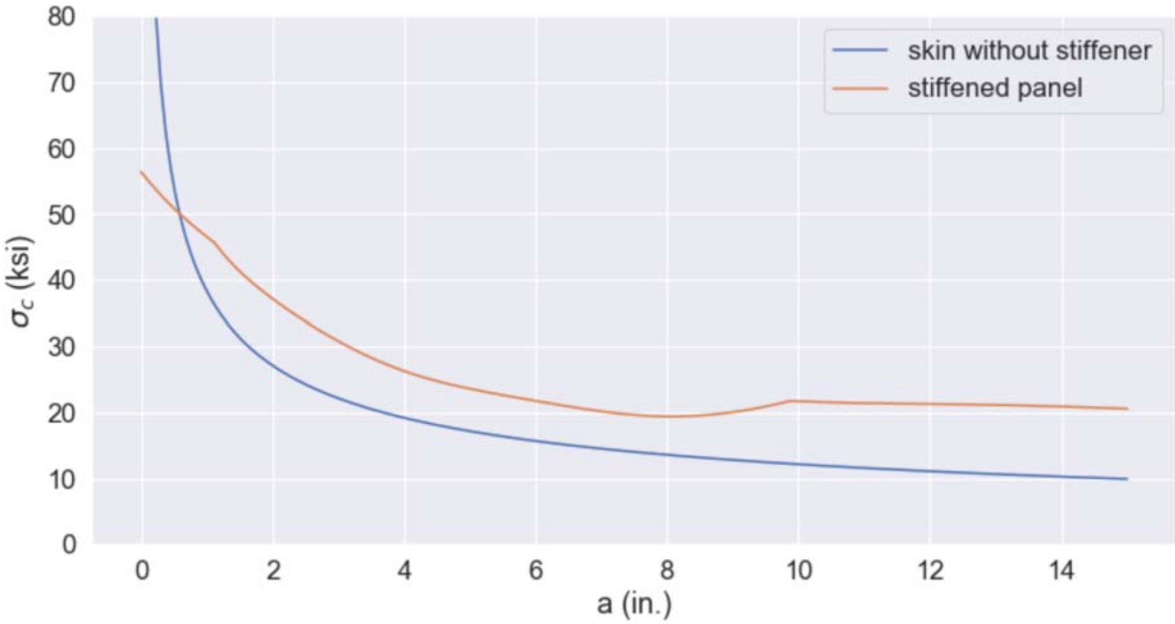
Out[12]: (0, 80)



The residual strength of the stiffened panel will be the minimum of the skin/stiffener residual strength

In [13]:
```
#plot
plt.figure(figsize=(12,6))
plt.plot(a,sc_f,label='skin without stiffener')
plt.plot(a,[min([sc_st[i],sc_f_s[i]]) for i in range(len(a))],label='stiffened pane
l')
plt.xlabel('a (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.legend(loc='best')
plt.ylim([0,80])
```

Out[13]: (0, 80)

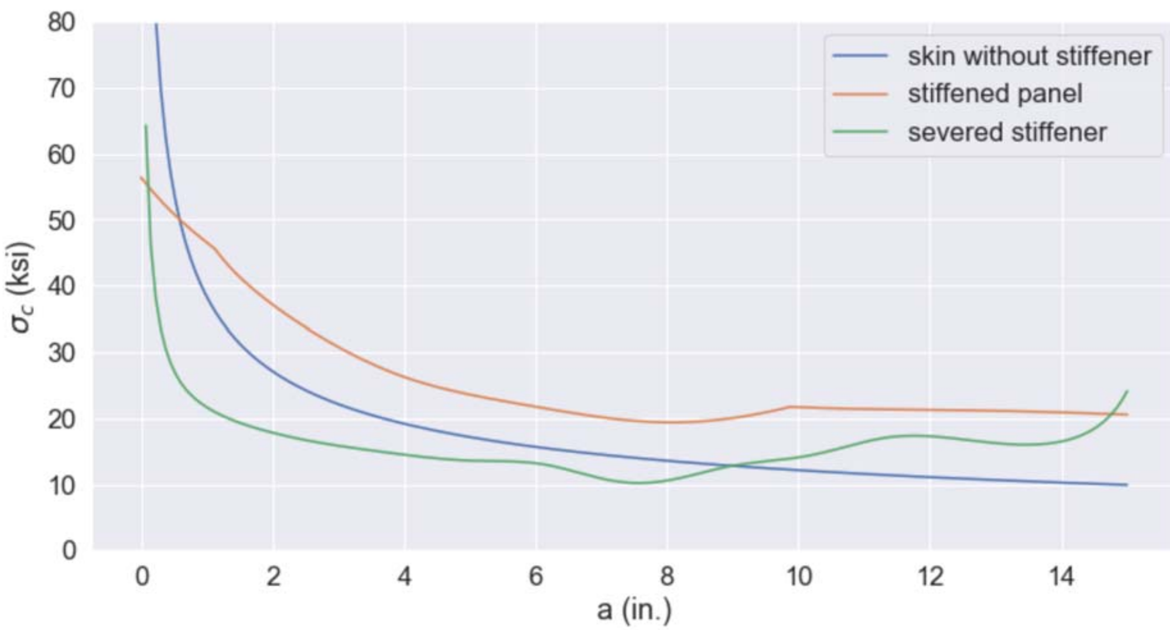


With the stiffener severed, we need to look up the new β values from case 11 on p. 195.

In [14]:
```
a_severed = np.array([.3125,.625,.9375,1.25,2.5,3.75,5.0,6.25,7.5,8.75,10.0,11.25,1
2.5,13.75])
beta_severed = np.array([2.0766,1.9305,1.7930,1.6874,1.4530,1.3384,1.2625,1.2002,1.
374,1.0552,0.8624,0.6754,0.6462,0.6418])

#interpolate between data points
beta_i_sev = interpolate.splrep(a_severed,beta_severed)

#stiffened skin
sc_f_sev = Kc/(np.sqrt(np.pi*a)*interpolate.splev(a,beta_i_sev))

#plot
plt.figure(figsize=(12,6))
plt.plot(a,sc_f,label='skin without stiffener')
plt.plot(a,[min([sc_st[i],sc_f_s[i]]) for i in range(len(a))],label='stiffened pane
l')
plt.plot(a,sc_f_sev,label='severed stiffener')
plt.xlabel('a (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.legend(loc='best')
plt.ylim([0,80])
```

```
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:8: RuntimeWarn
ing: divide by zero encountered in true_divide
```

Out[14]: (0, 80)



6

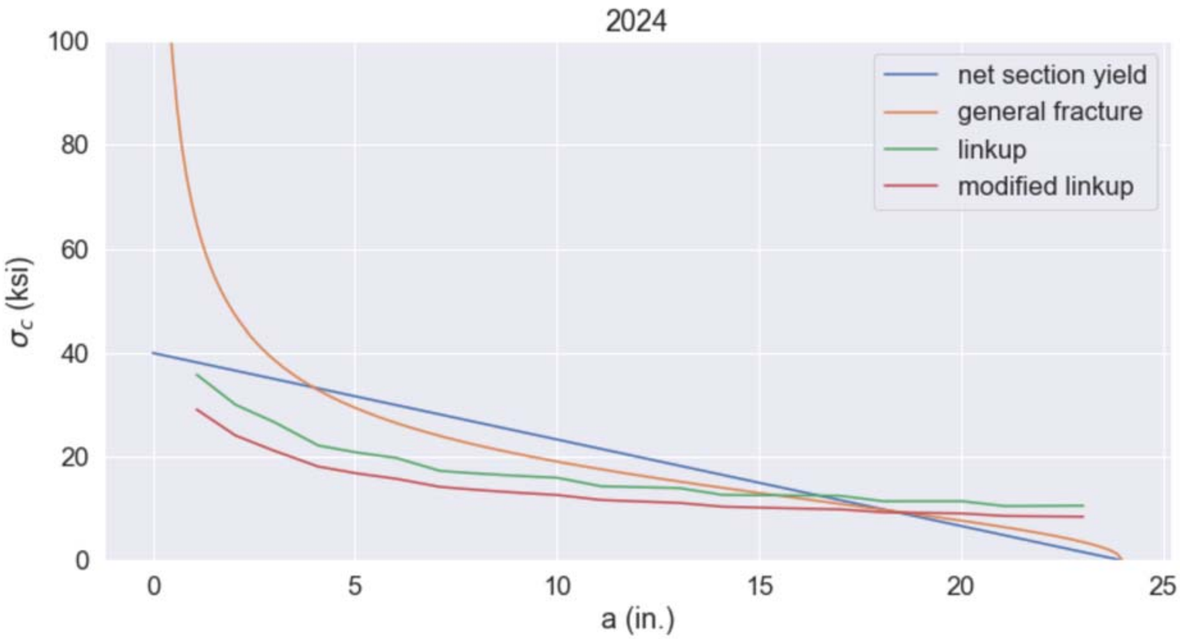
Net Section yield and brittle fracture will be calculated the same way as we have done previously (where brittle fracture only considers the largest, "lead" crack").

For linkup, we use (11.5f), while for the modified linkup the equation we use will depend on the material. (11.6) for 2024 and 2524 and either (11.7) or (11.8) for 7075.

In [15]:
```
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sb
sb.set(font_scale=1.5)
%matplotlib inline

#panel properties
a0 = 0.0 #inches, initial crack length
W = 48.0 #inches, panel width
a = np.linspace(a0,W/2,200) #crack length array (from a0 to end of panel, W/2)
t = 0.063 #inches, panel thickness
c = 0.05 #inches, MSD crack(s)
d = 0.1875 #inches, hole diameter
r = d/2 #inches, hole radius
p = 1.0 #inches, hole pitch
l = c + r #inches, parameter for link-up equation

#2024
s_ys = 40.0 #ksi, yield strength
kc = 120.0 #ksi sqrt(in), fracture toughness
A1 = 0.3054 #b-basis
A2 = 1.3502 #b-basis

#net section yield
s_net = s_ys*(W/2-a)/(W/2)

#general fracture
s_frac = kc/(np.sqrt(np.pi*a/np.cos(np.pi*a/W)))

#beta for crack near a hole, p. 53-54
beta_a = 0.934
beta_l = 2.268

s_msd = [] #empty array
s_msd_mod = [] #empty array
#loop through crack values to find L
for i in a:
    #i%1.5 gives remainder of i/1.5
    #calculate L
    L = p - d - c - i%p
    if L < c:
        s_msd.append(0) #this means crack tip is inside a hole or msd crack
        s_msd_mod.append(0)
    else:
        #otherwise we proceed with the usual calculation
        sc = s_ys*np.sqrt(2*L/(i*beta_a**2 + l*beta_l**2))
        s_msd.append(sc)
        #modified linkup
        sc_mod = sc/(A1*np.log(L) + A2)
        s_msd_mod.append(sc_mod)


#create line through local extrema
from scipy.signal import argrelextrema
s_msd = np.array(s_msd)
s_msd_mod = np.array(s_msd_mod)
vals = argrelextrema(s_msd,np.greater)
vals_mod = argrelextrema(s_msd_mod,np.greater)


plt.figure(figsize=(12,6))
plt.plot(a,s_net,label='net section yield')
plt.plot(a,s_frac,label='general fracture')
#plt.plot(a,s_msd,label='linkup')
```

```
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:28: RuntimeWar
ning: divide by zero encountered in true_divide
```

Out[15]: `Text(0.5, 1.0, '2024')`

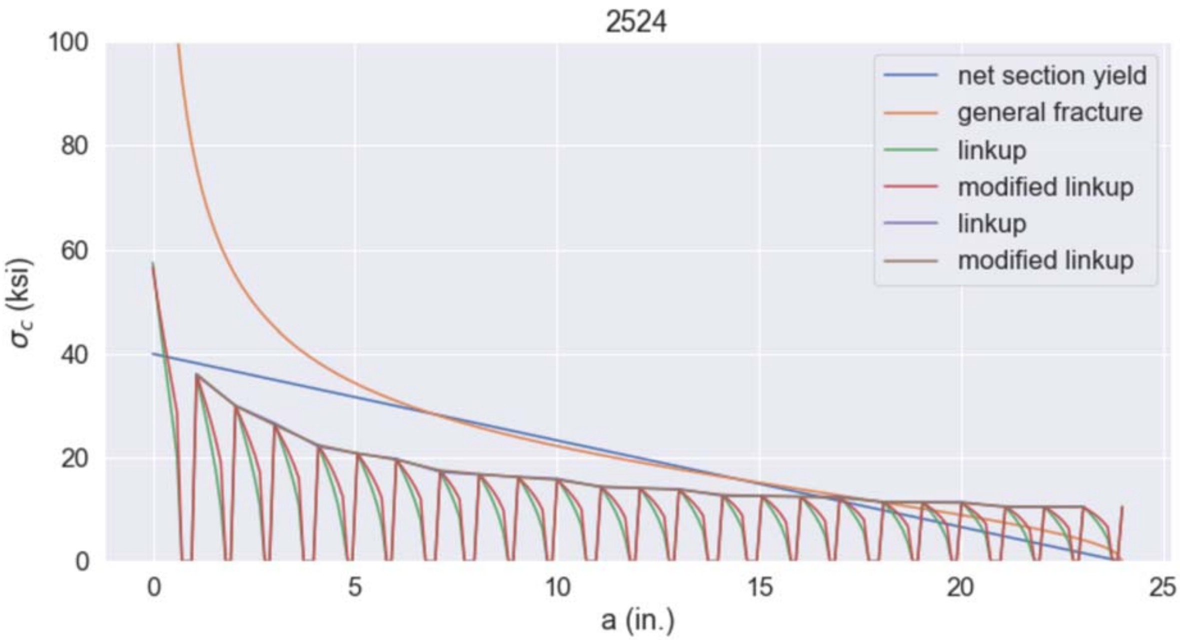


We can see the effect of using a constant β for the lead crack in this plot, as the MSD predictions begin to taper off as the crack gets longer. On a real panel there are many complicated effects as the crack propagates (re-distribution of fastener loads at each of the holes, stiffeners and crack stoppers, etc.) so the variable β is generally determined using Finite Elements.

In [16]:
```
#2524
s_ys = 40.0 #ksi, yield strength
kc = 140.0 #ksi sqrt(in), fracture toughness
A1 = 0.2024 #b-basis
A2 = 1.0719 #b-basis

#net section yield
s_net = s_ys*(W/2-a)/(W/2)

#general fracture
s_frac = kc/(np.sqrt(np.pi*a/np.cos(np.pi*a/W)))

#beta for crack near a hole, p. 53-54
beta_a = 0.934
beta_l = 2.268

s_msd = [] #empty array
s_msd_mod = [] #empty array
#loop through crack values to find L
for i in a:
    #i%1.5 gives remainder of i/1.5
    #calculate L
    L = p - d - c - i%p
    if L < c:
        s_msd.append(0) #this means crack tip is inside a hole or msd crack
        s_msd_mod.append(0)
    else:
        #otherwise we proceed with the usual calculation
        sc = s_ys*np.sqrt(2*L/(i*beta_a**2 + l*beta_l**2))
        s_msd.append(sc)
        #modified linkup
        sc_mod = sc/(A1*np.log(L) + A2)
        s_msd_mod.append(sc_mod)


#create line through local extrema
s_msd = np.array(s_msd)
s_msd_mod = np.array(s_msd_mod)
vals = argrelextrema(s_msd,np.greater)
vals_mod = argrelextrema(s_msd_mod,np.greater)


plt.figure(figsize=(12,6))
plt.plot(a,s_net,label='net section yield')
plt.plot(a,s_frac,label='general fracture')
plt.plot(a,s_msd,label='linkup')
plt.plot(a,s_msd_mod,label='modified linkup')
plt.plot(a[vals],s_msd[vals],label='linkup')
plt.plot(a[vals_mod],s_msd_mod[vals_mod],label='modified linkup')
plt.legend(loc='best')
plt.xlabel('a (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.ylim([0,100])
plt.title('2524')
```

```
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:11: RuntimeWar
ning: divide by zero encountered in true_divide
  # This is added back by InteractiveShellApp.init_path()
```

Out[16]: `Text(0.5, 1.0, '2524')`



It is interesting to note that the modified linkup equations do not change the strength prediction in any perceptible way for 2524. We can examine the effect of the modification by checking the value of $A_1 \ln(L) + A_2$

In [17]:
```python
for L in np.linspace(0,p-d-c,10):
    print (A1*np.log(L) + A2)
```

```
-inf
0.5723004247978338
0.7125934141431667
0.7946595520242592
0.8528864034884996
0.8980506582744957
0.9349525413695922
0.9661526389666292
0.9931793928338326
1.0170186792506846

C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:2: RuntimeWarn
ing: divide by zero encountered in log
```

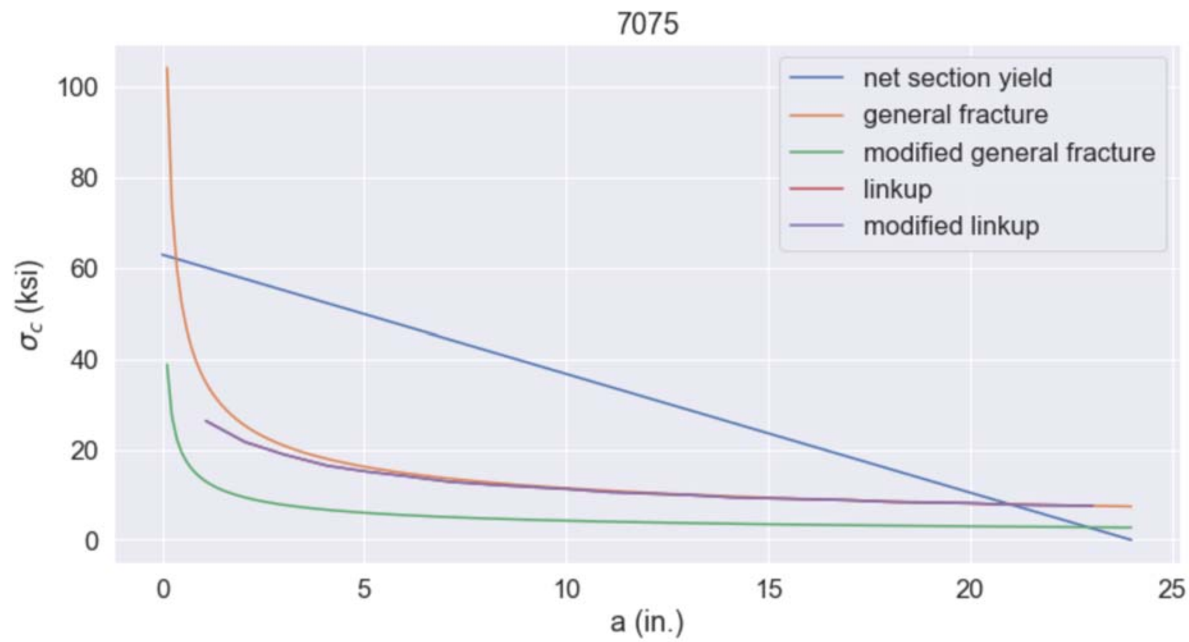
We observe that while the term does have some effects, they do not change the local maxima, and thus do not effect the residual strength.

For 7075 we will compare both (11.7) and (11.8)

In [18]:
```
#7075
s_ys = 63.0 #ksi, yield strength
kc = 60.0 #ksi sqrt(in), fracture toughness
B1 = 1.417 #b-basis
B2 = 1.073 #b-basis

#net section yield
s_net = s_ys*(W/2-a)/(W/2)

#beta given in problem
beta_a = 0.934
beta_l = 2.268

#general fracture
s_frac = kc/(np.sqrt(np.pi*a)*beta_a)
#modified fracture, 11.8
s_frac_mod = kc/(np.sqrt(np.pi*a)*beta_a*(.856-.946*np.log(l)))

s_msd = [] #empty array
s_msd_mod = [] #empty array
#loop through crack values to find L
for i in a:
    #i%1.5 gives remainder of i/1.5
    #calculate L
    L = p - d - c - i%p
    if L < 0:
        s_msd.append(0) #this means crack tip is inside a hole or msd crack
        s_msd_mod.append(0)
    else:
        #otherwise we proceed with the usual calculation
        sc = s_ys*np.sqrt(2*L/(i*beta_a**2 + l*beta_l**2))
        s_msd.append(sc)
        sc_mod = sc/(B1+B2*L)
        s_msd_mod.append(sc_mod)

#create line through local extrema
s_msd = np.array(s_msd)
s_msd_mod = np.array(s_msd_mod)
vals = argrelextrema(s_msd,np.greater)
vals_mod = argrelextrema(s_msd_mod,np.greater)

plt.figure(figsize=(12,6))
plt.plot(a,s_net,label='net section yield')
plt.plot(a,s_frac,label='general fracture')
plt.plot(a,s_frac_mod,label='modified general fracture')
plt.plot(a[vals],s_msd_mod[vals],label='linkup')
plt.plot(a[vals_mod],s_msd_mod[vals_mod],label='modified linkup')
plt.legend(loc='best')
plt.xlabel('a (in.)')
plt.ylabel('$\sigma_c$ (ksi)')
plt.title('7075')
```

```
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:15: RuntimeWar
ning: divide by zero encountered in true_divide
  from ipykernel import kernelapp as app
C:\ProgramData\Miniconda3\lib\site-packages\ipykernel_launcher.py:17: RuntimeWar
ning: divide by zero encountered in true_divide
```

Out[18]: Text(0.5, 1.0, '7075')



7

For the maximum circumferential stress criterion, we use (11.14) to solve for the crack extension angle

In [19]:
```python
import numpy as np

tau = 1.0
sigma = 5.0
K_IC = 70.0 #ksi sqrt(in)
a = 0.75 #in

K_I = sigma*np.sqrt(np.pi*a)
K_II = tau*np.sqrt(np.pi*a)

#import a solver library to solve non-linear trig equations
from scipy import optimize

def myeqn(theta):
    return K_I*np.sin(theta) + K_II*(3*np.cos(theta)-1)

sol = optimize.root(myeqn,0)
theta_p = sol.x[0]
print (theta_p*180/np.pi)
```

```
-21.088795134252223
```

We find the crack will extend along $\theta = -21.1^\circ$.

Next we substitute this angle into (11.15) to solve for the critical fracture stress.

In [20]:
```
print (3*np.cos(theta_p/2) + np.cos(3*theta_p/2))
print (np.sin(theta_p/2) + np.sin(3*theta_p/2))
```

```
3.8007634378294752
-0.707476621033822
```

Substituting into (11.15) we have

$$4(60) = 3.80K_I - 3(-0.707)K_{II}$$

Since $\tau_c = \frac{1}{5}\sigma_c$, we can substitute and solve

$$240 = 3.80\sigma_c\sqrt{.5\pi} + 2.121/5\sigma_c\sqrt{.5\pi}$$

In [21]:
```
sc = 240.0/(np.sqrt(np.pi*a)*(3.80+2.121/5.0))
print (sc)
```

```
37.01358927345637
```

We find $\sigma_c = 37.0$ ksi

For the principal stress criterion, we ignore the effects of the crack stress field and find the direction of the principal stresses using (11.17d)

In [22]:
```
theta_p = np.arctan(2.0*tau/(-sigma))/2.0
print (theta_p*180/np.pi)
```

```
-10.900704743175906
```

Here we find that $\theta_p = -10.9^\circ$, which we can substitute into (11.17b) to find the hoop stress in the principal direction.

In [23]:
```
sigma_theta = sigma*np.cos(theta_p)**2 - 2*tau*np.sin(theta_p)*np.cos(theta_p)
print (sigma_theta)
```

```
5.192582403567252
```

Thus $\sigma_\theta = 5.19\tau$ or $\sigma_\theta = 5.19/5.0\sigma$

We can now find σ_c according to (11.19)

In [24]:
```
sc = K_IC/(sigma_theta/sigma*np.sqrt(np.pi*a))
print (sc)
```

```
43.91158152792188
```

And we predict $\sigma_c = 43.9$ ksi. In this case we see that the principal stress criterion gives a non-conservative estimate of the critical stress that is not very close to the circumferential stress criterion.

In [ ]: