FAVOR Flaw creation scripts (SAND2022-15957 O)

written by Michael J. Starr and Nathan Porter
Sandia National Laboratories[1]
P. O. Box 8500
Albuquerque, NM 87185

This document contains the notated versions of MATLAB and Python scripts that were developed to support analysis performed in the NRC Technical Letter Report, TLR-RES/DE/REB-2021-16: "Probabilistic Fracture Mechanics Application: FAVOR Case Study", available under ADAMS Accession Number ML21267A469.

[1] Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

1A. MATLAB Flaw script: ‘create_isb_flaw_files.m’

This script must be used with the matlab function: ‘write_isb_flaw.m’

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create_isb_flaw_files.m
% written by MJS
% 2019-09-24
% updated 2020-11-24
%
% This version of the flaw file creator script was written to use with the
% RPV geometry and flaw definition defined for the SMiRT 2019 benchmarking
% problem. The script allows the user to input a vector of uncertainty
% values and write full sets of flaw files for each of the uncertainty
% values in the uncertainty vector.
%
% In this script I will sample crack depth and crack length for both Flaw A
% and Flaw B. It is assumed that there is only one analysis option:
%
% Both cracks are surface-breaking cracks, based on classification of
% the nominal crack parameters. Bounds and distributions will be applied to
% each of the five crack parameters that characterize the two flaws.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Vessel Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rpv.ri = 86; % inches
rpv.clad = 0.25; % inches
rpv.wall = 8.5; % inches
rpv.height = 156; % inches

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Flaw Properties
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% For all cases, flaw parameters will be bounded. For example, the minimum
% crack depth is 0.25 inches (so that the crack penetrates beyond the inner
% cladding). There are practical limitations to the crack lengths and outer
% depths, so all distributions will be bounded.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Inspector Data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The observed flaws have the following nominal characteristics
d_a = 0.26; % Flaw A depth
L_a = 1.4; % Flaw A length
d_b = 0.40; % Flaw B outer depth
s_b = 0.05; % Flaw B inner depth
L_b = 2.2; % Flaw B length

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Sampling
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Let's just use a uniform distribution over the defined flaw parameters.
% These samples can always be transformed to our distribution of choice as
```

```
% needed using the appropriate inverse CDF. This is done

% Sample parameters
N = 1000000;

% Let's start the engine! Sample and start the machinery. The columns of
% random numbers are applied to calculate the following:
% 1. Flaw A crack depth
% 2. Flaw B crack depth
% 3. Flaw A crack length
% 4. Flaw B crack length
% 5. Flaw B inner depth

crack = rand(N,5);

% For now let's just assume we are bounded in a regime around the
% inspector's observations using a global parameter uncertainty.
unc = [0.01 0.02 0.05 0.10 0.20 0.25 0.50 1.00];

for j = 1:size(unc,2)

    adepth = [max([0.25 (1-unc(j))*d_a]) min([(1+unc(j))*d_a rpv.wall+rpv.clad])];
    bdepth = [max([0.25 (1-unc(j))*d_b]) min([(1+unc(j))*d_b rpv.wall+rpv.clad])];
    alength = [(1-unc(j))*L_a min([(1+unc(j))*L_a 2*pi*(rpv.ri+rpv.clad+rpv.wall)])];
    blength = [(1-unc(j))*L_b min([(1+unc(j))*L_b 2*pi*(rpv.ri+rpv.clad+rpv.wall)])];
    sdepth = [max([0 0.05-unc(j)*0.2]) min([0.05+unc(j)*0.2 0.25])];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Flaw File Bins
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % In FAVOR, there are two sets of bins required for the different flaw
    % depending on whether the flaws are surface-breaking or embedded. Both
    % types of flaw files use the same definition for normalized crack depth.
    % The aspect ratios for the surface-breaking flaw file are single numbers,
    % so the definitions for those bins are arbitrary, but can be made
    % conservative by rounding all values to the next highest value. The
    % embedded flaw bins are explicitly defined in FAVOR.

    % Note that only the surface-breaking bins are used in this script.

    ar_s = [2 6 10 Inf];
    ar_s_breaks = [2 6 10];

    ar_e = [1 1.25 1.5 2 3 4 5 6 8 10 15];
    ar_e_breaks = [1.25 1.5 2 3 4 5 6 8 10 15];

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%       Analysis
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    distr = 'uniform'; % Choose uniform or normal distribution for crack parameters

    nstr = num2str(100*unc(j)); % Flaw file naming convention
    if length(nstr)<2
        nstr = strcat('0',nstr);
    end
```

```
    % Transform the uniform samples into crack parameters
    if strcmp(distr,'uniform')
        a_outer = adepth(1) + (adepth(2)-adepth(1))*crack(:,1);
        a_long = alength(1) + (alength(2)-alength(1))*crack(:,3);
        b_outer = bdepth(1) + (bdepth(2)-bdepth(1))*crack(:,2);
        b_long = blength(1) + (blength(2)-blength(1))*crack(:,4);
        b_inner = sdepth(1) + (sdepth(2)-sdepth(1))*crack(:,5);
        isbstr = ['isb',nstr,'_uniform.dat'];
    end
    if strcmp(distr,'normal')
        pd = makedist('Normal');
        t_a = truncate(pd,(adepth(1)-d_a)*sqrt(12)/(adepth(2)-adepth(1)),Inf); % Truncate
normal distribution for Flaw A depth
        t_al = truncate(pd,(alength(1)-L_a)*sqrt(12)/(alength(2)-alength(1)),Inf); %
Truncate normal distribution for Flaw A depth
        t_bi = truncate(pd,(sdepth(1)-s_b)*sqrt(12)/(sdepth(2)-sdepth(1)),Inf); %
Truncate the normal distribution for Flaw B inner depth (must be positive)
        t_bo = truncate(pd,(bdepth(1)-d_b)*sqrt(12)/(bdepth(2)-bdepth(1)),Inf); %
Truncate the normal distribution for Flaw B outer depth (must be positive)
        t_bl = truncate(pd,(blength(1)-L_b)*sqrt(12)/(blength(2)-blength(1)),Inf); %
Truncate normal distribution for Flaw A depth
        a_outer = d_a + (adepth(2)-adepth(1))/sqrt(12)*icdf(t_a,crack(:,1));
        a_long = L_a + (alength(2)-alength(1))/sqrt(12)*icdf(t_al,crack(:,3));
        b_outer = d_b + (bdepth(2)-bdepth(1))/sqrt(12)*icdf(t_bo,crack(:,2));
        b_long = L_b + (blength(2)-blength(1))/sqrt(12)*icdf(t_bl,crack(:,4));
        b_inner = s_b + (sdepth(2)-sdepth(1))/sqrt(12)*icdf(t_bi,crack(:,5));
        isbstr = ['isb',nstr,'_normal.dat'];
    end

    AR_a = a_long./a_outer; % These aspect ratio calculations are appropriate if the
cracks are surface-breaking
    AR_b = b_long./b_outer;
    dnorm_a = round(100*a_outer/(rpv.clad + rpv.wall))/100;
    dnorm_b = round(100*b_outer/(rpv.clad + rpv.wall))/100;

    n_depth = min([min(dnorm_a) min(dnorm_b)]):0.01:max([max(dnorm_a) max(dnorm_b)]);

    % Fill the bins
    apdf = zeros(size(n_depth,2),size(ar_s,2)); % Flaw A - all surface-breaking
    bpdf = zeros(size(n_depth,2),size(ar_s,2)); % Flaw B - all surface-breaking
    for i = 1:N
        rid_a = find(abs(n_depth - dnorm_a(i)) < 0.005);
        cid_a = length(find(ar_s_breaks < AR_a(i))) + 1;
        apdf(rid_a,cid_a) = apdf(rid_a,cid_a) + 1;

        rid_b = find(abs(n_depth - dnorm_b(i)) < 0.005);
        cid_b = length(find(ar_s_breaks < AR_b(i))) + 1;
        bpdf(rid_b,cid_b) = bpdf(rid_b,cid_b) + 1;
    end

    % Normalize the depth density. This is how the depth density column will be
    % populated in the flaw file (this vector will be scaled by the flaw areal
    % density calculated for a given pressure vessel)

    dpdf = sum(apdf + bpdf,2)/N;

    % Normalize by samples at each depth. This is how the aspect ratio columns
    % will be populated in the flaw file

    ARpdf = (apdf+bpdf)./sum(apdf+bpdf,2);
```

```
    ARpdf(isnan(ARpdf)) = 0;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Create flaw file for this analysis option
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    isbnum = 2; % the expected number of flaws
    write_isb_flaw(n_depth,dpdf,ARpdf,rpv,isbstr,isbnum);

end
```

1B. MATLAB Flaw function: 'write_isb_flaw.m'

This function must be used with the matlab file: 'create_isb_flaw_files.m'

```
function [] = write_isb_flaw(d_norm,d_rho,ar_rho,g,fname,fnum)
% Use this function to write the inner surface-breaking flaw file. The
% function is passed two vectors, one matrix, one structure and one string
% and writes a file compatible with the syntax required to run FAVOR in PFM
% mode.
%
% d_norm = a nx1 vector that contains the real values of all the
% non-zero normalized depths of sampled cracks.
% d_rho = an nx1 vector that contains the flaw densities as a function of
% crack depth.
% ar_rho = an nxm matrix that contains the aspect ratio distributions at
% each depth.
% g = a structure containing the geometric parameters for the rpv under
% investigation.
% fname = the string that contains the name of the output file.
% fnum = the expected integer number of flaws

% FAVOR hard-coded parameters

Fidx = 161; %FAVOR version number, first entry of flaw file
nr = 1000; %number of records, required per FAVOR

% RPV surface area
a_rpv = 2*pi*g.ri*g.height/12^2;

% This is the flaw density over the entire vessel (beltline) volume
rho_f = 1/a_rpv;

% Here we perform a check to guarantee that we produce the expected number
% of flaws. (Actually, we'll scale to produce 0.1% above the expected number
% of whole number flaws. FAVOR will round down to the closest whole number)
d_rho = fnum*1.001/sum(d_rho)*d_rho;

% The distribution across the allowed depths is then calculated by scaling
% the density.
f_dis = rho_f*d_rho;

% The aspect ratio distribution for all allowed depths is calculated by
% scaling the distribution calculated above by 100.
ar_dis = ar_rho*100;

% Densities for depths with no flaws
rhozero = 0;
dzero = 0;

% Write dataset file
fdepth = round(100*d_norm);
fid = fopen(fname,'w');
fprintf(fid,'%s\n',num2str(Fidx));
for j = 1:nr
    for i = 1:100
        if ~isempty(find(i==fdepth,1))
            idx = find(i==fdepth,1);
```

```
            fprintf(fid,'%12u %15.8f %11.5f %12.5f %12.5f
%12.5f\n',i,ceil(1e8*f_dis(idx))/1e8,ar_dis(idx,1),ar_dis(idx,2),ar_dis(idx,3),ar_dis(idx
,4));
        else
            fprintf(fid,'%12u %14.7f %12.5f %12.5f %12.5f
%12.5f\n',i,rhozero,dzero,dzero,dzero,dzero);
        end
    end
end
fclose(fid);

end
```

2. Python Flaw script: ‘create_isb_flaw_files.py’

```
create_isb_flaw_files.py
translated by Nathan Porter
2021-06-01

This Python script has been translated from
MATLAB to create flaw files for the SMiRT study.

NOTE: flaws in the plate region are not implemented

For help with input options:
    python create_flaw_files.py --help
'''
from matplotlib import pyplot as plt
import numpy as np
from scipy.stats import truncnorm, norm
import argparse

# set up argparse arguments
p = argparse.ArgumentParser(
  description='''Create flaw files for FAVOR input. All inputs are optional
  EXAMPLES:
  python create_flaw_files.py
  python create_flaw_files.py --seed 123 --mixed --distributions norm
  python create_flaw_files.py --epsilon 0.5 1.0 -dist N U
  python create_flaw_files.py -eps 0.01 0.05 0.1 -s 123 -m -dist norm uni''',
  formatter_class=argparse.RawTextHelpFormatter
                            )
default_eps = [0.01, 0.02, 0.05, 0.10, 0.20, 0.25, 0.50, 1.00]
p.add_argument('--epsilon', '-eps', nargs='+', type=float, default=default_eps,
               help='list of values for epsilon (example \"-eps 0.01 0.02\")'
               )
p.add_argument('--distributions', '-dist', nargs='+', type=str, default=['norm', 'uni'],
               help='distributions to create flaw files for (example \"-dist norm U\")'
               )
p.add_argument('--seed', '-s', type=int, default=123,
               help='random seed value'
               )
p.add_argument('--mixed', '-m', action='store_true',
               help='make Flaw B mixed (default surface breaking)'
               )
args = p.parse_args()

print('\n***Processing user input', flush=True)

# set a seed so the results are repeatable
np.random.seed(args.seed)

# save the argument to set Flaw B as mixed
flaw_B_mixed = args.mixed

# save the argparse argument for epsilon and make sure they are
# between 0 and 1
epsilons = args.epsilon
for eps in epsilons:
    if eps<0.0 or eps>1.0:
        raise p.error('epsilon value not between 0 and 1: '+str(eps))
```

```python
# read the argparse argument for distributions and set up list
# of distribution types to create flaw files for
distributions = []
for dist in args.distributions:
    if dist in ['n', 'N', 'norm', 'normal', 'Norm', 'Normal']:
        distributions.append('normal')
    elif dist in ['u', 'U', 'uni', 'uniform', 'Uni', 'Uniform']:
        distributions.append('uniform')
    else:
        raise p.error('distribution not implemented: '+dist)

# print debug output so the user can confirm input was read correctly
if flaw_B_mixed:
    print('***Flaw B is mixed, creating im and wm files', flush=True)
else:
    print('***Flaw B is surface breaking, creating isb files', flush=True)
print('***Random seed: '+str(args.seed), flush=True)
print('***Distribution types from user input:'+(len(distributions)*'\n***
{:s}').format(*distributions))
print('***Uncertainty values from user input (epsilon):'+(len(epsilons)*'\n***
{:.2f}').format(*epsilons))
print('***User input processing complete. Creating flaw files.', flush=True)


def my_norm(bounds, mean, size=1, truncate=True):
    '''
    Samples from a truncated normal that is consistent with the MATLAB
    script. The distribution is centered about the nominal value with
    standard deviation equal to the bound divided by sqrt(12). Turn off
    truncate to sample from a traditional normal distribution.
        bounds (tuple) minimum and maximum of the distribution
        mean (float) nominal value of the distribution
        size (int) number of samples returned
        truncate (boolean) which distribution to use
    '''
    if truncate:
        stdev = (bounds[1] - bounds[0]) / np.sqrt(12.0)
        # lower bound is relative to mean/stdev in the truncnorm function
        lower = (bounds[0] - mean) / stdev
        dist = truncnorm(lower, np.inf, loc=mean, scale=stdev)
    else:
        stdev = (bounds[1]-bounds[0]) / 6.0
        dist = norm(loc=mean, scale=stdev)
    return dist.rvs(size=size)


def my_uni(bounds, size=1):
    '''
    Samples from a half-open uniform distribution [low, high)
    ****note: MATLAB samples from the open interval (low, high)
        bounds (tuple) minimum and maximum of the distribution
        size (int) number of samples returned
    '''
    return np.random.uniform(bounds[0], bounds[1], size=size)


def print_flaw_file(depths, bin_a, bin_b, A, Nf, filename):
    '''
    This function takes the processed data and prints out a flaw file
```

```
        depths (1D np array) array of depth bins
        bin_a (2D np array) flaw A bin distribution
        bin_b (2D np array) flaw A bin distribution
        A (float) normalizing area/volume
        Nf (int) expected number of flaws
        filename (str) name for the flaw file
    '''
    print('***    '+filename, flush=True)
    # first line is FAVOR version
    FAVOR_version = 161
    # number of records required by FAVOR
    Nr = 1000

    # sum the bins to get total counts
    if np.any(bin_b):
        binsum = np.sum(bin_a+bin_b, axis=1).reshape(-1, 1)
        # normalize by samples at each depth (get rid of NaNs)
        # multiply by 100 for FAVOR input
        AR_rho = 100.0*np.nan_to_num((bin_a+bin_b)/binsum)
    else:
        binsum = np.sum(bin_a, axis=1).reshape(-1, 1)
        # here, we are ignoring division by zero, since some bins have no counts
        with np.errstate(divide='ignore', invalid='ignore'):
            AR_rho = 100.0*np.nan_to_num(bin_a/binsum)

    width = len(AR_rho[0,:])
    if width not in [4, 11]:
        raise ValueError(width+' not a valid ')

    # normalize the depth density, which is how it is entered in the flaw file
    bins = binsum/N

    # check to guarentee that the expected number of flaws are produced.
    # a 0.1% factor is applied, since FAVOR will round down.
    # scale by flaw density
    d_rho = np.round((1.001*Nf/np.sum(bins)/A)*bins, decimals=6)

    depths = np.round(100.0*depths.reshape(-1,)).tolist()
    depths = [int(i) for i in depths]
    if width==4:
        fstr = '{:6d} {:15.6f}'+4*' {:12.5f}'+'\n'
    else:
        fstr = '{:6d} {:10.6f}'+11*' {:6.2f}'+'\n'
    with open(filename, 'w') as f:
        f.write('{:s}\n'.format(str(FAVOR_version)))
        for j in range(Nr):
            for i in range(100):
                if i+1 in depths:
                    # find the corresponding index
                    ind = np.where(np.isclose(depths, i+1))[0].tolist()[0]
                    # now print out the formatted line
                    # have to take care to convert numpy arrays to built-in python types
                    if width==4:
                        f.write(fstr.format(i+1, # index of this row
                                            d_rho[ind][0], # distribution across allowed
depths
                                            float(AR_rho[ind,0]), # all of the AR bins
                                            float(AR_rho[ind,1]),
                                            float(AR_rho[ind,2]),
                                            float(AR_rho[ind,3])
```

```
                                        ))
                    elif width==11:
                        f.write(fstr.format(i+1, # index of this row
                                        d_rho[ind][0]/2.0, # distribution across
allowed depths
                                        float(AR_rho[ind,0]), # all of the AR bins
                                        float(AR_rho[ind,1]),
                                        float(AR_rho[ind,2]),
                                        float(AR_rho[ind,3]),
                                        float(AR_rho[ind,4]),
                                        float(AR_rho[ind,5]),
                                        float(AR_rho[ind,6]),
                                        float(AR_rho[ind,7]),
                                        float(AR_rho[ind,8]),
                                        float(AR_rho[ind,9]),
                                        float(AR_rho[ind,10])
                                        ))
                else:
                    if width==4:
                        f.write(fstr.format(i+1, 0, 0, 0, 0, 0))
                    elif width==11:
                        f.write(fstr.format(i+1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))


# define the gemoetry in inches
rpv = {'ri': 86.0,
       'clad': 0.25,
       'wall': 8.5,
       'height': 156
       }

rpv['A'] = 2.0*np.pi*rpv['ri']*rpv['height']/12.0**2.0
rpv['A_weld'] = 2.0*3.0/8.0*np.pi*((rpv['ri']+rpv['clad']+rpv['wall'])**2.0
                                   -(rpv['ri']+rpv['clad'])**2.0
                                   )/12.0**2.0;

# nominal flaw characteristics in inches
d_a = 0.26
L_a = 1.4
d_b = 0.4
L_b = 2.2
s_b = 0.05

# sample size
N = 1000000

# define aspect ratio bins (reshape to column to be used later)
AR_surface = np.array([2.0, 6.0, 10.0, np.inf]).reshape(-1, 1)
AR_embedded = np.array([1.25, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0, 8.0, 10.0, 15.0,
np.inf]).reshape(-1, 1)

# loop over distribution types
for dist in distributions:

    print('***Creating flaw files for '+dist+' distribution', flush=True)

    # loop over each value of uncertainty
    for i, u in enumerate(epsilons):

        # calculate the bounds for each flaw characteristic
```

```
# ((tuples with min and max on separate lines for clarity))
bounds_d_a = (max(0.25, (1.0-u)*d_a),
              min((1.0+u)*d_a, rpv['wall']+rpv['clad'])
              )
bounds_L_a = ((1.0-u)*L_a,
              min((1.0+u)*L_a, 2.0*np.pi*(rpv['ri']+rpv['clad']+rpv['wall']))
              )
bounds_d_b = (max(0.25, (1.0-u)*d_b),
              min((1.0+u)*d_b, rpv['wall']+rpv['clad'])
              )
bounds_L_b = ((1.0-u)*L_b,
              min((1.0+u)*L_b, 2.0*np.pi*(rpv['ri']+rpv['clad']+rpv['wall']))
              )
bounds_s_b = (max(0.0, 0.05-0.2*u),
              min(0.05+0.2*u, 0.25)
              )

# sample crack parameters from uniform or normal distribution
if dist=='uniform':
    a_outer = my_uni(bounds_d_a, size=N)
    a_long = my_uni(bounds_L_a, size=N)
    b_outer = my_uni(bounds_d_b, size=N)
    b_long = my_uni(bounds_L_b, size=N)
    b_inner = my_uni(bounds_s_b, size=N)
elif dist=='normal':
    a_outer = my_norm(bounds_d_a, d_a, size=N)
    a_long = my_norm(bounds_L_a, L_a, size=N)
    b_outer = my_norm(bounds_d_b, d_b, size=N)
    b_long = my_norm(bounds_L_b, L_b, size=N)
    b_inner = my_norm(bounds_s_b, s_b, size=N)
else:
    raise ValueError(dist+' not implemented')

# calculate aspect ratios and normalize/round flaw depths
AR_a = a_long/a_outer
dnorm_a = np.round(a_outer/(rpv['clad']+rpv['wall']), decimals=2)
AR_b = b_long/b_outer
if flaw_B_mixed:
    b_2d = np.abs(b_outer - b_inner)
    # this aspect ratio is appropriate for embedded flaws
    AR_b_em = b_long/b_2d
    # check if Flaw B is surface breaking (sb=True) or mixed (sb=False)
    sb = b_outer-b_2d<0.4*b_2d/2.0
dnorm_b = np.round(b_outer/(rpv['clad']+rpv['wall']), decimals=2)

# create array of possible normalized flaw depths (for surface breaking flaws)
rmin = min(min(dnorm_a), min(dnorm_b))
rmax = max(max(dnorm_a), max(dnorm_b))
n_depth = np.arange(rmin, rmax+0.001, 0.01).reshape(-1, 1);

# instantiate matrices for binning
nrow, ncol_sb = len(n_depth), len(AR_surface)
bins_a = np.zeros((nrow, ncol_sb))
bins_b = np.zeros((nrow, ncol_sb))
if flaw_B_mixed:
    bins_b_sb = np.zeros((nrow, ncol_sb))
    ncol_em = len(AR_embedded)
    bins_b_em = np.zeros((nrow, ncol_em))

# bin each sampled value according to:
```

```
#     normalized flaw depth (ROW)
#     aspect ratio (COLUMN)
for i in range(nrow):
    for j in range(ncol_sb):
        # kind of complicated here...
        #  1. isclose() finds the row that each sample belongs to
        #  2. argmax() returns the corresponding index to that location
        #  3. i==... returns an array of booleans that is true if the
        #     sample in that location belongs in the bin corresponding
        #     to the current row
        row = i==np.argmax(np.isclose(n_depth, dnorm_a), axis=0)
        # this line is very similar, but for columns (with > instead of isclose)
        col = j==np.argmax(AR_surface>AR_a, axis=0)
        # now, count how many samples belong in the current row/column and
        # slot that number into the instatiated bin matrix
        bins_a[i, j] = np.count_nonzero(row&col)
        # repeat for flaw B
        row = i==np.argmax(np.isclose(n_depth, dnorm_b), axis=0)
        col = j==np.argmax(AR_surface>AR_b, axis=0)
        bins_b[i, j] = np.count_nonzero(row&col)
        if flaw_B_mixed:
            # if mixed, Flaw B surface breaking only counts sb cases
            bins_b_sb[i, j] = np.count_nonzero(sb&row&col)
    if flaw_B_mixed:
        # now count embedded flaws
        for j in range(ncol_em):
            row = i==np.argmax(np.isclose(n_depth, dnorm_b), axis=0)
            col = j==np.argmax(AR_embedded>AR_b_em, axis=0)
            bins_b_em[i, j] = np.count_nonzero(row&col&~sb)

# double check that all samples were included in the bins
if np.sum(bins_a)!=N or np.sum(bins_b)!=N:
    raise RuntimeError('Not counting all samples, bug in binning process')
if flaw_B_mixed:
    if np.sum(bins_b_sb)+np.sum(bins_b_em)!=N:
        raise RuntimeError('Not counting all samples, bug in binning process')

# flaw file name (based on uncertainty and distribution type)
filepost = str(int(100*u)).rjust(3, '0')+'_'+dist+'.dat'

# pass relevant information to the function which writes flaw files
if flaw_B_mixed:
    print_flaw_file(n_depth, bins_a, bins_b_sb, rpv['A'], 1, 'im'+filepost)
    print_flaw_file(n_depth, bins_b_em, False, rpv['A_weld'], 2, 'wm'+filepost)
else:
    print_flaw_file(n_depth, bins_a, bins_b, rpv['A'], 2, 'isb'+filepost)
```

3A. MATLAB flaw script: 'create_flaw_files_mixed.m'

This script must be used with the matlab functions: 'isb_flaw_mixed.m' and 'emb_weld_flaw_mixed.m'

Care should be taken when using this script to fully understand the way FAVOR employs "fractional" (non-whole number) flaws.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create_flaw_files_mixed.m
% written by MJS
% 2019-09-24
% updated 2019-10-18
%
% This version updates the prior version(v1) by allowing the user to input a
% vector of uncertainty values and write full sets of flaw files for each
% of the uncertainty values in the uncertainty vector.
%
% This version updates the prior version(v2) by retaining two additional
% significant figures in the crack depth density distributions and always
% rounds those values up, so that FAVOR will always include two flaws for
% each trial, in a manner consistent with the intended use per the SMiRT
% Challenge problem.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% In this script I will sample crack depth and crack length for both Flaw A
% and Flaw B. It is assumed that there are two analysis options:
%
% 1. Both cracks are surface-breaking cracks, based on classification of
% the nominal crack parameters. Bounds and distributions will be applied to
% each of the five crack parameters that characterize the two flaws.
%
% 2. The nominal crack parameters are contained within the distributions of
% the actual crack parameters. The sampling schemes will be yield cracks
% that are surface-breaking or embedded per ASME SC-XI, Section IWB-3600.
% All of the embedded cracks will be assumed to exist in the
% circumferential weld region.
%
% All of the required flaw files will be created to capture the features of
% both of the above analysis options.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Vessel Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rpv.ri = 86; % inches
rpv.clad = 0.25; % inches
rpv.wall = 8.5; % inches
rpv.height = 156; % inches

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Flaw Properties
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% For all cases, flaw parameters will be bounded. For example, the minimum
% crack depth is 0.25 inches (so that the crack penetrates beyond the inner
% cladding). There are practical limitations to the crack lengths and outer
% depths, so all distributions will be bounded.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% Inspector data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The observed flaws have the following nominal characteristics
d_a = 0.26; % Flaw A depth
L_a = 1.4; % Flaw A length
d_b = 0.40; % Flaw B outer depth
s_b = 0.05; % Flaw B inner depth
L_b = 2.2; % Flaw B length

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Sampling
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Let's just use a uniform distribution over the defined flaw parameters.
% These samples can always be transformed to our distribution of choice as
% needed using the appropriate inverse CDF. This is done

% Sample parameters
N = 1000000;

% Let's start the engine! Sample and start the machinery. The columns of
% random numbers are applied to calculate the following:
% 1. Flaw A crack depth
% 2. Flaw B crack depth
% 3. Flaw A crack length
% 4. Flaw B crack length
% 5. Flaw B inner depth

crack = rand(N,5);

% For now let's just assume we are bounded in a regime around the
% inspector's observations using a global parameter uncertainty.
% unc = [0.01 0.02 0.05 0.10 0.20 0.25 0.50 1.00];
unc = 1.00;

for j = 1:size(unc,2)

    adepth = [max([0.25 (1-unc(j))*d_a]) min([(1+unc(j))*d_a rpv.wall+rpv.clad])];
    bdepth = [max([0.25 (1-unc(j))*d_b]) min([(1+unc(j))*d_b rpv.wall+rpv.clad])];
    alength = [(1-unc(j))*L_a min([(1+unc(j))*L_a 2*pi*(rpv.ri+rpv.clad+rpv.wall)])];
    blength = [(1-unc(j))*L_b min([(1+unc(j))*L_b 2*pi*(rpv.ri+rpv.clad+rpv.wall)])];

    sdepth = [max([0 0.05-unc(j)*0.2]) min([0.05+unc(j)*0.2 0.25])];

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%% Flaw File Bins
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % There are two sets of bins required for the different flaw files. Both
    % types of flaw files use the same definition for normalized crack depth.
    % The aspect ratios for the surface-breaking flaw file are single numbers,
    % so the definitions for those bins are arbitrary, but can be made
    % conservative by rounding all values to the next highest value. The
    % embedded flaw bins are explicitly defined in FAVOR.

    ar_s = [2 6 10 Inf];
    ar_s_breaks = [2 6 10];

    ar_e = [1 1.25 1.5 2 3 4 5 6 8 10 15];
```

```
    ar_e_breaks = [1.25 1.5 2 3 4 5 6 8 10 15];

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%       Analysis
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    distr = 'normal'; % Choose uniform or normal distribution for crack parameters

    nstr = num2str(100*unc(j)); % Flaw file naming convention
    if length(nstr)<2
        nstr = strcat('0',nstr);
    end

    % Transform the uniform samples into crack parameters
    if strcmp(distr,'uniform')
        a_outer = adepth(1) + (adepth(2)-adepth(1))*crack(:,1);
        a_long = alength(1) + (alength(2)-alength(1))*crack(:,3);
        b_outer = bdepth(1) + (bdepth(2)-bdepth(1))*crack(:,2);
        b_long = blength(1) + (blength(2)-blength(1))*crack(:,4);
        b_inner = sdepth(1) + (sdepth(2)-sdepth(1))*crack(:,5);
        isbstr = ['isb',nstr,'_uniform.dat'];
        mixed_isbstr = ['im',nstr,'_uniform.dat'];
        mixed_weldstr = ['wm',nstr,'_uniform.dat'];
    end
    if strcmp(distr,'normal')
        pd = makedist('Normal');
        t_a = truncate(pd,(adepth(1)-d_a)*sqrt(12)/(adepth(2)-adepth(1)),Inf); % Truncate
normal distribution for Flaw A depth
        t_al = truncate(pd,(alength(1)-L_a)*sqrt(12)/(alength(2)-alength(1)),Inf); %
Truncate normal distribution for Flaw A depth
        t_bi = truncate(pd,(sdepth(1)-s_b)*sqrt(12)/(sdepth(2)-sdepth(1)),Inf); %
Truncate the normal distribution for Flaw B inner depth (must be positive)
        t_bo = truncate(pd,(bdepth(1)-d_b)*sqrt(12)/(bdepth(2)-bdepth(1)),Inf); %
Truncate the normal distribution for Flaw B outer depth (must be positive)
        t_bl = truncate(pd,(blength(1)-L_b)*sqrt(12)/(blength(2)-blength(1)),Inf); %
Truncate normal distribution for Flaw A depth
        a_outer = d_a + (adepth(2)-adepth(1))/sqrt(12)*icdf(t_a,crack(:,1));
        a_long = L_a + (alength(2)-alength(1))/sqrt(12)*icdf(t_al,crack(:,3));
        b_outer = d_b + (bdepth(2)-bdepth(1))/sqrt(12)*icdf(t_bo,crack(:,2));
        b_long = L_b + (blength(2)-blength(1))/sqrt(12)*icdf(t_bl,crack(:,4));
        b_inner = s_b + (sdepth(2)-sdepth(1))/sqrt(12)*icdf(t_bi,crack(:,5));
        isbstr = ['isb',nstr,'_normal.dat'];
        mixed_isbstr = ['im',nstr,'_normal.dat'];
        mixed_weldstr = ['wm',nstr,'_normal.dat'];
    end

    b_2d = abs(b_outer - b_inner);
    AR_a = a_long./a_outer; % These aspect ratio calculations are appropriate if the
cracks are surface-breaking
    AR_b = b_long./b_outer;
    AR_b_e = b_long./b_2d; % This aspect ratio is correct for embedded flaws
    dnorm_a = round(100*a_outer/(rpv.clad + rpv.wall))/100;
    dnorm_b = round(100*b_outer/(rpv.clad + rpv.wall))/100;

    n_depth = min([min(dnorm_a) min(dnorm_b)]):0.01:max([max(dnorm_a) max(dnorm_b)]);

    % Check if Flaw B is embedded or surface-breaking. The check is if
    % (b_outer-b_2d)>0.4*b_2d/2, then the crack is embedded. The simple check
```

```
    % can be reduced to checking the sampled values contained in the vector
    % crack(:,5). If the value is > 0.8333 (5/6) then the crack is considered
    % surface breaking, otherwise it is embedded. 'sb' is a logical vector that
    % identifies:
    %
    % 0 = embedded
    % 1 = surface-breaking

    sb = (b_outer-b_2d)<0.4*b_2d/2;

    % Fill the bins
    apdf = zeros(size(n_depth,2),size(ar_s,2)); % Flaw A - all surface-breaking
    bpdf = zeros(size(n_depth,2),size(ar_s,2)); % Flaw B - all surface-breaking
    bpdf_s = zeros(size(n_depth,2),size(ar_s,2)); % Flaw B - mix, surface-breaking
    bpdf_e = zeros(size(n_depth,2),size(ar_e,2)); % Flaw B - mix, embedded
    for i = 1:N
        rid_a = find(abs(n_depth - dnorm_a(i)) < 0.005);
        cid_a = length(find(ar_s_breaks < AR_a(i))) + 1;
        apdf(rid_a,cid_a) = apdf(rid_a,cid_a) + 1;

        rid_b = find(abs(n_depth - dnorm_b(i)) < 0.005);
        cid_b = length(find(ar_s_breaks < AR_b(i))) + 1;
        bpdf(rid_b,cid_b) = bpdf(rid_b,cid_b) + 1;
        if ~sb(i) % embedded
            cid_b = length(find(ar_e_breaks < AR_b_e(i))) + 1;
            bpdf_e(rid_b,cid_b) = bpdf_e(rid_b,cid_b) + 1;
        else % surface_breaking
            bpdf_s(rid_b,cid_b) = bpdf_s(rid_b,cid_b) + 1;
        end
    end

    % Normalize the depth density. This is how the depth density column will be
    % populated in the flaw file (this vector will be scaled by the flaw areal
    % density calculated for a given pressure vessel)

    dpdf = sum(apdf + bpdf,2)/N;
    dpdf_s = sum(apdf + bpdf_s,2)/N;
    dpdf_e = sum(bpdf_e,2)/N;

    % Normalize by samples at each depth. This is how the aspect ratio columns
    % will be populated in the flaw file

    ARpdf = (apdf+bpdf)./sum(apdf+bpdf,2);
    ARpdf_s = (apdf+bpdf_s)./sum(apdf+bpdf_s,2);
    ARpdf_e = bpdf_e./sum(bpdf_e,2);
    ARpdf(isnan(ARpdf)) = 0;
    ARpdf_s(isnan(ARpdf_s)) = 0;
    ARpdf_e(isnan(ARpdf_e)) = 0;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     % Create flaw file for first analysis option
%     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     isbnum = 2; % the expected number of flaws
%
%     isb_flaw_v2(n_depth,dpdf,ARpdf,rpv,isbstr,isbnum);
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Create flaw files for second analysis option
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%     embnum = ceil(sum(dpdf_e)); % the expected number of weld embedded flaws
%     isbnum = 2 - embnum; % the expected number of surface-breaking flaws

    isb_flaw_mixed(n_depth,dpdf_s,ARpdf_s,rpv);
    emb_weld_flaw_mixed(n_depth,dpdf_e,ARpdf_e,rpv,dpdf_s);

end
```

3B. MATLAB flaw function: 'isb_flaw_mixed.m'

This function must be used with the MATLAB script: 'create_flaw_files_mixed.m'

```
function [] = isb_flaw_mixed(d_norm,d_rho,ar_rho,g)
% Use this function to write the inner surface-breaking flaw file. The
% function is passed two vectors, one matrix, one structure and one string
% and writes a file compatible with the syntax required to run FAVOR is PFM
% mode.
%
% d_norm = a nx1 vector that contains the real values of all of the
% non-zero normalized depths of sampled cracks.
% d_rho = an nx1 vector that contains the flaw densities as a function of
% crack depth.
% ar_rho = an nxm matrix that contains the aspect ratio distributions at
% each depth.
% g = a structure containing the geometric parameters for the rpv under
% investigation.
% fname = the string that contains the name of the output file.
% fnum = the expected integer number of flaws

% FAVOR hard-coded parameters

Fidx = 161; %FAVOR version ID
nr = 1000; %number of records, required per FAVOR

% RPV surface area
a_rpv = 2*pi*g.ri*g.height/12^2;

% This is the flaw density over the entire vessel (beltline) volume
rho_f = 1/a_rpv;

% Here we perform a check to guarantee that we produce the expected number
% of flaws. (Actually, we'll scale to produce 0.1% above the expected number
% of whole number flaws. FAVOR will round down to the closest whole number)
d_rho_1 = 1.001/sum(d_rho)*d_rho;
d_rho_2 = 2.002/sum(d_rho)*d_rho;

% The distribution across the allowed depths is then calculated by scaling
% the density.
f_dis_1 = rho_f*d_rho_1;
f_dis_2 = rho_f*d_rho_2;

% The aspect ratio distribution for all allowed depths is calculated by
% scaling the distribution calculated above by 100.
ar_dis = ar_rho*100;

% Densities for depths with no flaws
rhozero = 0;
dzero = 0;

% Write dataset file
fdepth = round(100*d_norm);

numsb2 = round(1000*sum(d_rho)/2)

fid = fopen('im100_normal_test_A.dat','w'); % change data file name as needed
fprintf(fid,'%s\n',num2str(Fidx));
for j = 1:nr-numsb2
    for i = 1:100
```

```
        if ~isempty(find(i==fdepth,1))
            idx = find(i==fdepth,1);
            fprintf(fid,'%12u %15.8f %11.5f %12.5f %12.5f
%12.5f\n',i,ceil(1e8*f_dis_1(idx))/1e8,ar_dis(idx,1),ar_dis(idx,2),ar_dis(idx,3),ar_dis(i
dx,4));
        else
            fprintf(fid,'%12u %14.7f %12.5f %12.5f %12.5f
%12.5f\n',i,rhozero,dzero,dzero,dzero,dzero);
        end
    end
end
for j = nr-numsb2+1:nr
    for i = 1:100
        if ~isempty(find(i==fdepth,1))
            idx = find(i==fdepth,1);
            fprintf(fid,'%12u %15.8f %11.5f %12.5f %12.5f
%12.5f\n',i,ceil(1e8*f_dis_2(idx))/1e8,ar_dis(idx,1),ar_dis(idx,2),ar_dis(idx,3),ar_dis(i
dx,4));
        else
            fprintf(fid,'%12u %14.7f %12.5f %12.5f %12.5f
%12.5f\n',i,rhozero,dzero,dzero,dzero,dzero);
        end
    end
end
fclose(fid);

fid = fopen('im100_normal_test_B.dat','w'); % change data file name as needed
fprintf(fid,'%s\n',num2str(Fidx));
for j = 1:numsb2
    for i = 1:100
        if ~isempty(find(i==fdepth,1))
            idx = find(i==fdepth,1);
            fprintf(fid,'%12u %15.8f %11.5f %12.5f %12.5f
%12.5f\n',i,ceil(1e8*f_dis_2(idx))/1e8,ar_dis(idx,1),ar_dis(idx,2),ar_dis(idx,3),ar_dis(i
dx,4));
        else
            fprintf(fid,'%12u %14.7f %12.5f %12.5f %12.5f
%12.5f\n',i,rhozero,dzero,dzero,dzero,dzero);
        end
    end
end
for j = numsb2+1:nr
    for i = 1:100
        if ~isempty(find(i==fdepth,1))
            idx = find(i==fdepth,1);
            fprintf(fid,'%12u %15.8f %11.5f %12.5f %12.5f
%12.5f\n',i,ceil(1e8*f_dis_1(idx))/1e8,ar_dis(idx,1),ar_dis(idx,2),ar_dis(idx,3),ar_dis(i
dx,4));
        else
            fprintf(fid,'%12u %14.7f %12.5f %12.5f %12.5f
%12.5f\n',i,rhozero,dzero,dzero,dzero,dzero);
        end
    end
end
fclose(fid);

end
```

3C. MATLAB flaw function: ‘emb_weld_flaw_mixed.m’

This function must be used with the MATLAB script: ‘create_flaw_files_mixed.m’

```
function [] = emb_weld_flaw(d_norm,d_rho,ar_rho,g,sb_rho)
% Use this function to write the inner surface-breaking flaw file. The
% function is passed two vectors, one matrix, one structure and one string
% and writes a file compatible with the syntax required to run FAVOR is PFM
% mode.
%
% d_norm = a nx1 vector that contains the real values of all of the
% non-zero normalized depths of sampled cracks.
% d_rho = an nx1 vector that contains the flaw densities as a function of
% crack depth.
% ar_rho = an nxm matrix that contains the aspect ratio distributions at
% each depth.
% g = a structure containing the geometric parameters for the rpv under
% investigation.
% fname = the string that contains the name of the output file.
% fnum = the expected integer number of flaws.

% FAVOR hard-coded parameters

Fidx = 161; %FAVOR version ID
nr = 1000; %number of records, required per FAVOR

% Weld surface area, note that this weld surface area is specific to the
% SMiRT problem and only treats the 360 degree circumferential weld. Within
% the problem the observed embedded flaw was circumferential and therefore
% could only exist in the circumferential weld per FAVOR analysis.
a_weld = 2*3/8*pi*((g.ri + g.clad + g.wall)^2 - (g.ri + g.clad)^2)/12^2;

% This is the flaw density over the entire vessel (beltline) volume
rho_f = 1/a_weld;

% Here we perform a check to guarantee that we produce the expected number
% of flaws. (Actually, we'll scale to produce 0.1% above the expected number
% of whole number flaws. FAVOR will round down to the closest whole number)
if sum(d_rho) > 0
    d_rho = 1.001/sum(d_rho)*d_rho;
end

% The distribution across the allowed depths is then calculated by scaling
% the density.
f_dis = rho_f*d_rho;

% The aspect ratio distribution for all allowed depths is calculated by
% scaling the distribution calculated above by 100.
ar_dis = ar_rho*100;

% Densities for depths with no flaws
rhozero = 0;
dzero = 0;

% Write dataset file
fdepth = round(100*d_norm);

numsb2 = round(1000*sum(sb_rho)/2)

fid = fopen('wm100_normal_test_A.dat','w'); % change data file name as needed
```

```
fprintf(fid,'%s\n',num2str(Fidx));
for j = 1:nr-numsb2
    for i = 1:100
        if ~isempty(find(i==fdepth,1))
            idx = find(i==fdepth,1);
            fprintf(fid,'%6u %9.6f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f
%6.2f
%6.2f\n',i,ceil(1e6*f_dis(idx))/1e6,ar_dis(idx,1),ar_dis(idx,2),ar_dis(idx,3),ar_dis(idx,
4),...

ar_dis(idx,5),ar_dis(idx,6),ar_dis(idx,7),ar_dis(idx,8),ar_dis(idx,9),ar_dis(idx,10),ar_d
is(idx,11));
        else
            fprintf(fid,'%6u %9.6f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f
%6.2f
%6.2f\n',i,rhozero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero);
        end
    end
end
for j = nr-numsb2+1:nr
    for i = 1:100
        if ~isempty(find(i==fdepth,1))
            idx = find(i==fdepth,1);
            fprintf(fid,'%6u %9.6f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f
%6.2f
%6.2f\n',i,rhozero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero);
        else
            fprintf(fid,'%6u %9.6f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f
%6.2f
%6.2f\n',i,rhozero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero);
        end
    end
end
fclose(fid);

fid = fopen('wm100_normal_test_B.dat','w'); % change data file name as needed
fprintf(fid,'%s\n',num2str(Fidx));
for j = 1:numsb2
    for i = 1:100
        if ~isempty(find(i==fdepth,1))
            idx = find(i==fdepth,1);
            fprintf(fid,'%6u %9.6f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f
%6.2f
%6.2f\n',i,rhozero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero);
        else
            fprintf(fid,'%6u %9.6f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f
%6.2f
%6.2f\n',i,rhozero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero);
        end
    end
end
for j = numsb2+1:nr
    for i = 1:100
        if ~isempty(find(i==fdepth,1))
            idx = find(i==fdepth,1);
            fprintf(fid,'%6u %9.6f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f
%6.2f
%6.2f\n',i,ceil(1e6*f_dis(idx))/1e6,ar_dis(idx,1),ar_dis(idx,2),ar_dis(idx,3),ar_dis(idx,
4),...
```

```
ar_dis(idx,5),ar_dis(idx,6),ar_dis(idx,7),ar_dis(idx,8),ar_dis(idx,9),ar_dis(idx,10),ar_d
is(idx,11));
        else
            fprintf(fid,'%6u %9.6f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f
%6.2f
%6.2f\n',i,rhozero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero,dzero);
        end
    end
end
fclose(fid);
end
```