Reuven Reyman

MAE 190 – Winter 2024

Shaft Design Project Report

View Program: https://reuvenrey.github.io/me/projects/shaft-design-project.html

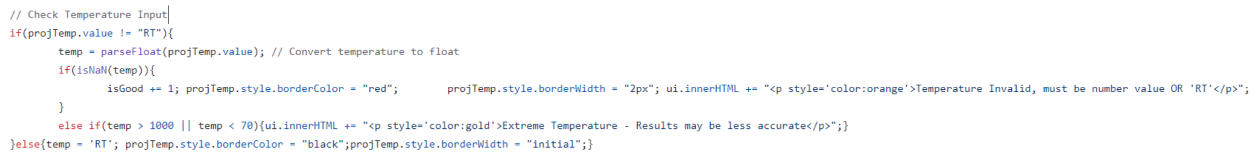
Introduction. The goal of the shaft design project is to create a computer software which, given necessary inputs by the user, can determine the minimum diameter of a metal shaft required to perform without failure. To create such a program, I needed to find an adaptable method for determining failure criteria as well as a proper coding language which made my program easily accessible and versatile. In my software, which was programmed in JavaScript and hosted on an HTML website, the main method for forming an initial guess for shaft diameter is the DE-Goodman method. Additionally, for usability, my software has an innovative "console" which displays various information about the results of the code as it is calculating a diameter. Once the diameter has been determined, the user can run through the console and see how this number was achieved and various warnings or notes about the result.

To use this method, a few values are required by the user, others are calculated. My program requires the following inputs from the user:

- Desired Factor of Safety
- Reliability (ke)
- Material Choice
- Surface Finish
- D/d Ratio
- Fillet Radius
- Temperature

These inputs are used to make appropriate guesses for other required values such as the endurance limit, Se, Kf and Kfs, and others.

Initial Guess. Before beginning calculations, the software first verifies the user's inputs to ensure that the final result will make sense as some extreme values can skew the calculations. This process is completed in a single function which records the value of each text input and dropdown and compares these values with min and max values for all the equations used. For example, the equation used to calculate temperature is less accurate with temperature values outside the range of 70°C to 1000°C. If the user inputs a value outside this range, the code will display a warning.



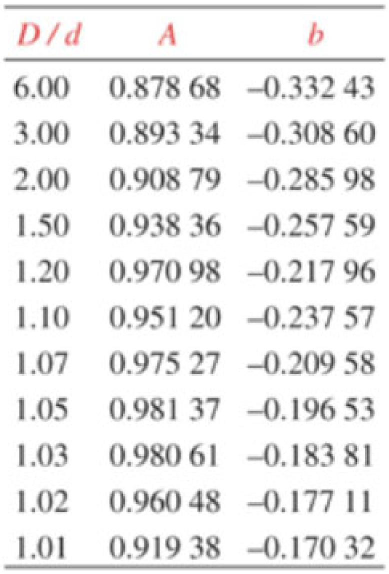
```
// Check Temperature Input
if(projTemp.value != "RT"){
        temp = parseFloat(projTemp.value); // Convert temperature to float
        if(isNaN(temp)){
                isGood += 1; projTemp.style.borderColor = "red";        projTemp.style.borderWidth = "2px"; ui.innerHTML += "<p style='color:orange'>Temperature Invalid, must be number value OR 'RT'</p>";
        }
        else if(temp > 1000 || temp < 70){ui.innerHTML += "<p style='color:gold'>Extreme Temperature - Results may be less accurate</p>";}
}else{temp = 'RT'; projTemp.style.borderColor = "black";projTemp.style.borderWidth = "initial";}
```

Warning Example:

Extreme Temperature - Results may be less accurate

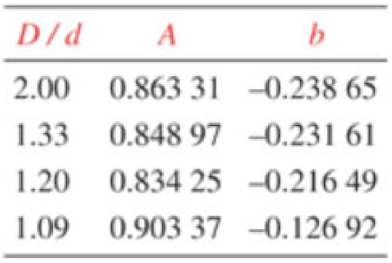
Once the inputs are verified, the software will form an initial guess for the small diameter d. Before we can make that guess, the program needs to first find Kf, Kfs, and Se. Because Kf and Kfs values require d to be known, initial guesses for diameter are made under the assumption that these values are equal to Kt and Kts respectively, both of which are determined from a preset table:

Kt Table



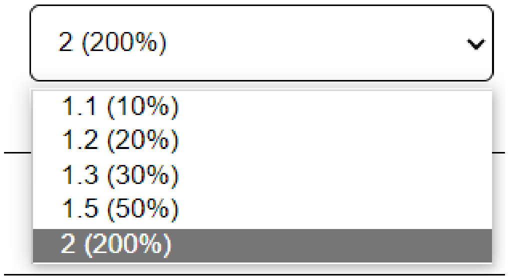
D/d	A	b
6.00	0.878 68	–0.332 43
3.00	0.893 34	–0.308 60
2.00	0.908 79	–0.285 98
1.50	0.938 36	–0.257 59
1.20	0.970 98	–0.217 96
1.10	0.951 20	–0.237 57
1.07	0.975 27	–0.209 58
1.05	0.981 37	–0.196 53
1.03	0.980 61	–0.183 81
1.02	0.960 48	–0.177 11
1.01	0.919 38	–0.170 32

Kts Table



D/d	A	b
2.00	0.863 31	–0.238 65
1.33	0.848 97	–0.231 61
1.20	0.834 25	–0.216 49
1.09	0.903 37	–0.126 92

The value of D/d is determined by the user by selecting one of four options from a drop-down:



Because the two tables for Kt and Kts had different D/d options, I used linear-interpolation to form two JavaScript arrays with the same D/d and resulting A and b values:

```
const KtGuessTable = ["1.1,0.95120,-0.23757", "1.2,0.97098,-0.21796", "1.3,0.96011,-0.23117", "1.5,0.93836,-0.25759", "2,0.90879,-0.28598"];
const KtsGuessTable = ["1.1,0.89709,-0.135063", "1.2,0.83425,-0.21649", "1.3,0.84557,-0.228121", "1.5,0.85261,-0.23340", "2,0.86331,-0.23865"];
```

Each array index consists of a single string containing three variables separated by commas. The string is arranged *D/d value, A value, b value*. Using the user-inputted D/d value, the code takes each index in the array, separates out the D/d value, tests it against the user's D/d value, and either moves on or (if D/d matches) collects the A and b values. Next, these values are plugged into the equation:

Theoretical

$$K_t = A\left(\frac{r}{d}\right)^b$$

Code

```
 var Aval = parseFloat(tempVal[1]);
 var Bval = parseFloat(tempVal[2]);
Kt = Aval*(Math.pow(filletRadius,Bval));
```

Where r/d is the user-provided fillet radius.

To determine Se, the program calculates various modifying factors using the user-provided information.

For ka:

```
// Get Surface Factor ka
        switch(surfaceFinish){
                case "ground":
                        if(units == "english"){
                                var Avar = 1.21;
                                var Bvar = -0.067;
                        }else{
                                var Avar = 1.38;
                                var Bvar = -0.067;
                        }
                        break;
                case "machined":
                        if(units == "english"){
                                var Avar = 2;
                                var Bvar = -0.217;
                        }else{
                                var Avar = 3.04;
                                var Bvar = -0.217;
                        }
                        break;
                case "as-forged":
                        if(units == "english"){
                                var Avar = 12.7;
                                var Bvar = -0.758;
                        }else{
                                var Avar = 54.9;
                                var Bvar = -0.758;
                        }
                        break;
        }// end of switch

        console.log("For Ka: Avar = " + Avar + ", Bvar = " + Bvar);

        ka = Avar*(Math.pow(Su,Bvar));
```

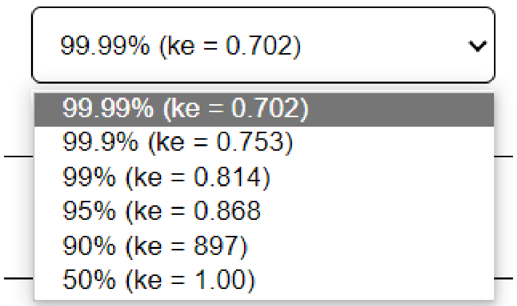
For kb, because this value requires the diameter to be known, it is assumed to be equal to 0.9 for initial guess and will be updated later.

For factor kc, I have set it equal to 1 since Torsion and Bending are combined.

For kd, the temperature factor, I have used an equation in the textbook:

```
// For Tempurature Factor
        if(temp == "RT"){
                kd = 1;
        }else{
                if(units == "english"){
                        kd = 0.98 + (3.5*Math.pow(10,-4))*temp - (6.3*Math.pow(10,-7))*(Math.pow(temp,2));
                }else{
                        kd = 0.99 + (5.9*Math.pow(10,-4))*temp - (2.1*Math.pow(10,-6))*(Math.pow(temp,2));
                }
        }
```

And for the final factor, ke, the reliability is chosen by the user from a drop-down menu:



99.99% (ke = 0.702)

99.99% (ke = 0.702)
99.9% (ke = 0.753)
99% (ke = 0.814)
95% (ke = 0.868
90% (ke = 897)
50% (ke = 1.00)

Together, these values are combined with Se', which is determined using the following code to form an initial estimate of Se:

```
// (1) Get Se' value
        if(units == "english"){ // English Units
                ui.innerHTML += "<p>Using English Units (KPSI)</p>";
                Sy = materialSY;
                Su = materialSUT;

                // Page 326 in text book
                if(Su <= 200){
                        SePrime = 0.5*Su;
                }else{
                        SePrime = 100;
                }
```

Once Kf, Kfs, and Se have been determined, the code moves on to the next step which is forming an initial guess for d. This is done using DE-Goodman criteria and the theory is as follows:

$$A = \sqrt{4\left(K_f M_a\right)^2 + 3\left(K_{fs} T_a\right)^2} \;\;,\;\; B = \sqrt{4\left(K_f M_m\right)^2 + 3\left(K_{fs} T_m\right)^2}$$

Where Kf, Kfs = stress concentration factors, Ma,Ta = alternating loads, Mm, Tm = mean loads.

$$d = \left[\frac{16n}{\pi}\left(\frac{A}{S_e} + \frac{B}{S_{ut}}\right)\right]^{\frac{1}{3}}$$

Where d = guessed diameter, n = desired factor of safety, Se = endurance limit, Sut = ultimate tensile strength.

This theory applies similarly to JavaScript:

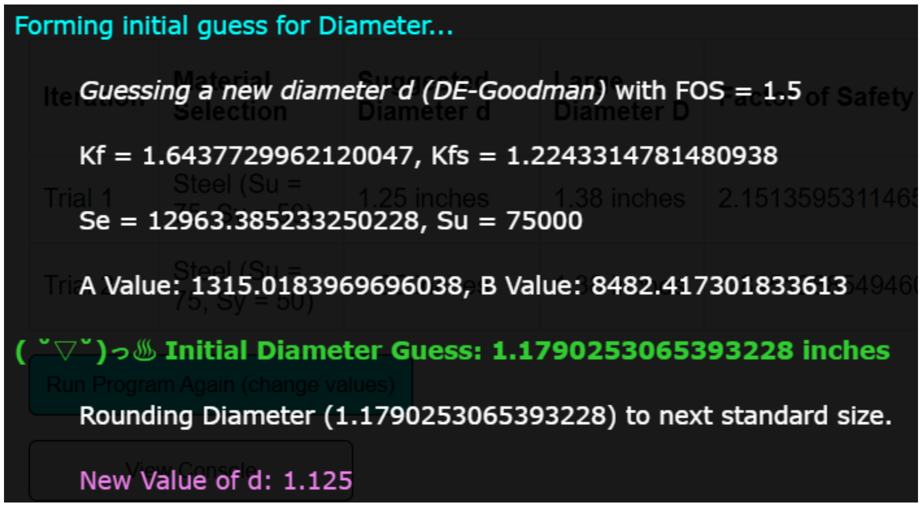
```
    var $tempVar = 4*(Math.pow((fKf*Ma),2)) + 3*(Math.pow((fKfs*Ta),2));
    let Aval = parseFloat(Math.sqrt($tempVar));

    $tempVar = 4*(Math.pow((fKf*Mm),2)) + 3*(Math.pow((fKfs*Tm),2));
    let Bval = parseFloat(Math.sqrt($tempVar));

     ui.innerHTML += "<p style='margin-left:40px;'>A Value: " + Aval + ", B Value: " + Bval + "</p>";

  $tempVar = ((16*Nfos) / Math.PI)*((Aval / fSe) + (Bval / fSu));
          console.log("Inner DE-Goodman Calculated = " + $tempVar);
  d_new = Math.pow($tempVar, (1/3));
```

Once the initial guess has been made, the code will next convert the guessed value to the nearest standard diameter size:



```
Forming initial guess for Diameter...

    Guessing a new diameter d (DE-Goodman) with FOS = 1.5

    Kf = 1.6437729962120047, Kfs = 1.2243314781480938

    Se = 12963.385233250228, Su = 75000

    A Value: 1315.0183969696038, B Value: 8482.417301833613

( ˘▽˘)っ♨ Initial Diameter Guess: 1.1790253065393228 inches

    Rounding Diameter (1.1790253065393228) to next standard size.

    New Value of d: 1.125
```

The process for this is rather simple: I have hard coded a table of various purchasable shaft sizes ranging from 0.25” to 8” determined from a large manufacturer of steel shafts. Once a guess is made for d, the code will run a function which runs a while loop. The code starts at the beginning of the table and checks the value before and the value after each iteration (starting at the second table value). If the before value is smaller than the guess diameter, and the after value is larger than the guess diameter, then the function outputs the current table value. This value is the closest standard shaft size to the initial guess. To prevent errors, if the guessed diameter is larger or smaller than the largest or smallest respective values on the hard-coded table, the code will automatically choose the nearest possible value as its output.

Iterative Guessing. Now that the initial guess has been made, the software can begin to calculate various other attempts until the desired factor of safety is achieved. First, new values of Kf and Kfs are calculated using the following textbook formulas converted into JavaScript math:

$$K_f = 1 + \frac{K_t - 1}{1 + \sqrt{a}/\sqrt{r}}$$

Bending or axial:

$$\sqrt{a} = 0.246 - 3.08(10^{-3})S_{ut} + 1.51(10^{-5})S_{ut}^2 - 2.67(10^{-8})S_{ut}^3 \quad 50 \le S_{ut} \le 250 \text{ kpsi}$$

Torsion:

$$\sqrt{a} = 0.190 - 2.51(10^{-3})S_{ut} + 1.35(10^{-5})S_{ut}^2 - 2.67(10^{-8})S_{ut}^3 \quad 50 \le S_{ut} \le 220 \text{ kpsi}$$

Then the value of kb is recalculated and a new value of Se is formed:

$$\text{Kb} = \begin{cases} (d/0.3)^{-0.107} = 0.879d^{-0.107} & 0.3 \le d \le 2 \text{ in} \\ 0.91d^{-0.157} & 2 < d \le 10 \text{ in} \end{cases}$$

Using the new Kf and Kfs, the program finds the Von Mises Alternating and Mean stresses:

```
var fvonStress = 0;

var fSa = (32*fKf*fMoment)/(Math.PI * Math.pow(dval,3));
var fTa = (16*fKfs*fTorque)/(Math.PI * Math.pow(dval,3));

var tempVar = (Math.pow(fSa,2)) + 3*(Math.pow(fTa,2));

fvonStress = Math.pow(tempVar, 0.5);
```

Once the stresses and Se values have been calculated using the guessed diameter, the program tests for two factor of safeties: the yielding factor of safety and DE-Goodman factor of safety:

Yielding

$$n_y = \frac{S_y}{\sigma'_{\max}} > \frac{S_y}{\sigma'_a + \sigma'_m}$$

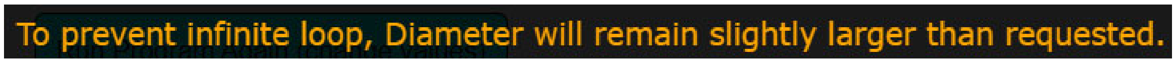
DE-Goodman Criteria

$$n = \frac{\pi d^3}{16}\left(\frac{A}{S_e} + \frac{B}{S_{ut}}\right)^{-1}$$

See page 5 for A and B values

If either the yielding or DE-Goodman factor of safeties are smaller than the desired factor of safety set by the user, the program increases the diameter to the next larger standard size and performs all the previous calculations starting with Kf and Kfs again. If the DE-goodman factor of safety is larger than 5% of the desired factor of safety, then the program decreases the diameter to the next smaller standard size and performs the calculations again.

Problems With Iterative Guessing. One of the problems with this method of iterative guessing is the possibility of the program becoming stuck in an infinite loop. In some instances, if the current diameter returns a factor of safety which is larger than 5% of the desired factor of safety, but decreasing the diameter to the next standard size causes the factor of safety to be less than the desired factor of safety, this can cause the program to keep looping between these two values. To prevent this from happening, I have hardcoded a safety feature which tests if the previous diameter was larger or smaller than the current diameter before making a new guess. This way, if the diameter is decreased, but then forced to increase again to achieve a net-positive factor of safety, the program will stop itself and warn the user that the diameter will be larger than requested (see below):

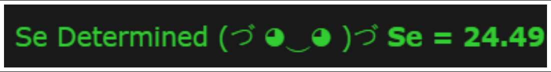
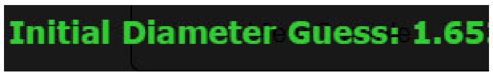
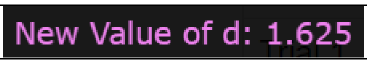
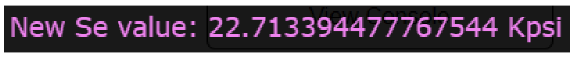


```
To prevent infinite loop, Diameter will remain slightly larger than requested.
```

Additionally, the program will stop the loop should the diameter be smaller than or less than the smallest or largest available standard size respectively.

Final Results. Once the final diameter has been achieved, either with a factor of safety no larger than 5% of the desired factor of safety, or with the minimum possible size to achieve a factor of safety larger than the desired value, the diameter will be displayed in a simple table along with some other useful values. For ease-of-use, I have programmed an easy way for the user to run the program again with different values, which will be displayed in the same table with all previous iterations of the program.

One of the main methods I used to test the correctness of my program was by comparing its results with that of *Example 7-2* in the textbook (11th edition, page 387 – 389). To test, I plugged in the values used in this example problem, then used the console I programmed to track the program and see if its values lined up with the example. Some key results I used are below:

Example Problem	My Code
$S_e = (0.801)(0.9)(0.5)(68) = 24.5$ kpsi	 `Se Determined (づ ◕‿◕ )づ Se = 24.49`
$d = \left\{ \frac{16(1.5)}{\pi} \left(\frac{2(1.7)(3651)}{24\,500} + \frac{\{3[(1.5)(3240)]^2\}^{1/2}}{68\,000} \right) \right\}^{1/3}$ $d = 1.69$ in	 `Initial Diameter Guess: 1.65`
$d = 1.625$ in.	 `New Value of d: 1.625`
$S_e = (0.801)(0.835)(0.5)(68) = 22.7$ kpsi	 `New Se value: 22.713394477767544 Kpsi`

$$\frac{1}{n_f} = \frac{\sigma'_a}{S_e} + \frac{\sigma'_m}{S_{ut}} = \frac{12\,910}{22\,700} + \frac{8659}{68\,000} = 0.696$$

$$n_f = 1.44$$

Factor of Safety

1.437155108462283

As you can see, the results of my code match very closely with that of the example problem.

I also ran the same loading and geometry as the example problem, but with different material choices and received seemingly accurate results:

Final Results

Iteration	Material Selection	Suggested Diameter d	Large Diameter D	Factor of Safety	Desired Factor of Safety
Trial 1	1020 CD Steel (Su = 68, Sy = 57)	1.75 inches	2.100 inches	1.780	1.5
Trial 2	1020 CD Steel (Su = 68, Sy = 57)	2 inches	2.400 inches	1.941	1.5
Trial 3	1095 HR Steel (Su = 120, Sy = 66)	1.625 inches	1.950 inches	1.637	1.5

As another way to test my code, I ran a trial in which I kept all values the same except for the factor of safety which I decreased incrementally:

Final Results

Iteration	Material Selection	Suggested Diameter d	Large Diameter D	Factor of Safety	Desired Factor of Safety
Trial 1	Steel (Su = 75, Sy = 50)	3.25 inches	3.90 inches	12.542047472876199	10
Trial 2	Steel (Su = 75, Sy = 50)	3 inches	3.60 inches	9.978107699545484	8
Trial 3	Steel (Su = 75, Sy = 50)	2.75 inches	3.30 inches	7.782065084168864	6
Trial 4	Steel (Su = 75, Sy = 50)	2.25 inches	2.70 inches	4.387173304265018	4
Trial 5	Steel (Su = 75, Sy = 50)	1.75 inches	2.10 inches	2.1301104345332185	2
Trial 6	Steel (Su = 75, Sy = 50)	1.375 inches	1.65 inches	1.060815881133294	1

As expected, the suggested diameter decreases as the factor of safety decreases which is resembling of real life scenario.

Concluding Remarks. Currently, my program is relatively simple, however effective for the specific use case outlined in the project description. I have also added quality-of-life features such as the ability to view the console during and after the code has run which provides an extensive look into the process and variables used in determining a satisfying diameter value. Additionally, the code saves all iterations performed into a simple table so that the user may make adjustments and view how these affect the outcome and compare each outcome with the others.

Despite its innovative appearance and usability, my code also suffers some limitations. Due to the time constraint with this project, many of the user-inputted values, for sake of time-saving, must be chosen from a drop-down menu which greatly limits the possible inputs. Additionally, the program only works in English units, though this is something I could easily update should I have more time.

If I was to add more to my program, I would definitely add an option to work in Metric units. I would also add the feature to input custom Sy and Sut values as well as more flexibility with the other parameters. Additionally, my code does not currently convert the large D value to standard shaft dimensions, which is something I would need to add in another update.

As for now, I have a robust and user-friendly program which properly demonstrates some of the material we have learned in MAE 190.

View Program (online site): https://reuvenrey.github.io/me/projects/shaft-design-project.html