SANDIA REPORT

SAND2022-8871
Printed October 2021

Sandia National Laboratories

Mass Property Calculator

E. Corona and C.J. Fietek

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

A mass property calculator has been developed to compute the moment of inertia properties of an assemblage of parts that make up a system. The calculator can take input from spreadsheets or Creo mass property files or it can be interfaced with Phoenix Integration Model Center. The input must include the centroidal moments of inertia of each part with respect to its local coordinates, the location of the centroid of each part in the system coordinates and the Euler angles needed to rotate from the part coordinates to the system coordinates. The output includes the system total mass, centroid and mass moment of inertia properties. The input/output capabilities allow the calculator to interface with external optimizers. In addition to describing the calculator, this document serves as its user's manual. The up-to-date version of the calculator can be found in the Git repository https://cee-gitlab.sandia.gov/cjfiete/mass-properties-calculator.

ACKNOWLEDGMENT

The input by Josiah Bigelow and Jeff Gruda during the development of this work is acknowledged with thanks. Also, thanks go to Bill Scherzinger for his review of selected sections of this report.

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.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

$\boldsymbol{D}_i^Y, \boldsymbol{D}_i^P, \boldsymbol{D}_i^R$	Transformation matrices for yaw, pitch and roll
$\boldsymbol{D}_i$	Transformation matrix from $\boldsymbol{r}'$ to $\boldsymbol{r}$ $(= \boldsymbol{D}_R \boldsymbol{D}_P \boldsymbol{D}_Y)$
$\boldsymbol{I}_i$	Inertia matrix about system centroid in system coordinates
$\boldsymbol{I}_i'$	Inertia matrix about part centroid in system coordinates
$\boldsymbol{I}_i^p$	Inertia matrix about part centroid in part coordinates
$\boldsymbol{I}^s$	Centroidal system inertia matrix
$I_{x'x'}, I_{y'y'}, I_{z'z'}, I_{x'y'}, I_{y'z'}, I_{z'x'}$	Part centroidal moment of inertia components
$I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{yz}, I_{zx}$	Part moment of inertia components about system centroid
I_1, I_2, I_3	System principal moments of inertia. $I_1 \geq I_2 \geq I_3$
M	System mass
$\boldsymbol{R}_i$	Relative position of part centroid with respect to system centroid
$\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$	Unit vectors along $\{x, y, z\}$
m_i	Part mass
$\{n_{11}, n_{21}, n_{31}\}$	Principal directions of I_1
$\{n_{12}, n_{22}, n_{32}\}$	Principal directions of I_2
$\{n_{13}, n_{23}, n_{33}\}$	Principal directions of I_3
$\boldsymbol{r} = \{x, y, z\}$	System coordinates
$\boldsymbol{r}' = \{x', y', z'\}$	Part local coordinates
$\boldsymbol{r}_i^c = \{x_i^c, y_i^c, z_i^c\}$	Part centroid coordinates in system coordinates
$\theta_i^Y, \theta_i^P, \theta_i^R$	Part yaw, pitch and roll Euler angles

1. INTRODUCTION

This document presents a description of the formulation, use and validation of a Matlab mass property calculator script that was developed in response to a customer request. The input to the calculator is a collection of parts with known mass properties. These include the mass and the centroidal inertia tensor with respect to local, or part, coordinates as well as the location of the part center of gravity (cg) in global, or system, coordinates and the Euler angles that specify the orientation of the part. The output is the total mass and centroidal inertia tensor for the assembly with respect to the global coordinate system. Other derived outputs are also calculated. These include the principal inertia tensor components, the corresponding principal directions and the principal angle of inclination (PAI), defined as the angle between the principal direction corresponding to the largest principal value and the system x-axis.

The intended use of the calculator is in design studies and trim/balance calculations of a total assembly. As such, input and output operations and the interfaces with external optimization packages are important. The part input to the calculator can be provided from either a spreadsheet csv file, whose format is described in the next section. Another option is to read part properties from a Creo mass properties text file. The output can be written as a spreadsheet file that contains the final results and can be used to interface with external optimization programs.

Another feature of the calculator that was implemented is an interface with Phoenix Integration Model Center. This allows input directly from SolidWorks and Creo files as well as the optimization of the output quantities with respect to the parameters of the parts.

Use of the calculator requires writing a 'driver' Matlab script to integrate it with Model Center. Templates and examples will be presented in subsequent sections.

The document is organized as follows: Section 2 introduces important conventions used in the calculator that must be understood to properly interpret the results. Section 3 presents the organization of the calculator and brief descriptions of the functions used to conduct the calculation. Here the input/output operations are strictly through spreadsheet files. Section 4 describes the capability to read Creo mass properties text files. The next section presents an example of the use of the calculator for optimization calculations with Matlab. Section 6 presents the procedure to integrate the calculator with Model Center. The report concludes with the presentation of a number of validation problems in Section 7. Finally, the listings of the core and auxiliary function scripts of the calculator at the time of publication are shown in Appendix A. The most up-to-date version can be found in the Git repository https://cee-gitlab.sandia.gov/cjfiete/mass-properties-calculator.

2. CONVENTIONS

The formulation of the calculator is based on a set of conventions as listed below:

- The coordinate systems are all Cartesian and right-handed.
- 'Part' coordinates $\{x', y', z'\}$ are local to each of the parts in the system
- 'System' coordinates $\{x, y, z\}$ are global to the system.
- The moment of inertia matrix with respect to rectangular coordinates$\{p, q, r\}$ is taken as

$$\boldsymbol{I} = \begin{bmatrix} I_{pp} & I_{pq} & I_{pr} \\ I_{pq} & I_{qq} & I_{qr} \\ I_{pr} & I_{qr} & I_{rr} \end{bmatrix} \tag{2.1}$$

  where

$$\begin{aligned} I_{pp} &= \int (q^2 + r^2)\, dm \quad I_{pq} = -\int pq\, dm \\ I_{qq} &= \int (p^2 + r^2)\, dm \quad I_{pr} = -\int pr\, dm \\ I_{rr} &= \int (p^2 + q^2)\, dm \quad I_{pr} = -\int pr\, dm. \end{aligned} \tag{2.2}$$

  Here the $\{p, q, r\}$ can represent either $\{x', y', z'\}$ or $\{x, y, z\}$.
- The principal inertias $\{I_1, I_2, I_3\}$ are ordered such that $I_1 \geq I_2 \geq I_3$. The principal directions are, respectively $\{n_{11}, n_{21}, n_{31}\}$, $\{n_{12}, n_{22}, n_{32}\}$ and $\{n_{13}, n_{23}, n_{33}\}$.
- The principal angle of inclination is defined to be the angle between the principal direction $\{n_{11}, n_{21}, n_{31}\}$ and the system x axis. $\theta_p = \cos^{-1}(n_{11})$.
- Parts are brought in with their local coordinate system coinciding with the system coordinates by prescribing the locations of the part centroids in the system coordinate system.
- The parts can be re-oriented using yaw, pitch and roll Euler angles (zyx) as defined in Fig. 3-3 to re-orient the system coordinates with respect to the part coordinates.
- The output is given with respect to 'system' coordinates defined by the user.

3. CORE FUNCTIONALITY AND ORGANIZATION

Figure 3-1 shows the basic flowchart of the calculator. It is sequential in nature, and each of the boxes represents an operation for a specific task. A 'driver' function is needed to call all calculator functions. Internally, the calculator operates on two structure arrays called `part` and `system` in the script below. The first contains the list of part properties while the second contains the system properties. In order to maintain ease of use, the arrays are extended as needed in each function, and 'old' information is never deleted. Although this scripting choice might make the calculator slightly less efficient, it provides a relatively simple, linear flow that users can more easily follow and that allows for interrogation at the end of the calculations of the results at any given step.

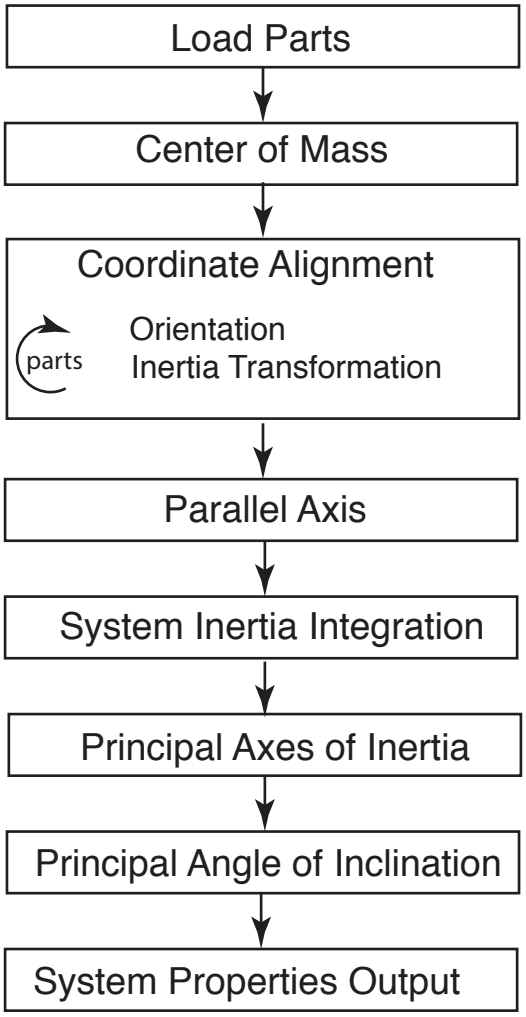

Figure 3-1. Core calculator flowchart.

To use the calculator, a basic driver program needs to be written by the user using the example below as a guide.

```
clc
clear
close all

% Driver program for mass calculator

name = '/home/ecorona/MassProperties2021/NewCalculator/Ver_2bar/Two_Bar_Problem.xlsx';
% Name (and path) of the part spreadsheet that will be loaded.

part = loading_parts(name);
% Loads the spreadsheet and puts it into a structured array containing all the parts.

[system(1)] = deal(cg_calc(part));
% Locates the cg of the system, and total mass of the system.

[part] = coordinate_alignment(part);
% Calculates inertia matrix with respect to axes directions of system using part orientation

[part] = deal(parallel_axis(part,system));
%Transforms the MOI's of each part to the system cg location.

[system] = deal(inertia_integration(part,system));
%This sums the inertia for the entire system.

[system] = principals(system);
%This calculates the principal MOI for the system, and the principal directions.

[system] = pai(system);
%Calculates the largest angle from principal axis to system axis

spread_gen('System_Two_Bar_Problem.csv',system)
```

Note that the file name containing the input is defined first. Each of the subsequent statements are calls to functions, which will be described next. The name of the functions appears between square brackets in the section headers.

3.1. Load Parts [`loading_parts`]

This function loads in a spreadsheet (Excel or csv) containing the information about the parts used. The spreadsheet data must be in the format shown in the example in Table 3.1 with one part per row and the exact headers. Here, the "Name" column contains a string with the name of the part and is followed by the mass of the part. The calculator uses two Cartesian coordinate systems: $\{x, y, z\}$ corresponds to the system coordinates while $\{x', y', z'\}$ corresponds to the part coordinates. The location of the part centroid is to be specified in the system coordinates. The next 6 entries are for the components of the inertia tensor with respect to the part coordinates. This is followed by the three Euler angles, in degrees, to orient the part in the system coordinates. These will be described in the next section.

Figure 3-2 shows an example of an input spreadsheet. It is important to note that the column headers that appear in the first row must be named exactly as shown in order for the calculator to work properly in all cases to be discussed.

Table 3-1. Format of Part Spreadsheet

Name	Mass	Centroid			Moment of Inertia						Orientation		
	m_i	x_i^c	y_i^c	z_i^c	$I^p_{x'x'i}$	$I^p_{y'y'i}$	$I^p_{z'z'i}$	$I^p_{x'y'i}$	$I^p_{y'z'i}$	$I^p_{x'z'i}$	θ_i^Y	θ_i^P	θ_i^R
part i	1.0	0	0	0	10	10	10	0	0	0	0	0	0

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	C
1	name	mass	centroidX	centroidY	centroidZ	Ixx	Iyy	Izz	Ixy	Iyz	Ixz	T1	T2	T3	
2	Cylindrical Shell	18.4569	10	0	0	637.338	933.898	933.898	0	0	0	0	0	0	
3	Cap_1	2.59672	0.125	0	0	42.9271	21.4771	21.4771	0	0	0	0	0	0	
4	Cap_2	2.59672	19.875	0	0	42.9271	21.4771	21.4771	0	0	0	0	0	0	
5	Rod_1	0.3927	2.75	1	1	0.04909	0.84267	0.84267	0	0	0	0	0	0	
6	Rod_2	0.3927	7.165111	1	-0.606969	0.04909	0.84267	0.84267	0	0	0	0	-40	0	
7	Sphere	1	10	0	0	0.7145	0.7145	0.7145	0	0	0	0	0	0	
8															

Figure 3-2. Input spreadsheet.

The input to the function is simply the name of the spreadsheet file that contains the properties of all the parts, as can be seen in the driver script. The spreadsheet data are then separated and restructured into a structure array (`part`) using a `for` loop to process all of the individual parts. This structured array then contains all of the different values for each part natively as string, scalar, 1-D array or 2-D array. The `part` array is then passed back to the calling function with the following components for the *i*th part:

`part(i).name`	String	Name of the part
`part(i).mass`	Scalar	Value of the mass
`part(i).centroid`	1-D array (3)	Centroid location in system coordinates
`part(i).inertia`	2-D array (3,3)	Moment of inertia in part coordinates
`part(i).orientation`	1-D array (3)	Euler Angles

3.2. Center of Mass [`cg_calc`]

This function takes the `part` structured array and creates a new structured array `system` containing the mass and centroid for the system. It takes the mass and centroid location for each individual part, and calculates the system total mass as

$$M = \sum_{i=1}^{n} m_i, \tag{3.1}$$

and center of mass from

$$\boldsymbol{d} = \frac{\sum_{i=1}^{n} m_i \boldsymbol{r}_i}{M} \tag{3.2}$$

Where $\boldsymbol{r}_i$ is the coordinate location of the *i*th part, and $\boldsymbol{d}$ is the coordinate location of the system center of mass. To accomplish this summation, a `for` loop and repeated sum program structure is used. The `system` array is passed back to the calling function with the following components:

`system(1).mass`	Scalar	Total mass of the system
`system(1).centroid`	1-D array (3)	Location of system centroid

3.3. Coordinate Alignment [`coordinate_alignment`]

Once the total mass and the center of gravity of the system have been calculated, the next step is to calculate the inertia tensor of each part in the system coordinates. This is accomplished using a loop (indicated by the circular arrow in Fig.3-1) that runs over the number of parts and contains calls to two functions: the first, `orientation`, calculates the transformation matrix to go from the individual part coordinates to the system coordinates while the second, `inertia-transformation` calculates the transformed inertia components from the part coordinates to the system coordinates. These will be described in more detail next.

This function adds the coordinate transformation matrix and the moment of inertia components to the part structure array as the variables

`part(i).transformation_matrix`	2-D array (3,3)	Transformation matrix
`part(i).transformed_part_inertia`	2-D array (3,3)	Moment of inertia in system coordinates

3.3.1. Orientation [`orientation`]

This function uses the Euler angles θ_i^Y, θ_i^P and θ_i^R to generate the transformation matrix from the local part coordinates $\{x', y', z'\}$ to the system coordinates $\{x, y, z\}$. Figures 3-3(a), (b) and (c) show the orientation sequence from the part coordinates, shown in black dashed lines, to the system coordinates in Fig 3-3(d). Starting with the part and system axes coinciding, as shown in Fig. 3-3(a), the first rotation is about the z axis (yaw), to the configuration shown in Fig. 3-3(b). The next rotation is about the new y (pitch), to the configuration shown in Fig. 3-3(c). The final rotation is about the new x (roll), to the final configuration in Fig. 3-3(d). The rotation sequence is zyx.

The transformation matrices for each of the rotations are

$$\boldsymbol{D}_i^Y = \begin{bmatrix} \cos\theta_i^Y & \sin\theta_i^Y & 0 \\ -\sin\theta_i^Y & \cos\theta_i^Y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$\boldsymbol{D}_i^P = \begin{bmatrix} \cos\theta_i^P & 0 & -\sin\theta_i^P \\ 0 & 1 & 0 \\ \sin\theta_i^P & 0 & \cos\theta_i^P \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{D}_i^R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_i^R & \sin\theta_i^R \\ 0 & -\sin\theta_i^R & \cos\theta_i^R \end{bmatrix}, \tag{3.5}$$

and the transformation matrix from $\{x', y', z'\}$ to $\{x, y, z\}$ is given by

$$\boldsymbol{D}_i = \boldsymbol{D}_i^R \boldsymbol{D}_i^P \boldsymbol{D}_i^Y \tag{3.6}$$

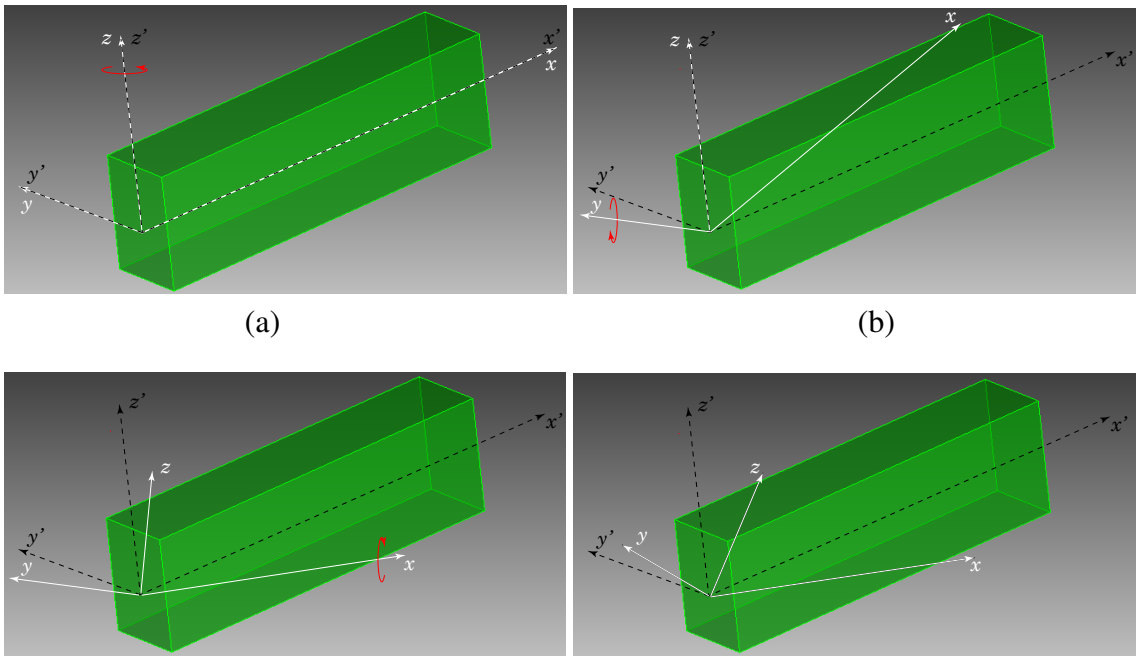

Figure 3-3. Definition of Euler angles. (a) Part and system coordinates start coincidental and the first rotation is applied, (b) orientation after the first rotation where the second rotation is applied, (c) orientation after the second rotation where the third rotation is applied and (d) final system axes orientation.

3.3.2. Inertia Transformation [`inertia_transformation`]

Once the transformation matrix $\boldsymbol{D}_i$ for a part has been determined, the moment of inertia can be easily transformed from the part to the system coordinates from

$$\boldsymbol{I}_i^p = \boldsymbol{D}_i \boldsymbol{I}_i' \boldsymbol{D}_i^T \,. \tag{3.7}$$

3.4. Parallel Axis Transformation [`parallel_axis`]

To calculate the moments of inertia for every part about the system center of gravity, the parallel axis theorem must be applied. This transformation can be succinctly written as

$$\boldsymbol{I}_i = \boldsymbol{I}_i^p + m_i((\boldsymbol{R}_i^T \boldsymbol{R}_i)\mathscr{I} - \boldsymbol{R}_i \boldsymbol{R}_i^T) \tag{3.8}$$

Where $\boldsymbol{I}_i$ is the inertia tensor of the part about the system center of gravity, $\boldsymbol{I}_i^p$ is the inertia tensor of the part about its cg, m_i is the mass of the part, and $\boldsymbol{R}_i$ is the position vector of the part's cg relative to the system cg,

$$\boldsymbol{R}_i = \boldsymbol{r}_i^c - \boldsymbol{d} \tag{3.9}$$

and $\mathscr{I}$ is the 3 by 3 identity matrix. The new transformed inertia tensor components are added to the part array as

`part(i).transformed_inertia`	2-D array (3,3)	Moment of inertia at system centroid

3.5. System Inertia Integration [`inertia_integration`]

To obtain the moment of inertia tensor for the total system, the inertia tensors about the system cg corresponding to each part must be summed. The `inertia_integration` function accomplishes this task and adds the total sum to the structure array system.

$$\boldsymbol{I}^s = \sum_{i=1}^{n} \boldsymbol{I}_i \tag{3.10}$$

The returned system array is now updated with

`system(1).inertia`	2-D array (3,3)	System moment of inertia

3.6. Principal Axes of Inertia [`principals`]

Once the total inertia of the system has been calculated the principal values of the system inertia matrix and the corresponding eigenvectors can be found by solving the eigenvalue problem

$$\boldsymbol{I}^s\boldsymbol{n} = I\boldsymbol{n}. \tag{3.11}$$

The result is a diagonal matrix with the principal inertias I_1, I_2 and I_3 ordered from largest to smallest and a matrix where the three rows are the three principal directions ordered so they correspond to the order of the principal inertias. The input to the function is the system structure array. The function uses the native Matlab function `eig` to find the principal values and directions and then `sort` to put them in the desired order. It also returns the updated system structure array with

`system(1).principal`	2-D array (3,3)	Principal moments of inertia
`system(1).principal_directions`	2-D array (3,3)	Principal directions as columns

3.7. Principal Angle of Inclination [`pai`]

The principal angle of inclination is defined as the angle between the principal direction with the largest direction cosine with respect to the system x-axis and the x-axis itself. Since the largest principal value is the first, it is easy to extract the largest component of the eigenvector to find the principal angle of inclination. The principal angle of inclination, θ_p is given by

$$\theta_p = \cos^{-1}(n_{11}) \tag{3.12}$$

where n_{11} is the first component of the first eigenvector. The function then returns the system array updated with

`system(1).principal_angle_inclination`	Scalar	Principal angle of inclination in degrees

3.8. Output [`spread_gen`]

At the end of the calculation, the part array contains the following components from $i = 1$ to the number of parts:

```
part(i).name
part(i).mass
part(i).centroid
part(i).inertia
part(i).orientation
part(i).transformed_part_inertia
part(i).transformed_inertia.
```

The system array contains the following components:

```
system(1).mass
system(1).system_centroid
system(1).inertia
system(1).principal
system(1).principal_directions
system(1).principal_angle_inclination.
```

The last function in the driver program provides a means to print the `system` structure array containing the result from the calculator as an Excel spreadsheet. This includes a header for each column of the file as follows (this will be important in the optimization example to be shown in Section 5):

mass, centroid x, centroid y, centroid z, Ixx, Iyy, Izz, Ixy, Iyz, Ixz, P1, P2, P3, n11, n21, n31, n12, n22, n32, n13, n23, n33, POI.

The input to this function is the desired name for the file and the `system` array.

4. INPUT FROM CREO MASS PROPERTY TEXT FILES

In some situations it may be desired to input the mass properties of parts or assemblies designed in Creo. One option is to manually copy the needed part parameters to a spreadsheet, but a more convenient option is to read the information directly from Creo mass property text files. These files are written by Creo upon request from the mass properties tool found in the analysis toolbar within Creo. The resulting mass property file contains the information as follows:

```
VOLUME =  1.2296837e+06  INCH^3
SURFACE AREA =  7.4861131e+04  INCH^2
DENSITY =  2.8540561e-01 POUND / INCH^3
MASS =  3.5095863e+05 POUND

CENTER OF GRAVITY with respect to _BRICK coordinate frame:
X   Y   Z     3.0729500e+01 -7.5396653e+01 -6.6343118e+01  INCH

INERTIA with respect to _BRICK coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  4.7197219e+09  8.1313656e+08  7.1549614e+08
Iyx Iyy Iyz  8.1313656e+08  2.5014981e+09 -1.7555123e+09
Izx Izy Izz  7.1549614e+08 -1.7555123e+09  3.1019864e+09

INERTIA at CENTER OF GRAVITY with respect to _BRICK coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  1.1799305e+09  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  6.2537452e+08  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  7.7549661e+08

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   6.2537452e+08  7.7549661e+08  1.1799305e+09

ROTATION MATRIX from _BRICK orientation to PRINCIPAL AXES:
       0.00000        0.00000        1.00000
       1.00000        0.00000        0.00000
       0.00000        1.00000        0.00000

ROTATION ANGLES from _BRICK orientation to PRINCIPAL AXES (degrees):
angles about x  y  z   0.000         90.000         90.000

RADII OF GYRATION with respect to PRINCIPAL AXES:
R1  R2  R3 4.2212603e+01  4.7006941e+01  5.7982942e+01  INCH
```

The parameters that need to be read out of this file are the mass and the inertia tensor with respect to the center of gravity. The centroid location in the system coordinates and the orientation of the part need to be supplied by the user.

When a mass property file corresponds to an assembly, the mass properties file contains information for the assembly as a whole and for each part as in the example below where the assembly contains two parts.

```
VOLUME =  1.2563606e+06  INCH^3
SURFACE AREA =  8.2262496e+04  INCH^2
AVERAGE DENSITY =  2.8536725e-01 POUND / INCH^3
MASS =  3.5852418e+05 POUND

CENTER OF GRAVITY with respect to _ASM0001 coordinate frame:
X   Y   Z     3.4726708e+01 -7.5741995e+01 -6.5732595e+01  INCH

INERTIA with respect to _ASM0001 coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  4.7957594e+09  9.6597376e+08  7.7780723e+08
Iyx Iyy Iyz  9.6597376e+08  2.8805152e+09 -1.7814841e+09
Izx Izy Izz  7.7780723e+08 -1.7814841e+09  3.5357377e+09

INERTIA at CENTER OF GRAVITY with respect to _ASM0001 coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  1.1898581e+09  2.2958311e+07 -4.0587543e+07
Iyx Iyy Iyz  2.2958311e+07  8.9905303e+08  3.5065900e+06
Izx Izy Izz -4.0587543e+07  3.5065900e+06  1.0465782e+09

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   8.9694301e+08  1.0364889e+09  1.2020574e+09

ROTATION MATRIX from _ASM0001 orientation to PRINCIPAL AXES:
      -0.08442        0.24606        0.96557
       0.99536        0.06578        0.07026
      -0.04622        0.96702       -0.25048

ROTATION ANGLES from _ASM0001 orientation to PRINCIPAL AXES (degrees):
angles about x  y  z-164.331         74.921       -108.936

RADII OF GYRATION with respect to PRINCIPAL AXES:
R1  R2  R3 5.0017641e+01  5.3767905e+01  5.7903309e+01  INCH

------------------------------------------------

MASS PROPERTIES OF COMPONENTS OF THE ASSEMBLY
(in assembly units and the _ASM0001 coordinate frame)

   DENSITY             MASS          C.G.:  X        Y         Z

                         TESTING            MATERIAL:           STAINLESS_STEEL_AUSTENITIC
        2.85406e-01  3.50959e+05  3.07295e+01 -7.53967e+01 -6.63431e+01
                            TEST2            MATERIAL:                    STEEL_HIGH_CARBON
        2.83599e-01  7.56555e+03  2.20153e+02 -9.17621e+01 -3.74110e+01
------------------------------------------------
MASS PROPERTIES OF THE PART TESTING
VOLUME =  1.2296837e+06  INCH^3
SURFACE AREA =  7.4861131e+04  INCH^2
DENSITY =  2.8540561e-01 POUND / INCH^3
MASS =  3.5095863e+05 POUND

CENTER OF GRAVITY with respect to _TESTING coordinate frame:
X   Y   Z     3.0729500e+01 -7.5396653e+01 -6.6343118e+01  INCH

INERTIA at CENTER OF GRAVITY with respect to _TESTING coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  1.1799305e+09  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  6.2537452e+08  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  7.7549661e+08

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   6.2537452e+08  7.7549661e+08  1.1799305e+09

CENTER OF GRAVITY with respect to _ASM0001 coordinate frame:
```

```
X   Y   Z     3.0729500e+01 -7.5396653e+01 -6.6343118e+01  INCH

INERTIA at CENTER OF GRAVITY with respect to _ASM0001 coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  1.1799305e+09  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  6.2537452e+08  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  7.7549661e+08

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   6.2537452e+08  7.7549661e+08  1.1799305e+09

-------------------------------------------
MASS PROPERTIES OF THE PART TEST2
VOLUME =  2.6676890e+04  INCH^3
SURFACE AREA =  7.4013650e+03  INCH^2
DENSITY =  2.8359924e-01 POUND / INCH^3
MASS =  7.5655458e+03 POUND

CENTER OF GRAVITY with respect to _TEST2 coordinate frame:
X   Y   Z    -2.5824838e+01 -2.5824838e+01  5.0000000e+00  INCH

INERTIA at CENTER OF GRAVITY with respect to _TEST2 coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  1.7449231e+06  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  1.7449231e+06  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  3.3637539e+06

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   1.7449231e+06  1.7449231e+06  3.3637539e+06

CENTER OF GRAVITY with respect to _ASM0001 coordinate frame:
X   Y   Z     2.2015349e+02 -9.1762075e+01 -3.7411013e+01  INCH

INERTIA at CENTER OF GRAVITY with respect to _ASM0001 coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  1.7449231e+06  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  1.7449231e+06  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  3.3637539e+06

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   1.7449231e+06  1.7449231e+06  3.3637539e+06
```

Only single part or complete assemblies can be input into the calculator. Parts within an assembly can not be used.

4.1. General part reading

To provide flexibility with the input, a function `part_config` was developed to provide flexibility to specify what assemblies or parts are considered in the mass calculations. The first few lines of the driver program when this option is to be used is shown below. The rest of the driver function is the same as in Section 3.

```
clc
clear
close all
```

```
name = 'test_config.txt';
% Provide the input file name to select the proper parts

[part] = part_config(name);
% Use the function part_config to read the data and construct the part structure array

%% Using the driver program.
[system(1)] = deal(cg_calc(part));
% Locates the cg of the system, and total mass of the system

[part] = coordinate_alignment(part);
% Calculates inertia matrix with respect to axes directions of system using part orientation
.
.
.
```

The file `test_config.txt` referenced above is a spreadsheet-generated text file that specifies the names of the files to be read and whether the file is a Creo file or a spreadsheet file. An example of such a file would be:

File Name	Creo/Spreadsheet	Name	centroidX	centroidY	centroidZ	T1	T2	T3
assembly2.txt	Creo	Cube	0	1	-2	0	0	0
brick.txt	C	Sphere	2.5	0	1.3	40	-24	20
POI_Optimization_Problem.csv	Spreadsheet	Prism	0	0	10	0	0	-35

The headers of the columns should be taken as shown above, and the list of files to include *must start on the second line.* Here the headers just indicate the fields expected for each column, which must be followed. Each row corresponds to a single file (part or assembly) and the part name is taken from the file name. The fields needed for each row are:

File Name: The name of the file to be read.

Creo/Spreadsheet: Write Creo or Spreadsheet to indicate a Creo mass property text file or a spreadsheet input similar to the one described in Table 3.1.

centroidX, centroidY, centroidZ: System coordinates of the part's center of mass.

T1, T2, T3: Yaw, pitch and roll Euler angles as defined in Fig. 3-3

The function `creo_config` uses the information provided to read either the Creo mass property files or the spreadsheet files by using a set of three functions `loading_parts`, described in Section 3, `CREO_Input` to read parts and `CREO_Assembly` to read assemblies.

5. OPTIMIZATION

The calculator can be used as a function to be evaluated in an optimization procedure. In these procedures, the optimization scheme must be able to change appropriate parameters in the `part` array based on results obtained in the `system` array. The communication can be accomplished through spreadsheet (Excel or csv) files that are read and written by wrappers around the calculator as demonstrated by the following example.

In order to accomplish optimization procedures, an appropriate programming structure needs to be developed that wraps around the calculator. Although it would be easy to share variables between the optimization scripts and the calculator because the whole procedure is done in Matlab, in this example we pretend that is not the case. Instead the communication between the required scripts and the calculator are conducted through the reading and writing of spreadsheet files. Also, to keep this example simple, all the input data is provided through the spreadsheet option. The input spreadsheet is shown in Fig. 5-1.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	name	mass	centroid x	centroid y	centroid z	Ixx	Iyy	Izz	Ixy	Iyz	Ixz	T1	T2	T3	
2	Cylindrical	23.6503	10	0	0	723.1618438	1483.18	1483.18	0	0	0	0	0	0	
3	Rod_1	0.392699	2.75	1	-1	0.049087385	0.842667	0.842667	0	0	0	0	0	0	
4	Rod_2	0.392699	7.165111	-0.60697	-1	0.049087385	0.842667	0.842667	0	0	0	0	0	40	
5	Sphere	0.07854	0	0	0	0.006483315	0.006483	0.006483	0	0	0	0	0	0	
6															

Figure 5-1. Input spreadsheet for optimization example.

The objective of the example is to move a sphere around a system in order to minimize the principal axis of inclination of the system using a constrained optimization function `fmincon` in Matlab. The front script is as follows:

```
clc
clear all
close all

%name = 'test_config.txt';
%[part] = part_config(name);

global Part_Name Inp_Param Out_Param inp_name

inp_name = 'Optimization_Problem_Metric.csv';
%Defines the name of the spreadsheet.

Part_Name = 'Sphere';
%Defines what number mass this is in the spreadsheet

Inp_Param = ["centroidX";"centroidY";"centroidZ"];
%Defines what parameter will be iterated through this can be a vector of parameters.

Out_Param = ["POI"];
%Defines the parameter that is minimized. If multiple outputs need to be optimized, then the sum of squares will
```

```
options = optimoptions('fmincon','Display','iter','PlotFcn',@optimplotfval)
% Optimizer options

XX = fmincon(@POI_opt_driver,[0.254,0,0],[],[],[],[],[0.955*0.0254,-5.296*0.0254,-5.296*0.0254],[19.045*0.0254,5.
%Constrained Optimizer where A*x <= b
```

It basically defines the name of the input spreadsheet, the part to be moved around, the parameters to be varied during the optimization and the parameter to be minimized. It then sets the optimization function options and calls the optimizer. The function to be optimized is `POI_opt_driver`. The rest of the parameters include an initial guess and the constraints to be prescribed in the optimization. The objective of the function `POI_opt_driver` is to provide the value of the objective function (the principal angle of inclination) given the location of the sphere defined by its centroid location as given below.

```
function Y = POI_opt_driver(X)
global Part_Name inp_name Inp_Param Out_Param
% Defines the global variables

warning('off','all')

loaded = readtable(inp_name);
% Read the input spreadsheet

mass_num = find(strcmp(Part_Name,loaded.name));
% Find the index of the mass number

for i = 1:length(Inp_Param)
    loaded.(Inp_Param(i))(mass_num) = X(i);
end
% Introduce the current guess X into the loaded array

writetable(loaded,inp_name);
% Overwrite the loaded array with the current guess

out_name = driver_fun(inp_name);
% Call the calculator driver with the current input array

output = readtable(out_name);
% Read the value of the variable being minimized from the output spreadsheet

Y = output.(Out_Param);
% Return the current function value
```

This function is called by `fmincon` and essentially reads the input spreadsheet, inserts the current guesses for the sphere's centroid, and replaces the old input spreadsheet with the updated one. It then calls the function that drives the calculator, which is a slightly modified version of the one shown in Section 3 as shown below, to conduct the mass property calculation and output a spreadsheet that contains the results. The function `POI_opt_driver` then reads this file and extracts the value of the principal angle of inclination and returns it to `fmincon` to generate a new guess. The process continues until it converges.

```
function out_name = driver_fun(inp_name)

part = loading_parts(inp_name);
```

```
% Loads the spreadsheet and puts it into a structured array containing all the parts.

[system(1)] = deal(cg_calc(part));
% Locates the cg of the system, and total mass of the system.

[part] = coordinate_alignment(part);
% Calculates inertia matrix with respect to axes directions of system using part orientation

[part] = deal(parallel_axis(part,system));
%Transforms the MOI's of each part to the system cg location.

[system] = deal(inertia_integration(part,system));
%This sums the inertia for the entire system.

[system] = principals(system);
%This calculates the principal MOI for the system, and the principal directions.

[system] = pai(system);
%Calculates the largest angle from principal axis to system axis

out_name = append('System_',inp_name);
spread_gen(out_name,system)
%Generates an Excel output of the System Properties with name specified.
```

6. INTERFACING WITH PHOENIX INTEGRATION MODEL CENTER

Model Center is an integration program that allows for the communication between different engineering software products, i.e. CAD platforms, finite element analysis codes, programming languages, and optimization algorithms. The specific use case detailed here is in regards to the use of the MATLAB mass property calculator in this report. This section has the purpose of walking the reader trough a simple example of an interface between Model Center and the mass property calculator.

6.1. Process Model Overview

Model center has two different types of models, but the focus here will be on the process model type. This type of model uses a linear flowchart-like organization to complete all of the required tasks in the correct order. The process we will be focusing on can be seen in Figure 6-1.

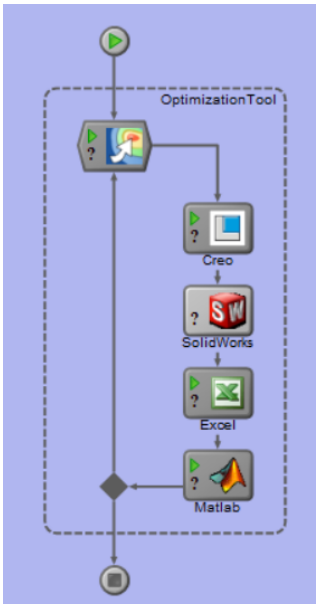

Figure 6-1. Example Process Model

The basic workflow of creating a process model is drag and drop. The desired module can be dragged and placed into the correct location in the process flowchart.

6.2. Importing Solidworks Parts

To import a Solidworks part into Model Center, begin by dragging the Solidworks icon into the process flowchart. From there you will be prompted to open your desired Solidworks file. Once it has been opened, select Mass, MomentsOfInertia_X_X, MomentsOfInertia_Y_Y, MomentsOfInertia_Z_Z, MomentsOfInertia_X_Y, MomentsOfInertia_X_Z, and MomentsOfInertia_Y_Z from the prompt. Click 'OK.' This can be seen in Figure 6-2.

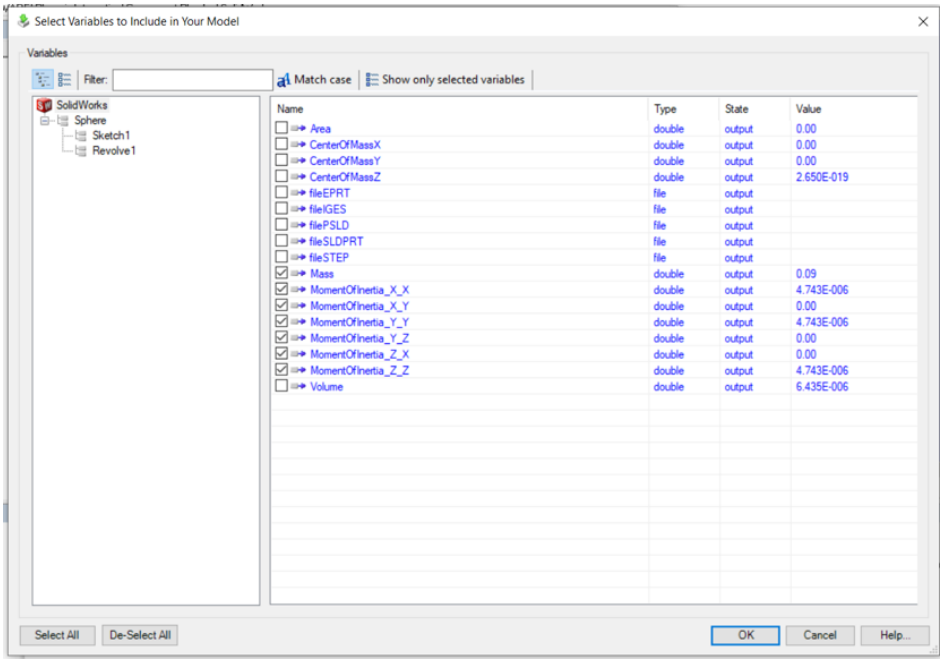

Figure 6-2. Selecting Solidworks Variables

To define a part's location and rotation within a total assembly, the user should define six additional variables L1, L2, L3, T1, T2, T3 that describe the part's centroid location and orientation in the system coordinates. This can be seen in Figure 6-3. Click on 'click to add' to include these variables and then click 'apply changes.'

Be aware that SolidWorks saves the models in SI units, independently of the system of units used to design the part. So, unless all models are introduced via SolidWorks, parts introduced by Creo or via a spreadsheet must be in SI units.

6.3. Importing Creo Parametric Parts

6.3.1. Preparing Creo Parametric Parts for Import

Unfortunately, Model Center does not support directly reading in the mass properties of a Creo Parametric part. This can, however, be circumvented by defining new variables within the Creo model that are equal in value to the mass properties that are required. To begin this process, open

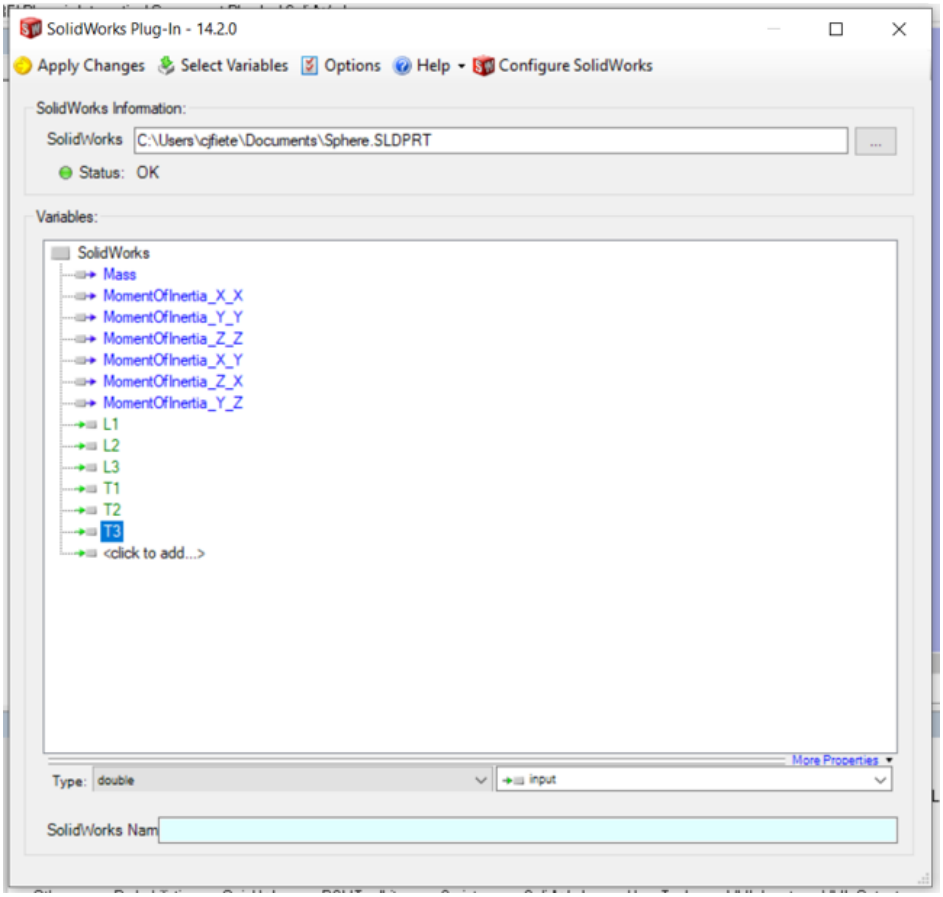

Figure 6-3. Additonal Solidworks Variables

the desired part within Creo Parametric. Once the part is open, navigate to the ‘Mass Properties’ tool under the ‘Analysis’ ribbon. Within this tool, select ‘Feature’ from the drop down near the bottom of the panel and click ‘preview.’ Then change ribbons of the ‘Mass Properties’ tool from ‘Analysis’ to ‘Feature’ (see Fig. 6-4) and select all center of gravity locations (XCOG, etc.), and moments of inertia (MP_IXX, etc.).

Next, user defined variables must be created. This can be done by navigating to the ‘Tools’ tab in the top ribbon then selecting ‘d= relations.’ Selecting this will open the tool. Within this tool, the user can define new variables, as can be seen in Figure 6-5.

The user should define new variables with the titles IXX, IYY, IZZ, IXY, IXZ, IYZ by selecting the drop-down ‘Local Parameters,’ clicking on the ’+’ in the lower-left corner and typing new variable names listed previously. The moments of inertia at the center of gravity are of interest to us, but Creo does not directly output these values. As such, they must be defined through a relation. For ease of use, the text that follows can simply be copied and pasted into the Relations textbox in the ‘Relations’ tool window.

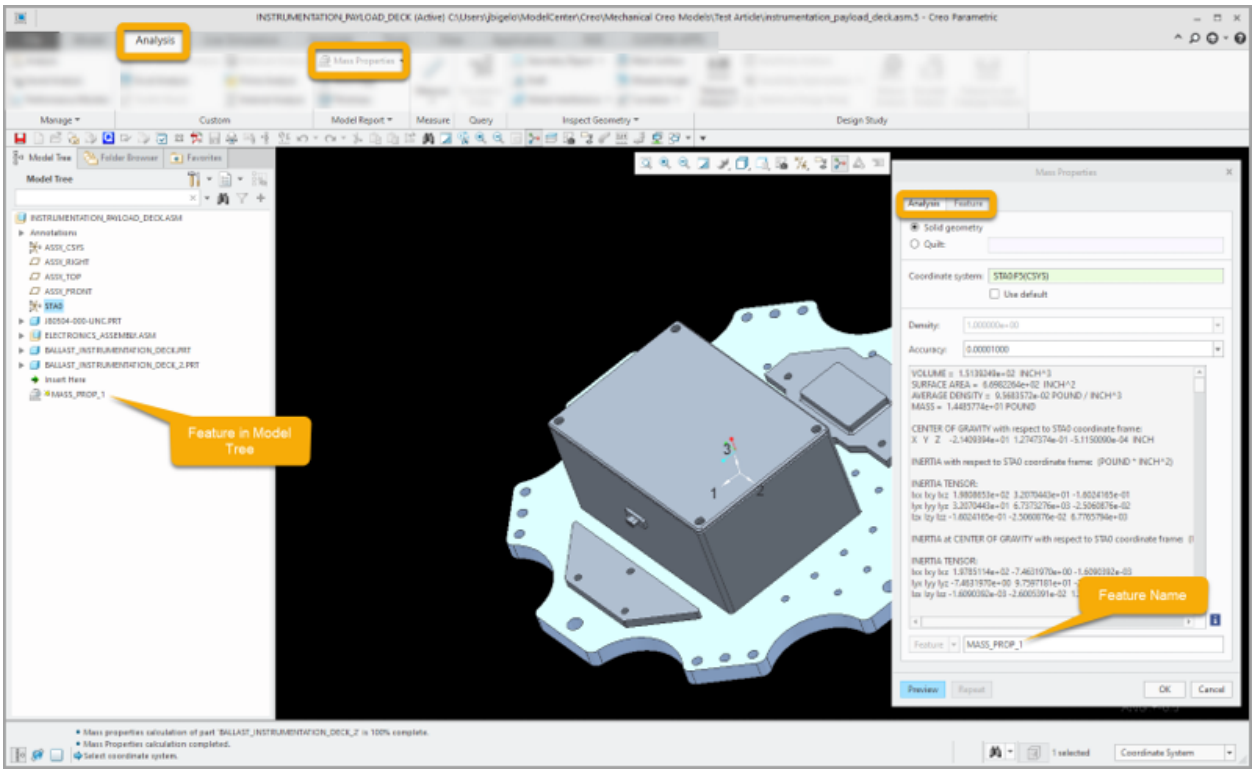

Figure 6-4. Obtaining Correct Mass Properties

```
IXX = MP_IXX:FID_MASS_PROP_1 - MASS:FID_MASS_PROP_1*(YCOG:FID_MASS_PROP_1^2 + ZCOG:FID_MASS_PROP_1^2)
IYY = MP_IYY:FID_MASS_PROP_1 - MASS:FID_MASS_PROP_1*(XCOG:FID_MASS_PROP_1^2 + ZCOG:FID_MASS_PROP_1^2)
IZZ = MP_IZZ:FID_MASS_PROP_1 - MASS:FID_MASS_PROP_1*(YCOG:FID_MASS_PROP_1^2 + XCOG:FID_MASS_PROP_1^2)
IXY = MP_IXY:FID_MASS_PROP_1 + MASS:FID_MASS_PROP_1*(XCOG:FID_MASS_PROP_1*YCOG:FID_MASS_PROP_1)
IXZ = MP_IXZ:FID_MASS_PROP_1 + MASS:FID_MASS_PROP_1*(XCOG:FID_MASS_PROP_1*ZCOG:FID_MASS_PROP_1)
IYZ = MP_IYZ:FID_MASS_PROP_1 + MASS:FID_MASS_PROP_1*(ZCOG:FID_MASS_PROP_1*YCOG:FID_MASS_PROP_1)
```

Now simply select ‘OK’ in the ‘Relations’ window, and regenerate the model by going to ‘Model’ in the top ribbon and selecting ‘Regenerate.’ At this point the file is ready for Model Center and can simply be saved.

6.3.2. Creo Part Import

To import a Creo Part into Model Center, begin by dragging the Creo icon into the process flowchart. From there you will be prompted to open your desired Creo file. If your model does not appear, you may need to choose ‘Versioned Creo Files’ from the box above the ‘Open’ button. Once this has been opened, select your part from the tree. From the menu on the right, select mass, IXX, IYY, IZZ, IXY, IXZ, and IYZ from the prompt and click ‘OK.’ This will take you back to the previous menu box. This can be seen in Figure 6-6. Click ‘Apply Changes’ to close.

To define a part’s location and rotation within a total assembly, add six user defined variables titled L1, L2, L3, T1, T2, T3. This can be seen in Figure 6-7.

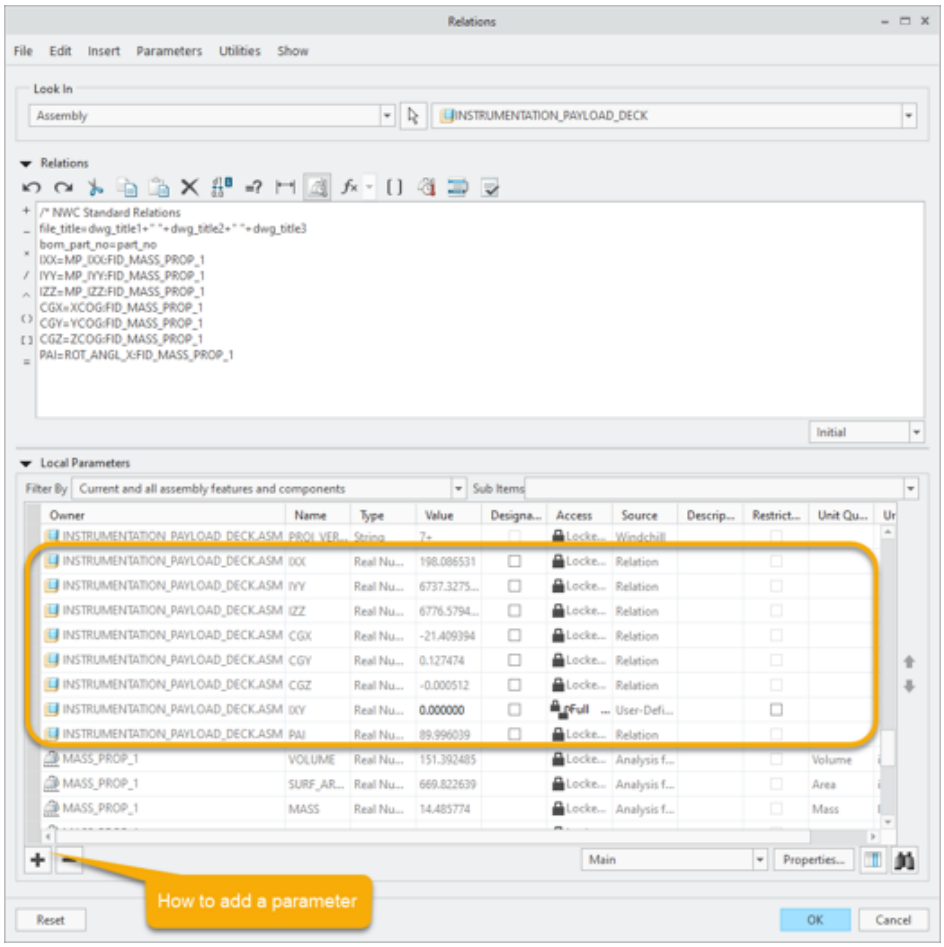

Figure 6-5. Adding User Defined Variables

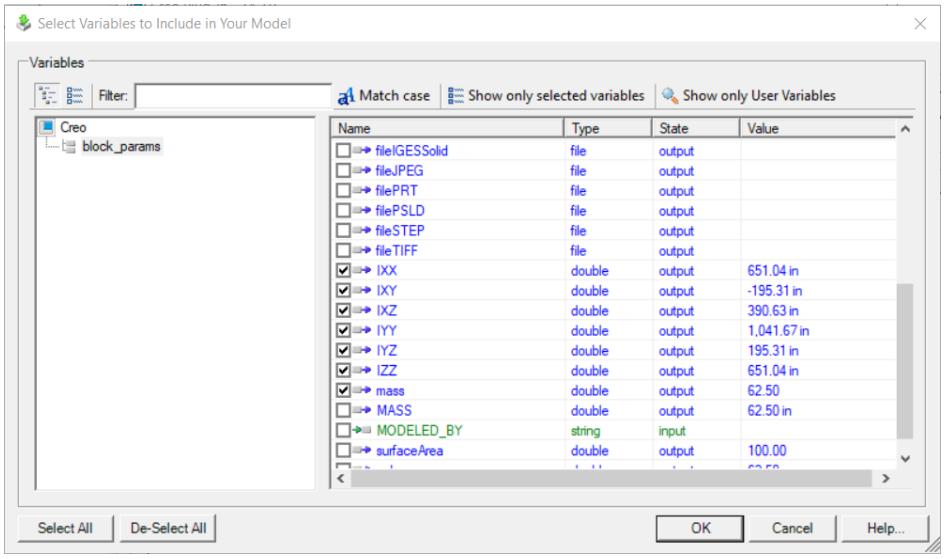

Figure 6-6. Selecting Creo Variables

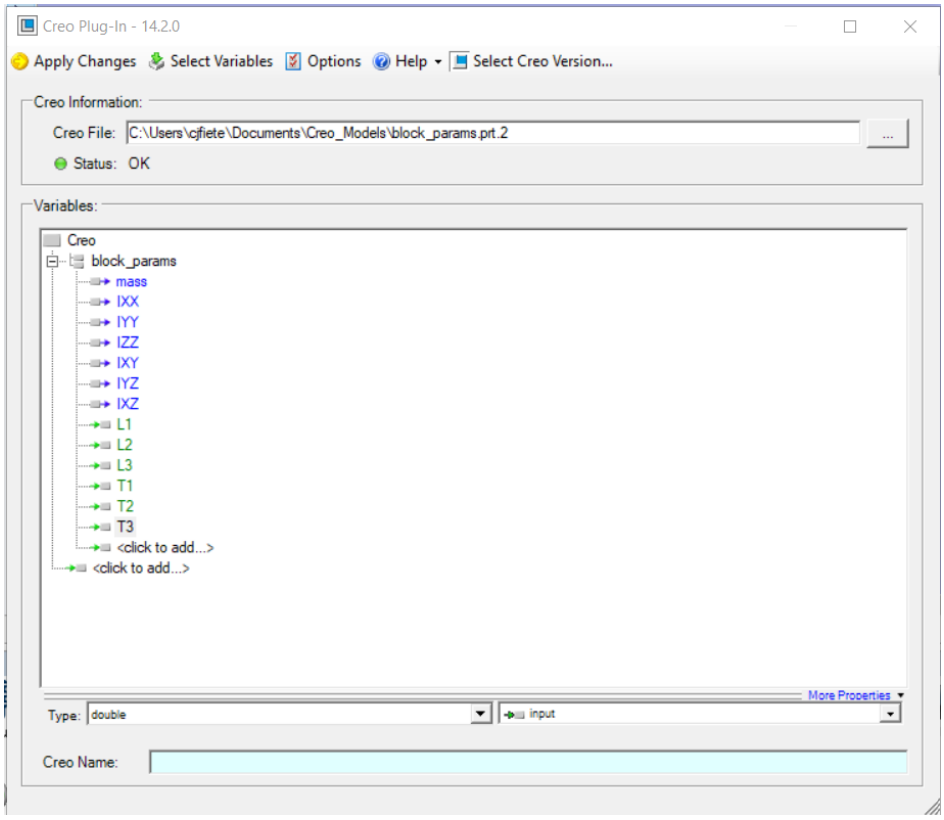

Figure 6-7. Selecting Creo Variables

6.4. Manual Input with Excel

To manually define a part's mass properties, a Microsoft Excel spreadsheet can be used. This spreadsheet should be in the format of Table 6-1. Note that this format is not the same as what was used in the previous sections.

Table 6-1. Format of Excel Spreadsheet

Mass	Value
Centroid X	Value
Centroid Y	Value
Centroid Z	Value
Ixx	Value
Iyy	Value
Izz	Value
Ixy	Value
Ixz	Value
Iyz	Value
T1	Value
T2	Value
T3	Value

To include the spreadsheet in the process model, begin by dragging the Microsoft Excel module into the process model flowchart. Once this module has been included, an 'Import Variables' window pops up. Click on 'Import Name/Value Pairs.' It will prompt you to select variables from Microsoft Excel. This can be done by selecting the two columns of the spreadsheet and clicking 'OK.' Select 'Yes' when asked to save changes.

6.5. Part Location and Orientation

While the Excel spreadsheet input should have all parameters defined already, the parts introduced with SolidWorks and Creo still need to have the location of the centroid and orientation defined. This can be done in the component tree by simply clicking on the values for L1, L2, L3 and T1, T2, T3 and entering the appropriate values. Note that the Ts are in degrees and the Ls have to be consistent with the system of units being used

6.6. Creation of Mass Properties Array

Once all parts have been added to Model Center, click on the green play button at the top of the flowchart to create all the variables. To integrate all values into the mass property calculation, a

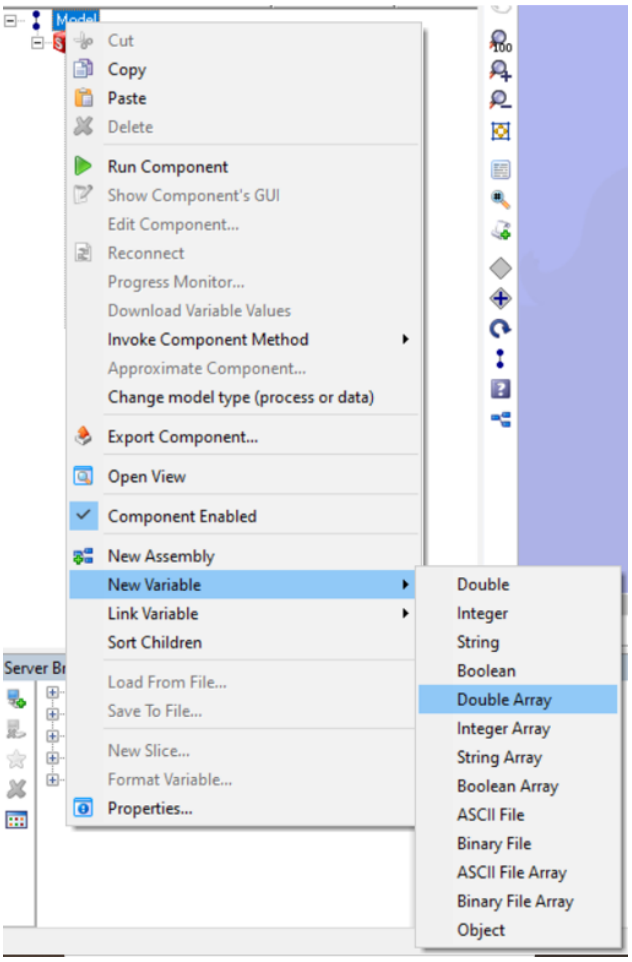

Figure 6-8. Double Array Creation

double array must be created. This can be done by right clicking the main model in the model tree, and selecting "New Variable" then "Double Array" as can be seen in Figure 6-8.

Next a dialogue box will appear asking for the name of this variable. This array should be named "Mass Properties". Once named, the array can be double clicked in the tree, asking for the number of variables and size. Set the number of dimensions to two. Then, the size of this array should be structured such that dimension 0 corresponds to the 13 mass property fields and dimension 1 corresponds to the number of parts in the model. Each column will have the format of Table 6-2. Close the dialog that appears.

6.7. Linking Variables

Now to connect each part of the flowchart, right click on the double array and select ‘Link Variable’ and ‘Show Variable in Link Editor’. This opens the Link Editor. Expand all variables. In this window, you connect variables by dragging them to one another. For each part, each variable should be dragged to the corresponding container in the Mass Properties double array.

Table 6-2. Format of Model Center Input Array

	Mass
	Centroid X
	Centroid Y
	Centroid Z
	Ixx
	Iyy
	Izz
	Ixy
	Ixz
	Iyz
	T1
	T2
	T3

Lines connecting the variables will automatically appear. This can be seen in Figure 6-9. The warning signs shown can be neglected for the Creo Parts. Select ‘OK’ when done and click on the play button again. The values will be updated in the component tree.

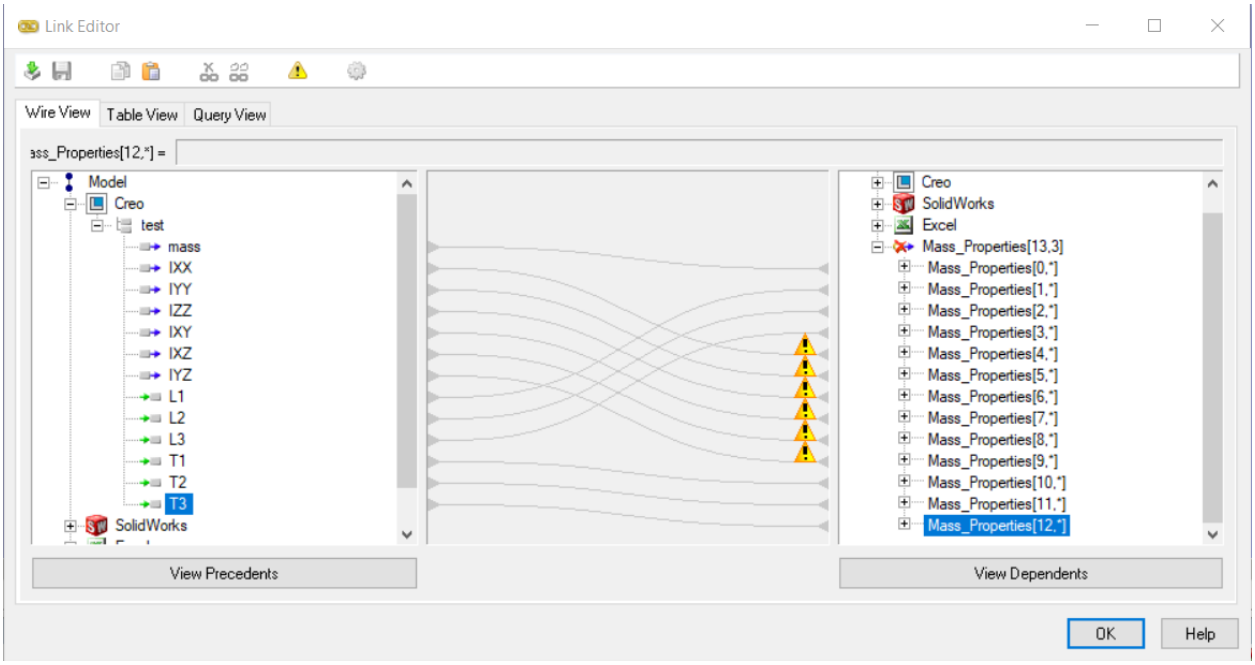

Figure 6-9. Variable Linking

6.8. MATLAB Mass Property Calculator

To use the MATLAB Mass Property Calculator begin by dragging the MATLAB module onto the end of the flowchart . A dialogue box will ask what you would like to do with MATLAB, select "Call External M-file from the plug-in" as shown in Figure 6-10.

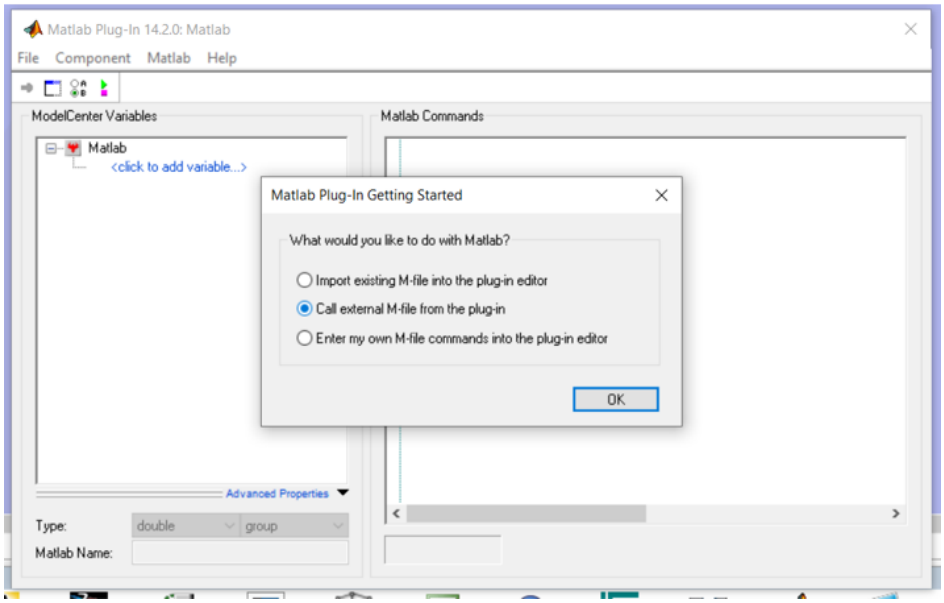

Figure 6-10. Load MATLAB Program

Select Model_Center_Connection.m, click OK and close the dialogue box. Next the input array must be linked using the link editor described above, but the process is shown in Figure 6-11. Expand the Matlab icon to show the 'Mass_Properties[0]' in the figure and click 'OK'

Once linked, the process will be ready to calculate the system mass properties.

6.9. Calculating Mass Properties of the System

Clicking on the green play button will run model center and the calculator. It will return an array with all of the system's mass properties in the component tree under the Matlab icon in the format shown in Table 6-3.

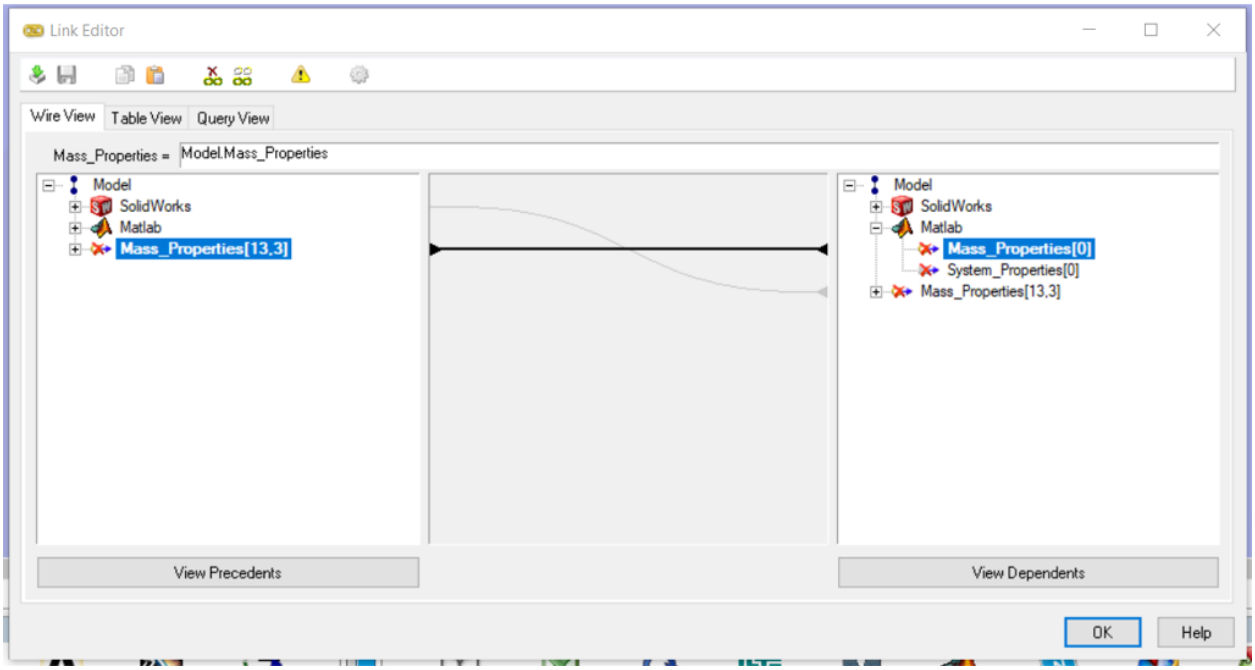

Figure 6-11. Input Array Linking

Table 6-3. Format of Model Center System Array

Mass	
Centroid X	
Centroid Y	
Centroid Z	
Ixx	
Iyy	
Izz	
Ixy	
Ixz	
Iyz	
P11	
P22	
P33	
PAI	

6.10. Optimization

Model Center also includes an optimization feature. This can be used with the process above. To include an optimization, drag the optimization tool onto the flowchart. A dialogue box will open where the objective, constraints, and algorithm can be chosen. Select OK, and the dialogue box will close. Now all of the previous features on the flowchart must be moved inside of the optimization tool and can be done simply by dragging the icons. Once this is completed, the model will be finished and can be run by the green play button at the top.

7. EXAMPLES AND VERIFICATION

This chapter has the objectives of providing use examples of the calculator while at the same time ensuring that the results of the calculations are correct. The latter is done in some cases by running the calculator in different ways to observe consistency and in others by comparing the results to those from calculations external to the calculator.

7.1. Problem 1: Coordinate Transformation, Principal Values and Directions

This problem has two sections, each with a different objective. The first is to input a single part with zero products of inertia ($I_{x'y'} = I_{y'z'} = I_{x'z'} = 0$) to verify both the transformation matrix for the part and the calculation of the principal values and directions. The second is to ensure the processing is correct when the part moment of inertia is fully populated.

In the first section the part has zero products of inertia in the part coordinate system. The part and system coordinates, however, are non-coincidental. The calculator is then used to find the inertia matrix, its principal values and principal directions. The results of the principal quantities are then compared to the original part properties. In this way two independent parts of the algorithm are checked: the transformation between the coordinate systems and the calculation of the principal values and directions.

The second part of the exercise consists of using the system inertia tensor found above as input and then calculating the principal values and directions. The results should correlate to the results of the first exercise. Since the input to the first part had zero cross moments, this is a way to verify that the input of the cross moments of inertia is being read and processed correctly.

For the first section of the exercise, the inertia matrix is taken to be

$$\boldsymbol{I}' = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.1}$$

and the the orientation of the part is prescribed by the three Euler angles

$$\theta_Y = -38^\circ, \theta_P = 43^\circ, \theta_R = 135^\circ. \tag{7.2}$$

The transformation matrix between the part and the system coordinates is then calculated as

$$\boldsymbol{D} = \begin{bmatrix} 0.5763 & -0.4503 & -0.6820 \\ -0.0553 & -0.8541 & 0.5171 \\ -0.8154 & -0.2603 & -0.5171 \end{bmatrix} \tag{7.3}$$

and the resulting components of the inertia matrix are then

$$\boldsymbol{I}_s = \begin{bmatrix} 1.86702 & 0.32081 & -0.82259 \\ 0.32081 & 1.73562 & 0.31255 \\ -0.82259 & 0.31255 & 2.39736 \end{bmatrix}. \tag{7.4}$$

The calculated principal values, as expected, are

$$\boldsymbol{I}_p = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.5}$$

with the principal directions given by the columns of the matrix

$$\hat{\boldsymbol{V}} = \begin{bmatrix} -0.5763 & 0.4503 & -0.6820 \\ 0.0553 & 0.8541 & 0.5171 \\ 0.8154 & 0.2603 & -0.5171 \end{bmatrix}. \tag{7.6}$$

Changing the sign of the first and second columns does not affect the principal directions and gives

$$\boldsymbol{V} = \begin{bmatrix} 0.5763 & -0.4503 & -0.6820 \\ -0.0553 & -0.8541 & 0.5171 \\ -0.8154 & -0.2603 & -0.5171 \end{bmatrix}. \tag{7.7}$$

The transformation of the principal directions to the part coordinates is given by $\boldsymbol{D}^T\boldsymbol{V} = \mathscr{I}$ (identity matrix), which confirms the results.

For the second section, the input to the calculator is the inertia matrix $\boldsymbol{I}_s$ in (7.4) with $\theta_Y = \theta_P = \theta_R = 0$. The calculation of the principal values returns the matrix $\boldsymbol{I}_p$ in (7.5) and the principal directions $\hat{\boldsymbol{V}}$ in (7.6) thus verifying that the cross-moments of inertia are being read and processed correctly when a fully populated inertia matrix is used as input.

7.2. Problem 2: Transformation Benchmark Comparison

Whereas the problem above provides verification that the coordinate transformation (rotation) matrix and the matrix of principal directions agree, it does not provide an independent check that the calculation of the rotation matrix, given by the three arbitrary Euler angles, is correct. In order to carry out a validation of the calculation of the rotation matrix, a double-check procedure was developed.

We consider a relatively slender bar of circular cross-section and moments of inertia components $I_{x'x'} = 0$, $I_{y'y'} = I_{z'z'} = I = 0.818125$, $I_{x'y'} = I_{y'z'} = I_{x'z'} = 0$ as shown in Fig. 7-1 The bar is centered at the origin but has an arbitrary orientation with respect to the $\{x, y, z\}$ axis. The orientation of the bar axis is given by randomly picking a point in each of the 8 octants of the 3-D space. Given a triad, the axis of the bar is from the origin to the point chosen and defines the local x' axis and

the direction $\boldsymbol{e}_{x'}$. Note that this and the other two directions $\boldsymbol{e}_{y'}$ and $\boldsymbol{e}_{z'}$ that define the part coordinates must satisfy

$$|\boldsymbol{e}_{x'}| = |\boldsymbol{e}_{y'}| = |\boldsymbol{e}_{z'}| = 1 \tag{7.8}$$

and

$$\boldsymbol{e}_{x'} \cdot \boldsymbol{e}_{y'} = \boldsymbol{e}_{x'} \cdot \boldsymbol{e}_{z'} = \boldsymbol{e}_{y'} \cdot \boldsymbol{e}_{z'} = 0. \tag{7.9}$$

Since $I_{y'y'} = I_{z'z'}$. the orientations of $\boldsymbol{e}_{y'}$ and $\boldsymbol{e}_{z'}$ are arbitrary as long as they satisfy the relations above and, together with $\boldsymbol{e}_{x'}$, form a right handed coordinate system.

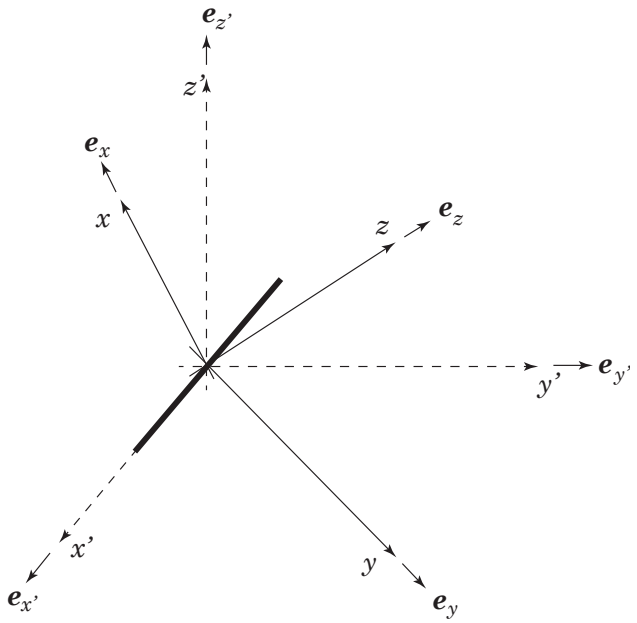

Figure 7-1. Geometry for example 2.

Note that the components of the moment of inertia $\{I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{yz}, I_{xz}\}$ can now be found in two different manners. In the first, the direction cosines of x' are simply the three components of $\boldsymbol{e}_{x'} = \{\cos\theta_x, \cos\theta_y, \cos\theta_z\}$. The moment of inertia in the $\{x, y, z\}$, or system, frame is given by

$$\boldsymbol{I}^s = I \begin{bmatrix} \cos^2\theta_y + \cos^2\theta_z & -\cos\theta_x\cos\theta_y & -\cos\theta_x\cos\theta_z \\ -\cos\theta_x\cos\theta_y & \cos^2\theta_x + \cos^2\theta_z & -\cos\theta_y\cos\theta_z \\ -\cos\theta_x\cos\theta_z & -\cos\theta_y\cos\theta_z & \cos^2\theta_x + \cos^2\theta_y \end{bmatrix}. \tag{7.10}$$

In the second, the rotation matrix can be constructed as

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{e}_{x'} & \boldsymbol{e}_{y'} & \boldsymbol{e}_{z'} \end{bmatrix}, \tag{7.11}$$

so that the components of a vector $\boldsymbol{V}$ in the part and system coordinates are related by

$$\boldsymbol{V} = \boldsymbol{R}\boldsymbol{V}', \boldsymbol{V}' = \boldsymbol{R}^T\boldsymbol{V}. \tag{7.12}$$

Given $\boldsymbol{R}$ the Euler angles for the rotation from the primed to the unprimed axis can be calculated. This can be done using the Matlab function `rotm2eul`, but one must be careful to note that in

Table 7-1. Rotation exercise results.

x	-0.3	-0.1	-0.12	-0.21	0.02	0.51	0.60	0.99
y	0.67	-0.33	-0.13	0.75	-0.01	-0.24	0.15	0.72
z	-0.63	-0.9	0.16	0.87	0.72	-1.00	-0.88	1.00
$\cos\theta_x$	-0.3101	-0.1038	-0.5031	-0.1798	0.0278	0.4443	0.5578	0.6263
$\cos\theta_y$	0.6926	-0.3424	-0.5450	0.6423	-0.0139	-0.2091	0.1395	0.4555
$\cos\theta_z$	-0.6513	-0.9338	0.6708	0.7451	0.9995	-0.8711	-0.8182	0.6326
θ^Y (°)	114.9	105.7	171.3	101.3	-88.39	-62.23	-56.00	-37.18
θ^P (°)	-42.51	67.49	-59.41	-23.44	-7.958	-8.269	4.008	-38.18
θ^R (°)	-154.2	-55.28	-28.99	126.2	-91.02	99.36	83.02	-93.42
I_{xx}	0.739	0.809	0.611	0.792	0.817	0.657	0.564	0.497
I_{yy}	0.425	0.722	0.575	0.481	0.818	0.782	0.802	0.648
I_{zz}	0.471	0.105	0.450	0.364	7.9×10^{-4}	0.197	0.270	0.491
I_{xy}	0.176	-0.029	-0.224	0.095	3.1×10^{-4}	0.076	-0.064	-0.233
I_{yz}	0.369	-0.262	0.299	-0.392	0.011	-0.149	0.093	-0.236
I_{xz}	-0.165	-0.079	0.276	0.110	-0.023	0.317	0.373	-0.324

Matlab the transformation is defined by the vector pre-multiplying the rotation matrix, opposite of how it is defined above. Therefore the input to `rotm2eul` must be $\boldsymbol{R}^T$. Once the Euler angles are known, they can be put through the calculator to calculate $\boldsymbol{I}^s$.

A third way to obain $\boldsymbol{I}^s$ is to draw the part using a CAD program (in this case SolidWorks) and then output the mass properties of the part. In summary, three independent ways of calculating $\boldsymbol{I}^s$ have been devised, and the verification of the procedures are that they all should produce the same results. Table 7.2 gives the results, with all three methods giving the same answer for the components of the moment of inertia, thus providing evidence verifying the procedure.

7.3. Problem 3: Mass Properties for a Two-Bar System

This problem is used as verification of the results when rotations are applied about a single axis over the range between 0 and 360 degrees, as well as a simple check on the cg calculation and the parallel axes theorem. The problem consists of two bars, each with the geometry shown in Fig. 7-2(a) and mass m. The axes $\{x,y,z\}$ are centroidal. If the density is uniform and $w \neq h \neq l$ then $I_{x'x'} \neq I_{y'y'} \neq I_{z'z'}$ and $I_{x'y'} = I_{y'z'} = I_{z'x'} = 0$. Three configurations of the two bars have been considered as shown in Figs. 7-2(b)-(d). In each case, the bottom bar is kept fixed while the top bar yaws, pitches and rolls respectively. In all cases the center of the bottom bar is at $(0,0,-a)$ and that of the top bar is at $(0,0,a)$. The following values are adopted: $m = 1$, $l = 1$, $h = 0.2$ and $w = 0.1$.

The verification was conducted first for the yaw rotation. A driver script varied the angle θ^Y in increments of 15° from 0 to 360°. These results were compared to solutions obtained through a

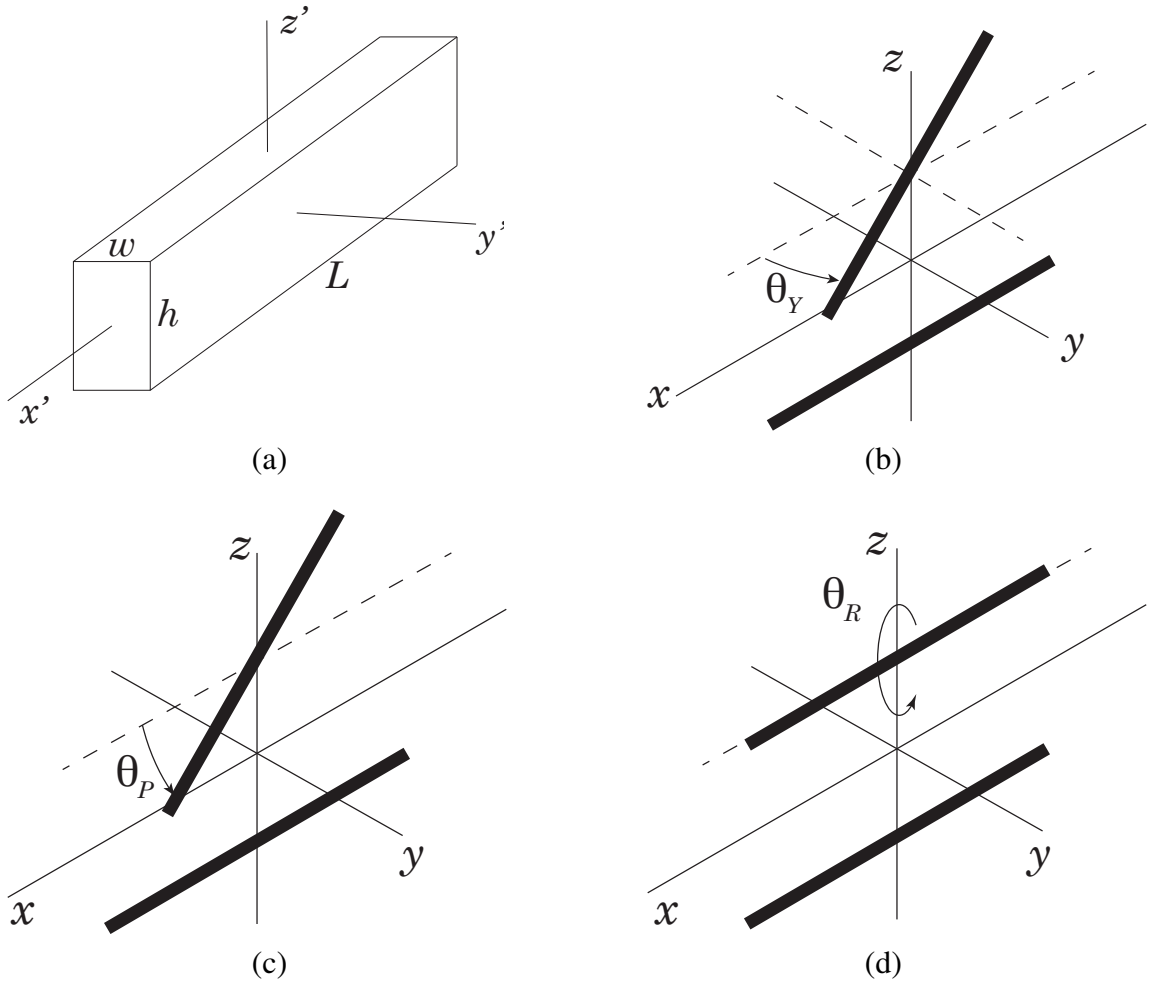

Figure 7-2. Two bar example. (a) Local axes, (b) yaw rotation, (c) pitch rotation and (d) roll rotation.

separate formulation. Here, the moments of inertia for the rotating bar components are given by

$$\begin{aligned} I^p_{xx} &= I'_{y'y'}\sin^2\theta_Y + I'_{x'x''}\cos^2\theta_Y \\ I^p_{yy} &= I'_{x'x'}\sin^2\theta_Y + I'_{y'y'}\cos^2\theta_Y \\ I^p_{zz} &= I'_{z'z'} \\ I^p_{yz} &= (I'_{y'y'} - I'_{x'x'})\sin\theta_Y\cos\theta_Y . \end{aligned} \tag{7.13}$$

The total moment of inertia of the system about its centroid (at the origin of the x, y, z axes) is then given by

$$\begin{aligned} I^s_{xx} &= [I'_{x'x'} + m(-a)^2] + [I^p_{xx} + ma^2] \\ I^s_{yy} &= [I'_{y'y'} + m(-a)^2] + [I^p_{yy} + ma^2] \\ I^s_{zz} &= [I'_{z'z'}] + [I^p_{zz}] \\ I^s_{xy} &= I^p_{xy} \\ I^s_{xz} &= I^p_{xz} \\ I^s_{yz} &= I^p_{yz}. \end{aligned} \tag{7.14}$$

The comparisons between the results by the calculator to those from (7.13) and (7.14) are shown in Fig. 7-3, thus completing the verification for this case.

The second exercise is for the pitch rotation. In this case the transformation of the moment of inertia components is given by

$$\begin{aligned} I^p_{xx} &= I'_{z'z'}\sin^2\theta_P + I^p_{x'x'}\cos^2\theta_P \\ I^p_{yy} &= I'_{y'y'} \\ I^p_{zz} &= I'_{x'x'}\sin^2\theta_P + I^p_{z'z'}\cos^2\theta_P \\ I^p_{zx} &= (I'_{x'x'} - I^p_{z'z'})\sin\theta_P\cos\theta_P. \end{aligned} \tag{7.15}$$

The parallel axis equations are the same as those in (7.14). The results are shown in Fig. 7-4.

Finally, the exercise is repeated for the roll rotation. The transformation equations are

$$\begin{aligned} I^p_{zz} &= I'_{z'z'} \\ I^p_{yy} &= I'_{z'z'}\sin^2\theta_R + I^p_{y'y'}\cos^2\theta_R \\ I^p_{zz} &= I'_{y'y'}\sin^2\theta_R + I^p_{z'z'}\cos^2\theta_R \\ I^p_{yz} &= (I'_{y'y'} - I^p_{z'z'})\sin\theta_R\cos\theta_R, \end{aligned} \tag{7.16}$$

and the parallel axis theorem is still given by (7.14). The results are shown in Fig. 7-5.

In all three cases the agreement between the two approaches is excellent, therefore providing the desired verification result.

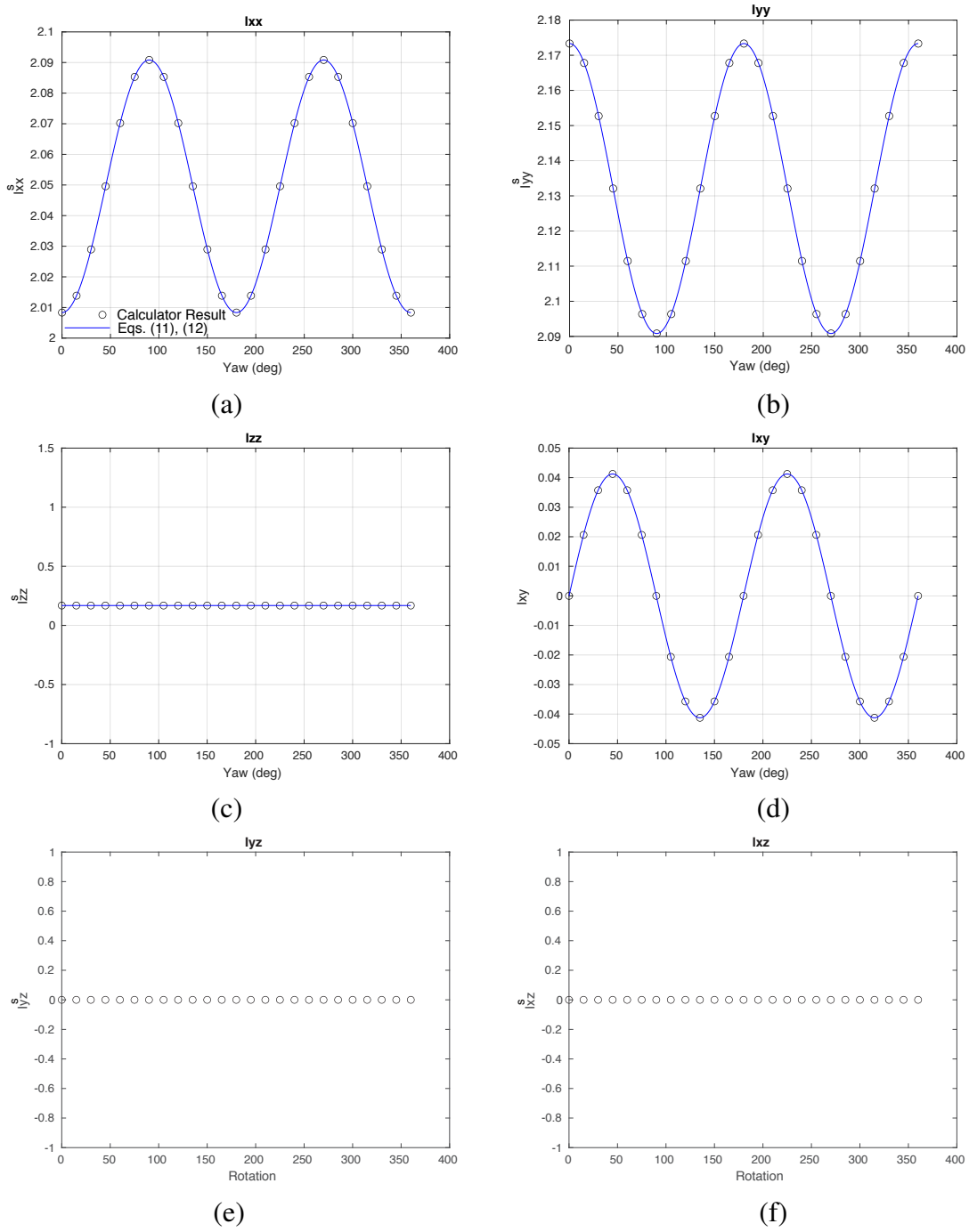

Figure 7-3. Two-bar verification problem results: yaw

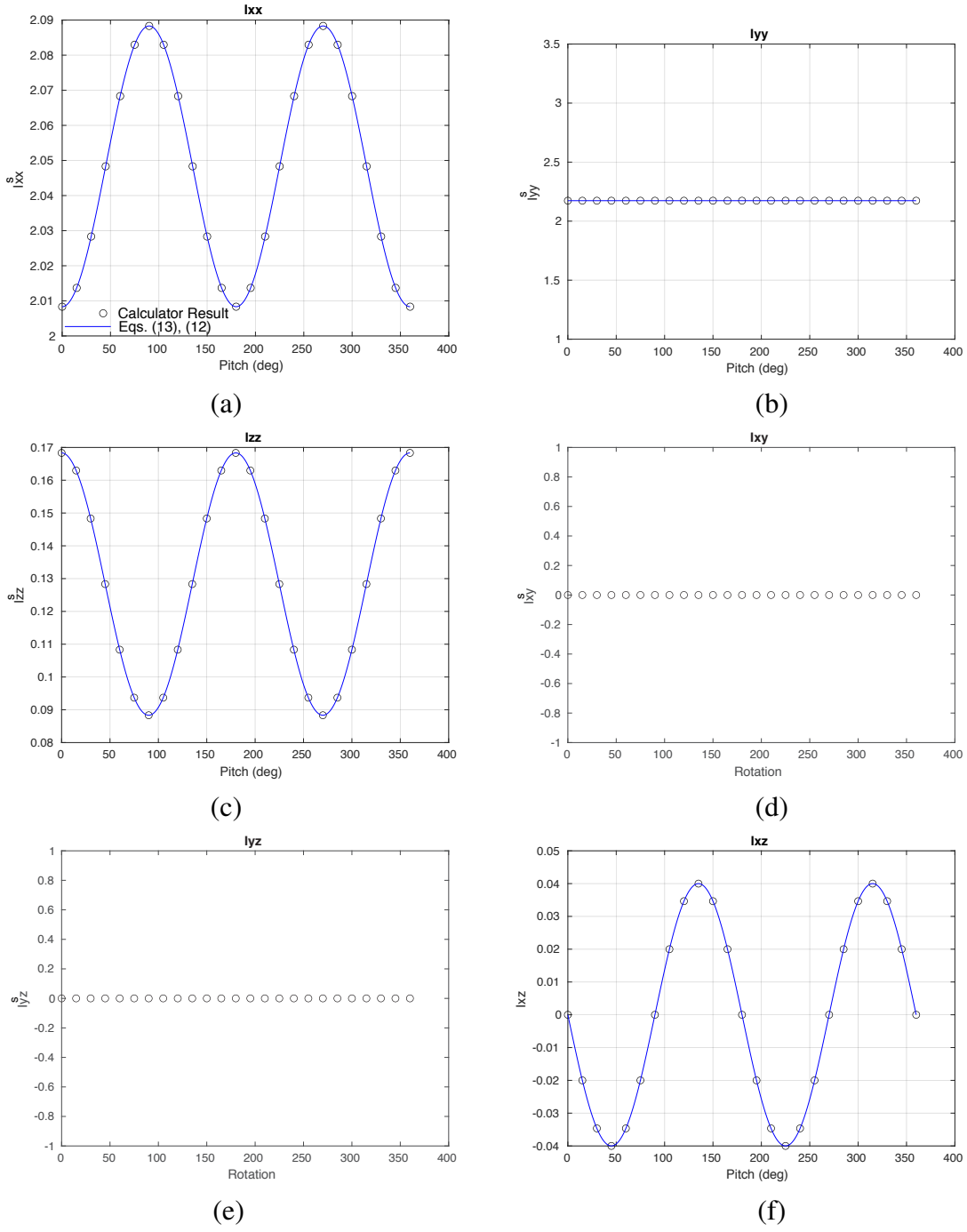

Figure 7-4. Two-bar verification problem results: pitch

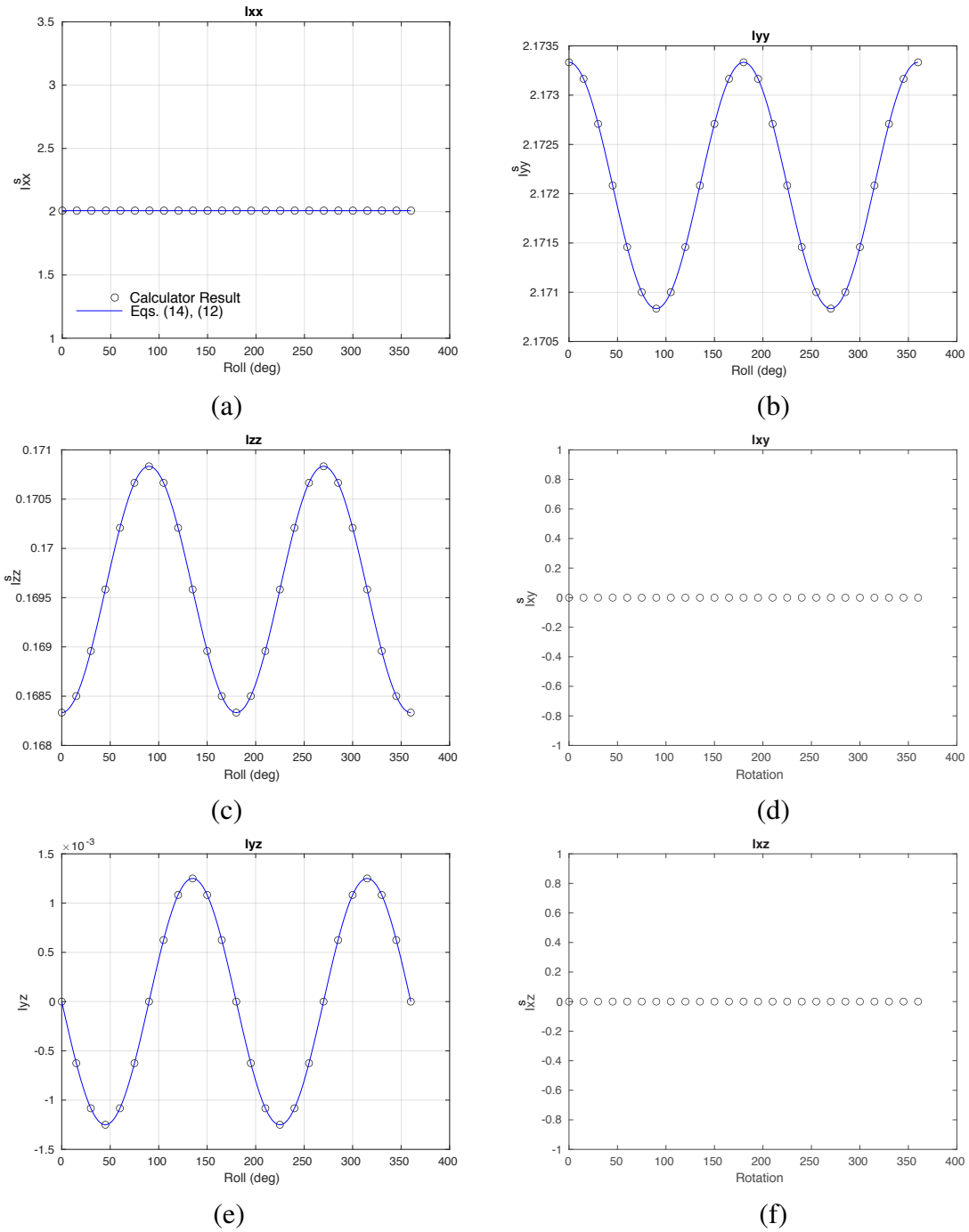

Figure 7-5. Two-bar verification problem results: roll

7.4. Problem 4: System Trimming. Spreadsheet Input

This example considers the system shown in Fig. 7-6. It consists of a cylindrical shell with outer diameter of 12 in., inner diameter of 11.5 in. and length of 20 in. It is closed at the two ends with circular caps that fit in the inner diameter of the shell. The caps also have thickness of 0.25 in. The shell contains two rods as shown in the figure, located somewhat arbitrarily to induce asymmetries to the problem. The density of the cylindrical shell, the cap and the rod material is 0.1 lb/in^3. A sphere of mass 1 lb is located inside the shell. Its location is to be determined as the system is trimmed to achieve a certain objective.

Here we consider an example where the location of the sphere has to be determined such that the centroid of the system is located as close to the center of the container as possible. So, taking the origin of the coordinate axes at the center of the left disk, on the outside surface, the objective function to be minimized is

$$f = (x_c - 10)^2 + y_c^2 + z_c^2. \tag{7.17}$$

where $\{x_c, y_c, z_c\}$ are the coordinates of the system centroid by choosing the appropriate location of the sphere.

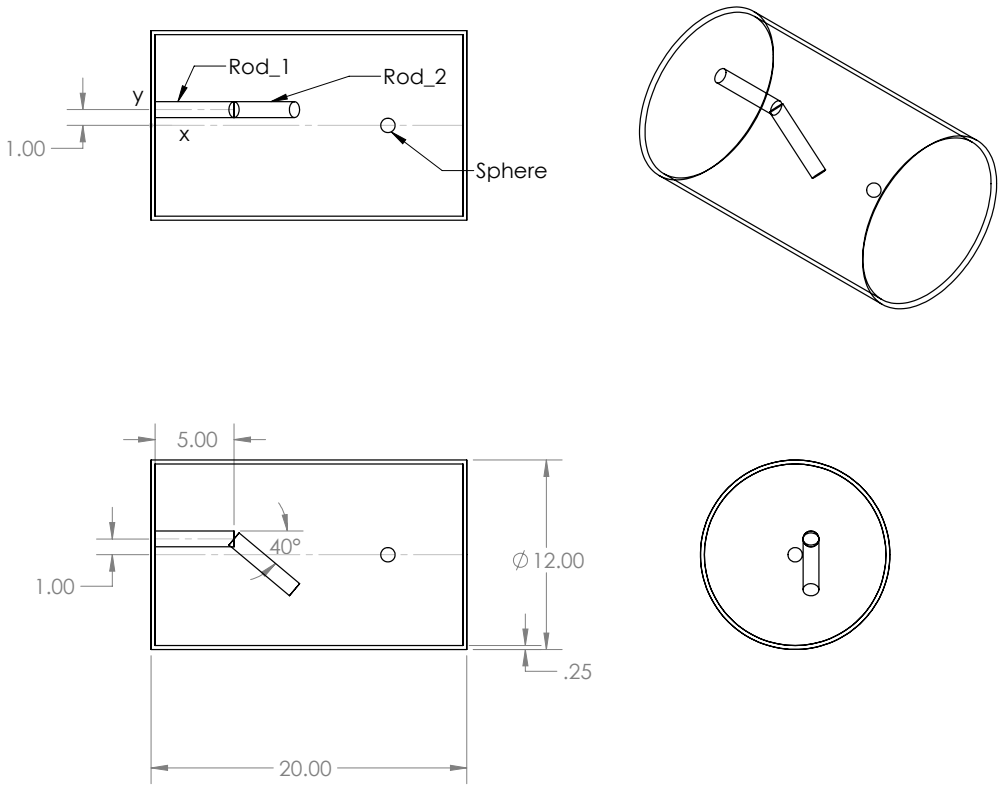

Figure 7-6. Optimization problem.

The input spreadsheet is shown in Fig. 7-7. It defines 6 parts: The cylindrical shell, the two caps, the two rods and the sphere. All parts are defined with the local x axis in the same direction as the system x axis. Therefore the only part that needs to be re-oriented is `Rod_2`. Note that the centroid of the sphere is immaterial since it will be changed during the optimization.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	C
1	name	mass	centroidX	centroidY	centroidZ	Ixx	Iyy	Izz	Ixy	Iyz	Ixz	T1	T2	T3	
2	Cylindrical Shell	18.4569	10	0	0	637.338	933.898	933.898	0	0	0	0	0	0	
3	Cap_1	2.59672	0.125	0	0	42.9271	21.4771	21.4771	0	0	0	0	0	0	
4	Cap_2	2.59672	19.875	0	0	42.9271	21.4771	21.4771	0	0	0	0	0	0	
5	Rod_1	0.3927	2.75	1	1	0.04909	0.84267	0.84267	0	0	0	0	0	0	
6	Rod_2	0.3927	7.165111	1	-0.606969	0.04909	0.84267	0.84267	0	0	0	0	-40	0	
7	Sphere	1	10	0	0	0.7145	0.7145	0.7145	0	0	0	0	0	0	
8															

Figure 7-7. Input spreadsheet.

The user needs to write the wrapper to call the constrained optimization routine `fmincon` as well as the functions that the optimization function will be calling to evaluate the objective function. The wrapper has the form below:

```
clc
clear all
close all

addpath...
('/ascldap/users/ecorona/MassProperties2021/NewCalculator/2020_07_23_Mass_Properties_Calc/Core')

global Part_Name Inp_Param Out_Param inp_name

inp_name = 'Optimization_Problem.xlsx';
%Defines the name of the spreadsheet.

Part_Name = 'Sphere';
%Defines what number mass this is in the spreadsheet.

Inp_Param = ["centroidX";"centroidY";"centroidZ"];
%Defines what parameter will be iterated through.

Out_Param = ["centroidX";"centroidY";"centroidZ"];
%Defines the parameter that is minimized.
%
% Constrained optimization
options = optimoptions('fmincon','Display','iter','PlotFcn',@optimplotfval)
Xx = fmincon(@sphere_opt_driver,[18,0,0],[],[],[],[],[0.955,-5.296,-5.296],...
    [19.045,5.296,5.296],@circlecon,options)
% %
```

The function of the wrapper is to define the input spreadsheet, the name of the part to be moved, the parameters to be varied and the parameters in the output needed to construct the objective function. Finally, the script calls the optimization function which itself will call a function `sphere_opt_driver` to evaluate the objective function as well as a constraint function `circlecon` to help constrain the search to locations inside the shell. The function that evaluates the objective function is as follows:

```
function Y = sphere_opt_driver(X)
global Part_Name Inp_Param Out_Param inp_name

loaded = readtable(inp_name);
mass_num = find(strcmp(Part_Name,loaded.name));

for i = 1:length(Inp_Param)
    loaded.(Inp_Param(i))(mass_num) = X(i);
end
```

```
writetable(loaded,inp_name);

out_name = driver_fun(inp_name);

output = readtable(out_name);

Y = sqrt((output.(Out_Param(1)) - 10.)^2 + (output.(Out_Param(2)) - 0)^2 ...
         + (output.(Out_Param(3)) - 0)^2);
```

Note that this function reads the input spreadsheet, and then it overwrites it with the current guess from the optimizer (`X`). It then calls `driver_fun` which is the driver function for the calculator, shown below, reads the output spreadsheet and calculates the objective function (`Y`).

The driver function is very similar to the ones shown before. It uses the name of the input spreadsheet and adds the prefix 'System_' to define the name of the output spreadsheet:

```
function out_name = driver_fun(inp_name)

part = loading_parts(inp_name);
% Loads the spreadsheet and puts it into a structured array containing all the parts.

[system(1)] = deal(cg_calc(part));
% Locates the cg of the system, and total mass of the system.

[part] = coordinate_alignment(part);
% Calculates inertia matrix with respect to axes directions of system using part orientation

[part] = deal(parallel_axis(part,system));
% Transforms the MOI's of each part to the system cg location.

[system] = deal(inertia_integration(part,system));
% Sums the inertia for the entire system.

[system] = principals(system);
% Calculates the principal MOI for the system, and the principal directions.

[system] = pai(system);
% Calculates the largest angle from principal axis to system axis

out_name = append('System_',inp_name);
% Names output spreadsheet

spread_gen(out_name,system) % Generates a Excel output of the System Properties with name specified.
```

Finally the nonlinear constraint function is

```
function [c,ceq] = circelcon(x)
% Constrain search in cylinder cross-section to the inside.
% Edmundo Corona 04/23/2021
c = x(2)^2 + x(3)^2 - 5.296^2;
ceq = [];
```

The final result can be found in the input spreadsheet once the optimization function declares convergence as shown in Fig. 7-8. Note that the final location of the cg of the sphere is now listed. The properties of the system is in the system spreadsheet shown in Fig. 7-9. Note that the location of the system centroid is listed as (10,0,0).

On a separate note, none of the previous examples tested for the use of the parallel axes theorem for the products of inertia. This example does. The values shown in Fig. 7-9 were re-calculated in a spreadsheet and gave the same values, thus verifying the calculations.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	name	mass	centroidX	centroidY	centroidZ	Ixx	Iyy	Izz	Ixy	Iyz	Ixz	T1	T2	T3
2	Cylindrical Shell	18.4569	10	0	0	637.338	933.898	933.898	0	0	0	0	0	0
3	Cap_1	2.59672	0.125	0	0	42.9271	21.4771	21.4771	0	0	0	0	0	0
4	Cap_2	2.59672	19.875	0	0	42.9271	21.4771	21.4771	0	0	0	0	0	0
5	Rod_1	0.3927	2.75	1	1	0.04909	0.84267	0.84267	0	0	0	0	0	0
6	Rod_2	0.3927	7.165111	1	-0.606969	0.04909	0.84267	0.84267	0	0	0	0	-40	0
7	Sphere	1	13.96033	-0.785398	-0.154343	0.7145	0.7145	0.7145	0	0	0	0	0	0
8														

Figure 7-8. Spreadsheet with updated location for the centroid of the sphere.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	mass	centroid x	centroid y	centroid z	Ixx	Iyy	Izz	Ixy	Iyz	Ixz	P1	P2	P3	n11	n21	n31	n12	n22	n32	n13	n23	n33	POI
2	25.436	10	2.687E-10	2.78E-10	726.296	1525.74	1526.25	7.07076	-0.27556	3.17336	1526.37	1525.69	726.221	0.00012	-0.39794	0.91741	0.00969	0.91737	0.39792	-1	0.00884	0.00397	0.55547

Figure 7-9. Spreadsheet with updated system properties.

7.5. Problem 5: System Trimming. Creo Mass Properties File

Users may want to create optimization problems using only Creo mass property text files, or in combination with manual spreadsheet inputs. This can be easily achieved with the functions detailed in previous sections. To give the user a baseline model to follow, and to ensure proper operation an example problem was created. This problem consists of the same rod assembly, cylindrical shell, and sphere used previously. The program reads in one Creo assembly mass property text file for the assembly of the two circular bars, one Creo mass properties text file for the cylindrical shell, and finally a spreadsheet input for the sphere.

The two Creo text files are as follows. For the rod assembly:

```
VOLUME =  7.8539816e+00  INCH^3
SURFACE AREA =  3.4557519e+01  INCH^2
AVERAGE DENSITY =  1.0000000e-01 POUND / INCH^3
MASS =  7.8539816e-01 POUND

CENTER OF GRAVITY with respect to _BLOCK_ASSEM coordinate frame:
X   Y   Z     1.9651549e-01  4.7075556e+00  1.0000000e+00  INCH

INERTIA with respect to _BLOCK_ASSEM coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  2.3375597e+01  1.0572733e+00 -1.5434276e-01
Iyx Iyy Iyz  1.0572733e+00  1.7488347e+00 -3.6973054e+00
Izx Izy Izz -1.5434276e-01 -3.6973054e+00  2.3455460e+01

INERTIA at CENTER OF GRAVITY with respect to _BLOCK_ASSEM coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  5.1849278e+00  1.7838511e+00  0.0000000e+00
Iyx Iyy Iyz  1.7838511e+00  9.3310580e-01  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  6.0198587e+00
```

```
PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   2.8383709e-01  5.8341965e+00  6.0198587e+00

ROTATION MATRIX from _BLOCK_ASSEM orientation to PRINCIPAL AXES:
      -0.34202       -0.93969        0.00000
       0.93969       -0.34202        0.00000
       0.00000        0.00000        1.00000

ROTATION ANGLES from _BLOCK_ASSEM orientation to PRINCIPAL AXES (degrees):
angles about x  y  z   0.000          0.000        110.000

RADII OF GYRATION with respect to PRINCIPAL AXES:
R1  R2  R3 6.0115938e-01  2.7254962e+00  2.7685235e+00  INCH

-----------------------------------------------

MASS PROPERTIES OF COMPONENTS OF THE ASSEMBLY
(in assembly units and the _BLOCK_ASSEM coordinate frame)

   DENSITY             MASS          C.G.:  X        Y         Z

                          ROD_1            MATERIAL:                                UNKNOWN
         1.00000e-01  3.92699e-01  1.00000e+00  2.50000e+00  1.00000e+00
                          ROD_1            MATERIAL:                                UNKNOWN
         1.00000e-01  3.92699e-01 -6.06969e-01  6.91511e+00  1.00000e+00
-----------------------------------------------
MASS PROPERTIES OF THE PART ROD_1
VOLUME =  3.9269908e+00  INCH^3
SURFACE AREA =  1.7278760e+01  INCH^2
DENSITY =  1.0000000e-01 POUND / INCH^3
MASS =  3.9269908e-01 POUND

CENTER OF GRAVITY with respect to _ROD_1 coordinate frame:
X   Y   Z     0.0000000e+00  2.5000000e+00  0.0000000e+00  INCH

INERTIA at CENTER OF GRAVITY with respect to _ROD_1 coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  8.4266681e-01  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  4.9087385e-02  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  8.4266675e-01

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   4.9087385e-02  8.4266670e-01  8.4266686e-01

CENTER OF GRAVITY with respect to _BLOCK_ASSEM coordinate frame:
X   Y   Z     1.0000000e+00  2.5000000e+00  1.0000000e+00  INCH

INERTIA at CENTER OF GRAVITY with respect to _BLOCK_ASSEM coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  8.4266681e-01  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  4.9087385e-02  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  8.4266675e-01

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   4.9087385e-02  8.4266670e-01  8.4266686e-01

-----------------------------------------------
MASS PROPERTIES OF THE PART ROD_1
VOLUME =  3.9269908e+00  INCH^3
SURFACE AREA =  1.7278760e+01  INCH^2
DENSITY =  1.0000000e-01 POUND / INCH^3
MASS =  3.9269908e-01 POUND

CENTER OF GRAVITY with respect to _ROD_1 coordinate frame:
X   Y   Z     0.0000000e+00  2.5000000e+00  0.0000000e+00  INCH
```

```
INERTIA at CENTER OF GRAVITY with respect to _ROD_1 coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  8.4266681e-01  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  4.9087385e-02  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  8.4266675e-01

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   4.9087385e-02  8.4266670e-01  8.4266686e-01

CENTER OF GRAVITY with respect to _BLOCK_ASSEM coordinate frame:
X   Y   Z    -6.0696902e-01  6.9151111e+00  1.0000000e+00  INCH

INERTIA at CENTER OF GRAVITY with respect to _BLOCK_ASSEM coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  5.1477891e-01  3.9076158e-01  0.0000000e+00
Iyx Iyy Iyz  3.9076158e-01  3.7697529e-01  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  8.4266675e-01

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   4.9087385e-02  8.4266670e-01  8.4266686e-01
```

For the cylindrical shell:

```
VOLUME =  2.3650302e+02  INCH^3
SURFACE AREA =  1.8924169e+03  INCH^2
DENSITY =  1.0000000e-01 POUND / INCH^3
MASS =  2.3650302e+01 POUND

CENTER OF GRAVITY with respect to _CYLINDER coordinate frame:
X   Y   Z     0.0000000e+00 -9.7500000e+00  0.0000000e+00  INCH

INERTIA with respect to _CYLINDER coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  3.7315508e+03  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  7.2319248e+02  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  3.7315508e+03

INERTIA at CENTER OF GRAVITY with respect to _CYLINDER coordinate frame:  (POUND * INCH^2)

INERTIA TENSOR:
Ixx Ixy Ixz  1.4832939e+03  0.0000000e+00  0.0000000e+00
Iyx Iyy Iyz  0.0000000e+00  7.2319248e+02  0.0000000e+00
Izx Izy Izz  0.0000000e+00  0.0000000e+00  1.4832939e+03

PRINCIPAL MOMENTS OF INERTIA:  (POUND * INCH^2)
I1  I2  I3   7.2319248e+02  1.4832936e+03  1.4832943e+03

ROTATION MATRIX from _CYLINDER orientation to PRINCIPAL AXES:
       0.00000        0.00000        1.00000
       1.00000        0.00000        0.00000
       0.00000        1.00000        0.00000

ROTATION ANGLES from _CYLINDER orientation to PRINCIPAL AXES (degrees):
angles about x  y  z   0.000         90.000         90.000

RADII OF GYRATION with respect to PRINCIPAL AXES:
R1  R2  R3 5.5297896e+00  7.9194534e+00  7.9194553e+00  INCH
```

Then a typical spreadsheet input was created for the sphere as can be seen in Figure 7-10.

name	mass	centroidX	centroidY	centroidZ	Ixx	Iyy	Izz	Ixy	Iyz	Ixz	T1	T2	T3
Sphere	1	0	0	0	0.7145	0.7145	0.7145	0	0	0	0	0	0

Figure 7-10. Input Parameters for Sphere Object

To compile these inputs an additonal spreadsheet was created. The Creo models were created with a different reference frame than that of the final system. The cylinder had its local axis in the *y* direction while the bar assembly was defined in the *x*-*y* plane, with its length along the *y* axis. Therefore both parts need to be rotated to agree with the orientations shown in Fig. 7-6. Additionally the centroid locations in relation to the final system must be defined. The definitions can be seen in Figure 7-11.

File Name	Creo/Spreadsheet	Part Name	Centroid X	Centroid Y	Centroid Z	T1	T2	T3
rod_assembly.txt	Creo	Rods	4.957555554	1	0.19651549	90	0	90
cylinder.txt	Creo	Cylinder	10	0	0	90	0	0
sphere.csv	Spreadsheet	Spheres	10	0	0	0	0	0

Figure 7-11. Example Input for Creo Optimization

These can then be read into the MATLAB optimization script once a total part array is constructed. The objective of this study was to identify the ideal location of the sphere to minimize the L2 (Euclidian) norm of the system's cg coordinates. The results of this optimization are given in Figs. 7-12 and 7-13.

name	mass	centroidX	centroidY	centroidZ	Ixx	Iyy	Izz	Ixy	Iyz	Ixz	T1	T2	T3
Rods	0.785398	4.957556	1	0.19651549	5.184928	0.933106	6.019859	1.783851	0	0	90	0	90
Cylinder	23.6503	10	0	0	1483.294	723.1925	1483.294	0	0	0	90	0	0
Spheres	1	13.96033	-0.785398148	-0.15434289	0.7145	0.7145	0.7145	0	0	0	0	0	0

Figure 7-12. Output spreadsheet with the final location of the sphere.

mass	centroid x	centroid y	centroid z	Ixx	Iyy	Izz	Ixy	Iyz	Ixz	P1	P2	P3	n11	n21	n31	n12	n22	n32	n13	n23	n33	POI
25.4357	10	4.75E-10	3.52E-10	726.2965	1525.736	1526.249	7.07076	-0.27556	3.173365	1526.369	1525.692	726.2213	0.000122	-0.39794	0.917411	0.009694	0.917368	0.397921	-0.99995	0.008845	0.003969	0.555471

Figure 7-13. Spreadsheet with resulting system properties.

These results agree with those in the previous section.

7.6. Problem 6: System trimming. Phoenix Model Center

Users of the calculator may want to conduct an optimization problem within Phoenix Model Center. To show this functionality, which includes reading the Creo files directly plus a manual Excel input, the trimming example in the previous two section is repeated here. This process reads in one Creo assembly for two circular bars, one Creo part of the cylindrical shell, and finally an Excel spreadsheet for the sphere. Note that the Creo files must include the information required as described in Section 6.3.1. The Model Center flow can be seen in Figure 7-14.

All of this is then interfaced with the MATLAB Model Center connection script as detailed in Section 6.8 to calculate the mass properties of the system. To constrain this optimization, limits

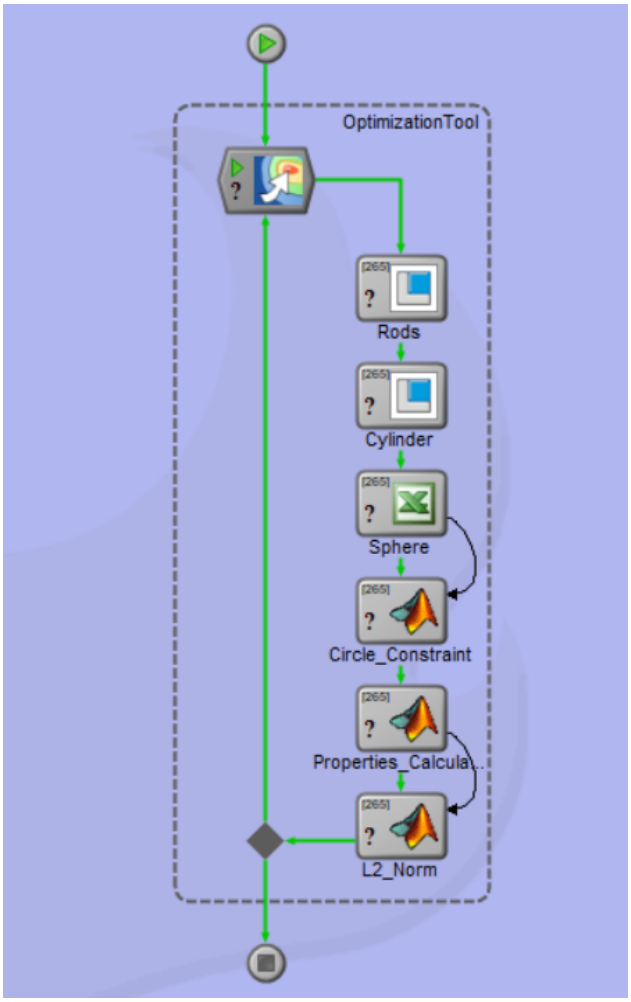

Figure 7-14. Model Center Optimization Flow

were set for the three coordinates of the sphere's centroid within Model Center's optimization options as can be seen in Figure 7-15.

A nonlinear constraint to keep the sphere within the circular cross section of the cylinder was also added using the simple MATLAB script where L2 and L3 are the x and y coordinates of the sphere:

```
%variable: L2 double input
%variable: L3 double input
%variable: Circ double output

Circ = (L2)^2 + (L3)^2
```

For this constraint to be valid the variable Circ must be less than the radius of the cross section squared. Additionally, to simplify the optimization process, a Euclidian (L2) norm was used as the goal function rather than the minimum of each centroidal coordinate. This is calculated with another MATLAB script given:

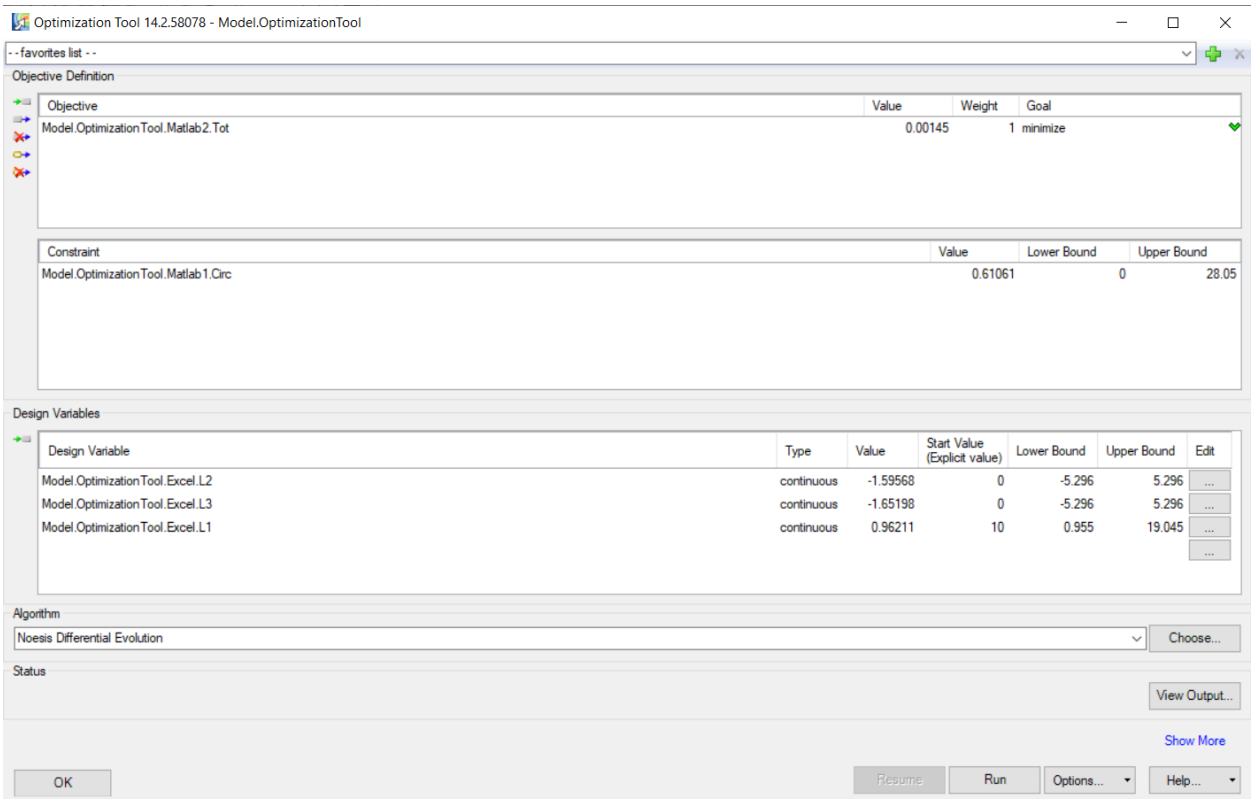

Figure 7-15. Model Center Optimizer Constraints

```
%variable: Sys_X double input
%variable: Sys_Y double input
%variable: Sys_Z double input
%variable: Tot double output

Tot =  sqrt((Sys_X-10)^2 + Sys_Y^2 + Sys_Z^2);
```

Where Sys_X, Sys_Y and Sys_Z are the centroid coordinates for the system. The optimization algorithm chosen was the Noesis Differential Evolution with default parameters due to its use in the Model Center Help Guide. This optimization ran for 301 iterations, finally yielding the optimal position of the sphere and system properties, as given in Figures 7-16 and 7-17.

Note that this optimization procedure found an answer close to that of the previous cases, although some differences can be appreciated. A more experienced user of Model Center, could perhaps set the parameters of the optimizer, or choose a different optimizer that could yield an answer that is closer to the previous ones.

	Rods	Cylinder	Sphere
Mass	0.785398	23.6503	1
L1	4.9576	10	13.89712
L2	1	0	-0.76732
L3	0.19652	0	-0.16428
Ixx	5.184928	1483.294	0.7145
Iyy	0.933106	723.1925	0.7145
Izz	6.019859	1483.294	0.7145
Ixy	1.783851	0	0
Ixz	0	0	0
Iyz	0	0	0
T1	90	90	0
T2	0	0	0
T3	90	90	0

Figure 7-16. Model Center Optimizer Results

Mass	25.4357
L1	9.997516
L2	0.000711
L3	-0.00039
Ixx	726.2716
Iyy	1525.242
Izz	1525.724
Ixy	6.950576
Ixz	3.202355
Iyz	-0.28041
P1	1525.853
P2	1525.187
P3	726.1983
PAI	0.54882

Figure 7-17. Model Center Optimizer Results

APPENDIX A. Listing of Functions

The function listings that follow are as of the time of publication of this report. The latest versions of the functions and the latest update of this document can be found in the Git repository below.

https://cee-gitlab.sandia.gov/cjfiete/mass-properties-calculator

A.1. Sample Calculator Driver

```
 clc
clear

addpath('/ascldap/users/ecorona/MassProperties2021/NewCalculator/2020_07_23_Mass_Properties_Calc/Core')

name = 'oneBodyPart2.xlsx'; % Name (and path) of the part spreadsheet
        % that will be loaded.

part = loading_parts(name); % Loads the spreadsheet and puts it into a
        % structured array containing all the parts.

[system(1)] = deal(cg_calc(part)); % Locates the cg of the system, and
        % total mass of the system.

[part] = coordinate_alignment(part); % Calculates inertia matrix with
         % respect to axes directions of system using part orientation

[part] = deal(parallel_axis(part,system)); %Transforms the MOI's of each
         % part to the system cg location.


[system] = deal(inertia_integration(part,system)); %This sums the inertia
         % for the entire system.

[system] = principals(system); %This calculates the principal MOI for
         % the system, and the principal directions.

[system] =     pai(system); %Calculates the angle between the
        % closest eigenvector to the system x-axis and the x-axis

out_name = append('System_',name); % Names output spreadsheet

spread_gen(out_name,system) % Generates a Excel output of the System Properties with name specified.
```

A.2. Core Functions

A.2.1. `loading_parts`

```
  %This function is for reading in a spreadsheet of parts with the format of
% name | mass | centroid x | y | z | Ixx | Iyy | Izz | Ixy | Iyz | Ixz |
% | T1 | T2 | T3 |
```

```
%Written by Carter Fietek 3/29/2021
%
% Input: This function simply needs the name of the file (path) to import.
%
%Output: This will output a structured array of parts with the name of the
%part, mass, centroid position (coordinates), inertia tensor, and
%orientation vector. These can be accessed by dot indexing ex: part(1).name
%
function [part] = loading_parts(name)
loaded = readtable(name);
[x,y] = size(loaded);

part = struct('name',{},'mass',{},'centroid',{},'inertia',{},'orientation',{});

for i = 1:x
    [part(i).name] = deal(loaded.name{i});
    [part(i).mass] = deal(loaded.mass(i));
    r = [loaded.centroidX(i),loaded.centroidY(i),loaded.centroidZ(i)];
    [part(i).centroid] = r;
    I = [loaded.Ixx(i),loaded.Ixy(i),loaded.Ixz(i);loaded.Ixy(i),loaded.Iyy(i),loaded.Iyz(i);loaded.Ixz(i),loaded
    [part(i).inertia] = I;
    T = [loaded.T1(i),loaded.T2(i),loaded.T3(i)];
    [part(i).orientation] = T;
end
```

A.2.2. cg_calc

```
  %This function is a function that imports the mass and centroid location of
%each part, and then calculates the mass and centroid of the system.
%Written by Carter Fietek 3/26/2021
%
%Input: This requires a structured array of all parts within the system,
%with a term for the mass and centroid of each part.
%
%Output: This will output a new structured array of the combined system,
%including terms for total mass, and for the system centroid.
%
function [system] = cg_calc(part)
system = struct('mass',{},'centroid',{});

system(1).mass = 0;
for i = 1:length(part)
    system(1).mass = part(i).mass + system(1).mass;
end

system(1).centroid = [0,0,0];
for i = 1:length(part)
    system(1).centroid = system(1).centroid + (part(i).centroid*part(i).mass)/system(1).mass;
end
```

A.2.3. coordinate_alignment

```
 function[part] = coordinate_alignment(part)
%
% This function calculates the inertia matrix of each and all parts
% in the part list with respect to the system coordinate axes.
%
% It adds the variables part.transformation_matrix and
% part.transformed_part_inertia to the part structured array.
%
```

```
% Input:
%   part: the structured array containing all part definitions in the system
%
% Output:
%   part: the updated structured array with part.transformed_part_inertia
%         added.
%
% Written by Edmundo Corona 03/29/2021
%
%
number_of_parts = length(part);
for i = 1:number_of_parts
    part(i).transformation_matrix = deal(orientation(part(i).orientation));
    system_coords_inertia_matrix = inertia_transformation...
                         (part(i).inertia,part(i).transformation_matrix);
    part(i).transformed_part_inertia = deal(system_coords_inertia_matrix);
end
%
```

A.2.4. orientation

```
 function[transformation_matrix] = orientation(euler_angles)
% This function uses the Euler angles that define the orientation of a
% part to calculate the coordinate transformation matrix
%
% The Euler angles are defined as follows:
%   * When the part coordinates and the system coordinates are aligned, the
%     Euler angles are zero.
%   * The first Euler angle is the rotation about the z axis, or yaw.
%   * After the first rotation, the second Euler angle is the rotation
%     about the rotated y axis, or the pitch in the rotated configuration.
%   * After the second rotation, the third Euler angle is the rotation
%     about the rotated x axis, or the roll in the rotated configuration.
%
% Input:
%   euler_angles: 1-D array with yaw, pitch and roll Euler angles in degrees.
%
% Output:
%   transformation_matrix: The transformation matrix between the coordinate
%   systems.
%
% Written by Edmundo Corona 3/30/2021

deg_to_rad = pi/180.;
yaw = euler_angles(1)*deg_to_rad;
pitch = euler_angles(2)*deg_to_rad;
roll = euler_angles(3)*deg_to_rad;
% Yaw
cos_y = cos(yaw);
sin_y = sin(yaw);
D_y = [cos_y sin_y  0;
      -sin_y cos_y  0;
       0       0    1];
% Pitch
cos_p = cos(pitch);
sin_p = sin(pitch);
D_p = [cos_p  0 -sin_p;
         0    1    0 ;
       sin_p  0  cos_p];
% Roll
cos_r = cos(roll);
sin_r = sin(roll);
D_r = [1    0     0 ;
       0   cos_r sin_r;
       0  -sin_r cos_r];
```

```
% Calculate transformation matrix
transformation_matrix = D_r*D_p*D_y;
```

A.2.5. inertia_transformation

```
function[system_coords_inertia_matrix] = ...
 inertia_transformation(part_coords_inertia_matrix,transformation_matrix)
% This function transforms the moment of inertia matrix for a single part
% from the part coordinates to the system coordinates.
%
% Input:
%   part_coords_inertia_matrix: The inertia matrix in the part coordinates
%   transformation_matrix:  The coordinate transformation matrix from the
%      part to the system.
%
% Output:
%   system_coords_inertia_matrix: The inerta matrix for the part in the
%   system coordinates.
%
% Written by Edmundo Corona 3/30/2020
%
system_coords_inertia_matrix = ...
   transformation_matrix*part_coords_inertia_matrix*...
                     transpose(transformation_matrix) ;
```

A.2.6. parallel_axis

```
%This function will take the moments of inertia for each component in the
%a parallel coordinate system.
%and transform them to be centered about the system cg. Then these will be
%summed to obtain the total moment of inertia for the system.
%Written by Carter Fietek 3/26/2021
%
%Re-defined d to be more conventional. Modification does not affect the
%result
% Edmundo Corona 8/6/2021
%
% Input: This should be a structured array of parts including the
% coordinates of each part as a vector array contained in part.centroid,
% Additionally the centroid of the system is required, so the structured
% array containing the vector array of coordinates for the system is also
% required.
%
% Output: This will return the part array now with the additional term of
% system_cg_inertia which is the inertia tensor placed at the system cg.
%
function [part] = parallel_axis(part,system)

for i=1:length(part)
    % d = system(1).centroid - part(i).centroid;
    d = part(i).centroid - system(1).centroid;
    E = eye(3);
    [part(i).system_cg_inertia] = deal(part(i).transformed_part_inertia + part(i).mass*(dot(d,d)*E - d'*d));
end
```

A.2.7. inertia_integtration

```
%This Function takes all of the inertia tensors located at the system cg
%and sums them for the total system inertia tensor located at the system
```

```
%cg.
%
%Written by Carter Fietek 3/29/2021
%
%Input: The part, structured array will be used as input, specifically the
%part.system_cg_inertia will be pulled from the struct.
%
%Output: System structured array, now with total inertia tensor, contained
%in system.inertia
%
function [system] = inertia_integration(part,system)
total_inertia = 0;
for i = 1:length(part)
    total_inertia = part(i).system_cg_inertia + total_inertia;
end

[system(1).inertia] = total_inertia;
```

A.2.8. principals

```
function [system] = principals(system)
% This function calculates the eigenvalues and eigenvectors of a NxN matrix
% A and then orders them in a diagonal matrix so the largest is in position
% (1,1) and the smallest in position (N,N).  The corresponding eigenvectors
% are ordered so the ith column in the principal value matrix is the
% eigenvector that corresponds to the ith eigenvalue
%
% Input:
%   A: NxN matrix
%
% Output:
%   principal_values: Diagonal matrix with the eigenvalues in descending
%   order.
%   principal_directions: Matrix where the eigenvectors are ordered as the
%   columns of the matrix that correspond to the principal values
%
% Written by Edmundo Corona 03/30/2021
% Modified by Edmundo Corona to read from/write to system structure array 07/27/2021
%
A = system(1).inertia;
[principal_directions, principal_values]=eig(A);
[sorted_values,indx] = sort(diag(principal_values),'descend');
principal_values = principal_values(indx,indx);
system(1).principal = deal(principal_values);
principal_directions = principal_directions(:,indx);
system(1).principal_directions = deal(principal_directions);
```

A.2.9. pai

```
function [system] = pai(system)
%
% This function calculates the principal_inclination_angle defined as the
% angle between the vector with the largest direction cosine with respect
% to the x-axis and the x-axis.
%
% Input:
%   principal_directions: A matrix where the columns are the vectors of
%   interest.
%
% Output:
%   principal_inclination_angle: The angle between the vector with the
%   largest direction cosine with respect to the x-axis and the x-axis.  In
%   degrees.
```

```
% Written by Edmundo Corona 03/30/2021
% Modified by Edmundo Corona to read from/write directly to system array 07/27/2021
%
principal_directions = system(1).principal_directions;
max_direction_cosine = max(abs(principal_directions(1,:)));
principal_inclination_angle = acos(max_direction_cosine);
system(1).principal_angle_inclination = deal(principal_inclination_angle*180./pi);
```

A.3. Functions Specific to Reading Creo Files

A.3.1. Implementation Example

```
clc
clear
close all

name = 'Creo_Optimization_Prob.txt';

[part] = part_config(name);

for i=1:length(part)
    centroidX(i) = part(i).centroid(1);
    centroidY(i) = part(i).centroid(2);
    centroidZ(i) = part(i).centroid(3);
    Ixx(i) = part(i).inertia(1,1);
    Iyy(i) = part(i).inertia(2,2);
    Izz(i) = part(i).inertia(3,3);
    Ixy(i) = part(i).inertia(1,2);
    Iyz(i) = part(i).inertia(2,3);
    Ixz(i) = part(i).inertia(1,3);
    T1(i) = part(i).orientation(1);
    T2(i) = part(i).orientation(2);
    T3(i) = part(i).orientation(3);
end

input_table = table({part.name}',[part.mass]',centroidX',centroidY',centroidZ',...
Ixx',Iyy',Izz',Ixy',Ixz',Iyz',T1',T2',T3','VariableNames',...
 {'name','mass','centroid x','centroid y','centroid z',...
 'Ixx','Iyy','Izz','Ixy','Iyz','Ixz','T1','T2','T3'});

global Part_Name Inp_Param Out_Param inp_name part

inp_name = 'Creo_Optimization.csv';
%Defines the name of the spreadsheet.
Part_Name = 'Spheres';
%Defines what number mass this is in the spreadsheet.
Inp_Param = ["centroidX";"centroidY";"centroidZ"];
%Defines what parameter will be iterated through this can be a vector of parameters.
Out_Param = ["centroidX";"centroidY";"centroidZ"];
%Defines the parameter that is minimized. If multiple outputs need to be optimized,
%then the sum of squares will be used to scalerize parameters into one.

writetable(input_table,inp_name)

%Xx = fminsearch(@POI_opt_driver,1) %Unconstrained Optimizer
options = optimoptions('fmincon','Display','iter','PlotFcn',@optimplotfval)
XX = fmincon(@opt_driver,[1,0,0],[],[],[],[],[0.955,-5.296,-5.296],[19.045,5.296,5.296],...
@circlecon,options) %Constrained Optimizer where A*x <= b
```

A.3.2. Configure `part` Array from Creo Text Files. `part_config`

```
%Carter Fietek
```

```
%Written on 5/17 to unify the input of the Mass properties calculator.
%This function reads in a configuration file that is a tab delineated
%spreadsheet that will look for all parts. Of the form:
%| file name | CREO or Spreadsheet | Part or Assembly | Excluded Parts
%
function [part] = part_config(name)

config = readtable(name,'Delimiter','\t');
[x,y] = size(config);

cnt = 1;
for i = 1:x
    if strcmpi(config.Creo_Spreadsheet{i},'S') == 1 ||...
     strcmpi(config.Creo_Spreadsheet{i},'Spreadsheet') == 1
    %For the case of a spreadsheet with part properties.
        part1 = loading_parts(config.FileName{i});
            for j=1:length(part1)
                      if 1 == 1
                              part1(j).name = config.PartName(i);
                              part1(j).centroid(1) = config.CentroidX(i);
                              part1(j).centroid(2) = config.CentroidY(i);
                              part1(j).centroid(3) = config.CentroidZ(i);
                              part1(j).orientation(1) = config.T1(i);
                              part1(j).orientation(2) = config.T2(i);
                              part1(j).orientation(3) = config.T3(i);
                              part(cnt) = part1(j);
                              cnt = cnt +1;
                      else
                      end
            end
    else %For the case of a CREO Mass properties file
            part2 = CREO_Input(config.FileName{i});
                for j=1:length(part2)
                          if 1 == 1
                              part2(j).name = config.PartName(i);
                              part2(j).centroid(1) = config.CentroidX(i);
                              part2(j).centroid(2) = config.CentroidY(i);
                              part2(j).centroid(3) = config.CentroidZ(i);
                              part2(j).orientation(1) = config.T1(i);
                              part2(j).orientation(2) = config.T2(i);
                              part2(j).orientation(3) = config.T3(i);
                              part(cnt) = part2(j);
                              cnt = cnt +1;
                          else
                          end
                end
    end
end

%% Outputs all parts into a spreadsheet for future usage.

spread = struct('name',{},'mass',{},'centroidX',{},'centroidY',{},'centroidZ',{},...
'Ixx',{},'Iyy',{},'Izz',{},'Ixy',{},'Iyz',{},'Ixz',{},'T1',{},'T2',{},'T3',{});

[x,y] = size(part);

for i = 1:y
    [spread(i).name] = deal(part(i).name);
    [spread(i).mass] = deal(part(i).mass);
    spread(i).centroidX = part(i).centroid(1);
    spread(i).centroidY = part(i).centroid(2);
    spread(i).centroidZ = part(i).centroid(3);
    spread(i).Ixx = part(i).inertia(1,1);
    spread(i).Iyy = part(i).inertia(2,2);
    spread(i).Izz = part(i).inertia(3,3);
    spread(i).Ixy = part(i).inertia(1,2);
    spread(i).Iyz = part(i).inertia(2,3);
    spread(i).Ixz = part(i).inertia(1,3);
```

```
    spread(i).T1 = part(i).orientation(1);
    spread(i).T2 = part(i).orientation(2);
    spread(i).T3 = part(i).orientation(3);
end

spread = struct2table(spread);
writetable(spread,'Streamlined_Input.csv');
```

A.3.3. Read Creo Files. *CREO_input*

```
%This function will read in all the desired information from a CREO
%Part Mass properties file.
%Written by Carter Fietek 5/11/2021
function [part] = CREO_Input(name)

T = fileread(name);

%% Importing Part Names
Names = erase(name,'.txt');

%% Part Masses
Mass = regexp(T,'MASS =[\s\.=]+(\d+(\.\d+)?(e(+|-)?\d+))','match');
%Returns all of the cases for mass

for i=1:length(Mass)
    Mass{i} = erase(Mass{i},'MASS =  ');
    %Removes additional search characters
    Mass{i} = str2double(Mass{i});
    %Converts from string into double precision variable. Mass{1} corresponds to the system mass.
end

%% Part CG Locations
CG = regexp(T,'X   Y   Z[\s\.=]+(((\s|-)?\d+(\.\d+)?(e(+|-)?\d+))+\s((\s|-)?\d+(\.\d+)?...
(e(+|-)?\d+))+\s((\s|-)?\d+(\.\d+)?(e(+|-)?\d+)))','match');

for i=1:length(CG)
    CG{i} = erase(CG{i},'X   Y   Z    ');
    %Removes Additional Search Characters.
    CG{i} = str2num(CG{i});
    % Converts String Array of CG Locations into a number array. CG{1} is the system CG,
    % while even CG indices are in respect to part coordinate frames, and odd CG indices
    %  are in respect to assembly
end

%% Moments of Inertia

Ix = regexp(T,'Ixx Ixy Ixz[\s\.=]+(((\s|-)?\d+(\.\d+)?(e(+|-)?\d+))+\s((\s|-)?\d+(\.\d+)?...
(e(+|-)?\d+))+\s((\s|-)?\d+(\.\d+)?(e(+|-)?\d+)))','match');

for i=1:length(Ix)
    Ix{i} = erase(Ix{i},'Ixx Ixy Ixz ');
    Ix{i} = str2num(Ix{i});
end

Iy = regexp(T,'Iyx Iyy Iyz[\s\.=]+(((\s|-)?\d+(\.\d+)?(e(+|-)?\d+))+\s((\s|-)?\d+(\.\d+)?...
(e(+|-)?\d+))+\s((\s|-)?\d+(\.\d+)?(e(+|-)?\d+)))','match');

for i=1:length(Iy)
    Iy{i} = erase(Iy{i},'Iyx Iyy Iyz ');
    Iy{i} = str2num(Iy{i});
end

Iz = regexp(T,'Izx Izy Izz[\s\.=]+(((\s|-)?\d+(\.\d+)?(e(+|-)?\d+))+\s((\s|-)?\d+(\.\d+)?...
```

```
(e(+|-)?\d+))+\s((\s|-)?\d+(\.\d+)?(e(+|-)?\d+)))',’match’);

for i=1:length(Iz)
    Iz{i} = erase(Iz{i},’Izx Izy Izz ’);
    Iz{i} = str2num(Iz{i});
end

%% Putting into Structured Array as We Want
part = struct(’name’,{},’mass’,{},’centroid’,{},’inertia’,{},’orientation’,{});

    [part(1).name] = deal(Names);
    [part(1).mass] = deal(Mass{1});
    [part(1).centroid] = CG{1};
    I = [Ix{2};Iy{2};Iz{2}];
    [part(1).inertia] = I;
    [part(1).orientation] = zeros(1,3);
```

A.4. Functions Specific to Use with Model Center

A.4.1. Connect Parts in Model Center to Matlab

```
%Carter Fietek
%Script to connect loaded solidworks and creo parts in Model Center to MATLAB
%Calculator.
%
%variable: Mass_Properties double[] input
%variable: System_Properties double[] output

[x,y] = size(Mass_Properties);

part = struct(’name’,{},’mass’,{},’centroid’,{},’inertia’,{},’orientation’,{});

for i=1:y
    part(i).name = strcat("PART",num2str(i));
    part(i).mass = Mass_Properties(1,i);
    part(i).centroid = [Mass_Properties(2,i),Mass_Properties(3,i),Mass_Properties(4,i)];
    part(i).inertia = [Mass_Properties(5,i),Mass_Properties(8,i),Mass_Properties(9,i);...
     Mass_Properties(8,i),Mass_Properties(6,i),Mass_Properties(10,i);...
    Mass_Properties(9,i),Mass_Properties(10,i),Mass_Properties(7,i)];
    part(i).orientation = [Mass_Properties(11,i),Mass_Properties(12,i),Mass_Properties(13,i)];
end

[system] = deal(cg_calc(part));
% Locates the cg of the system, and total mass of the system.

[part] = coordinate_alignment(part);
% Calculates inertia matrix with respect to axes directions of system using part orientation

[part] = deal(parallel_axis(part,system));
%Transforms the MOI’s of each part to the system cg location.

[system] = deal(inertia_integration(part,system));
%This sums the inertia for the entire system.

[system] = principals(system);
%This calculates the principal MOI for the system, and the principal directions.

[system] = pai(system);
%Calculates the largest angle from principal axis to system axis

System_Properties = [system.mass; system.centroid(1);system.centroid(2);system.centroid(3);...
```

```
system.inertia(1,1);system.inertia(2,2);system.inertia(3,3);system.inertia(1,2);system.inertia(1,3);...
system.inertia(2,3);system.principal(1,1);system.principal(2,2);system.principal(3,3);...
system.principal_angle_inclination;];
```

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Technical Library	1911	sanddocs@sandia.gov

J. Gruda 0511
J. Bigelow 2453
A. Brundage 1554
S. Nelson 1558

Sandia National Laboratories

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.