Function Generation using Freudenstein's Equation

A presentation of using Freudenstein's Equation in the synthesis of four-bar linkages that will mechanically generate functions like $y = \sin x,\ y = e^x,\ y = \log_{10} x;\ y = x^\pi,\ y = \pi^x$, and almost anything else you can think of.

1 Freudenstein's Equation

The development begins with the **loop closure equation** for a four-bar linkage, as shown in text Figure 10.39, reproduced in Figure 1 below:

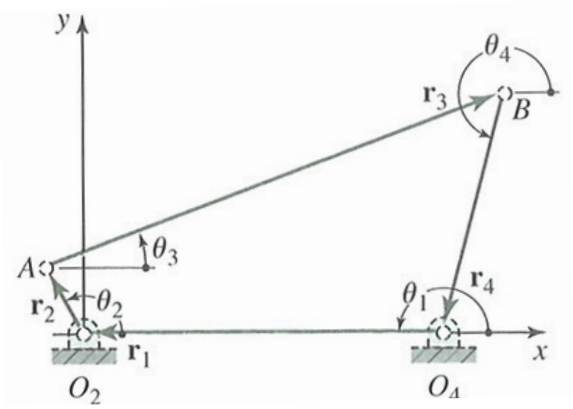


Figure 1: Loop closure diagram, showing $\mathbf{r}_i$ and θ_i for all four links.

1.1 Loop Closure Equation

The loop closure equation simply sums the position vectors around the complete four-bar linkage, and in vector form is given by

$$\mathbf{r}_1 + \mathbf{r}_2 + \mathbf{r}_3 + \mathbf{r}_4 = \mathbf{0}. \tag{1}$$

Each link has length r and is at angle θ, hence the complex form may be written as

$$r_1 e^{j\theta_1} + r_2 e^{j\theta_2} + r_3 e^{j\theta_3} + r_4 e^{j\theta_4} = 0 \tag{2}$$

We can expand (2) using Euler's identity, then separate Real and Imag terms. Before doing that, notice that the angle of link 1 is $\theta_1 = 180°$, thus $\cos\theta_1 = -1$ andd $\sin\theta_1 = 0$. Finally, for conciseness, use the short form

$$\cos\theta_i = c_i$$
$$\sin\theta_i = s_i$$

With these substitutions, equation (2) yields the two equations

$$-r_1 + r_2 c_2 + r_3 c_3 + r_4 c_4 = 0 \tag{3}$$
$$r_2 s_2 + r_3 s_3 + r_4 s_4 = 0 \tag{4}$$

1.2 Solving for the Link Lengths

Since the input of our mechanism will be link 2, and the output will be link 4, angles θ_2 and θ_4 should be preserved in the solution, but angle θ_3 should be eliminated.

Elimination of θ_3 is done by solving (3) and (4) for the θ_3 terms, then squaring and adding:

$$\begin{aligned} (r_3 c_3)^2 &= (r_1 - r_2 c_2 - r_4 c_4)^2 \\ +\,(r_3 s_3)^2 &= (-r_2 s_2 - r_4 s_4)^2 \end{aligned}$$

One gets a nice result on the left side, but the right is messy:

$$r_3^2 = f(r_1, r_2, r_4, \theta_2, \theta_4) \tag{5}$$

By using the trigonometric identity for $\cos(\theta_2 - \theta_4)$, the result can be written as

$$\boxed{K_1 c_2 + K_2 c_4 + K_3 = \cos(\theta_2 - \theta_4)} \tag{6}$$

where

$$\boxed{\begin{aligned} K_1 &= \frac{r_1}{r_4} \\ K_2 &= \frac{r_1}{r_2} \\ K_3 &= \frac{r_3^2 - r_1^2 - r_2^2 - r_4^2}{2 r_2 r_4} \end{aligned}} \tag{7, 8, 9}$$

If we have **three precision points** for the linkage, we will have **three** corresponding values for θ_2 and θ_4, and can hence write (6) **three times**. We will then have **three equations in three unknowns** and can solve for the three K_i.

Knowing the K_i then allows the calculation of the link lengths r_i. Actually we can only calculate three r_i...let $r_1 = 1$ and calculate the remaining three. As the entire linkage is scaled up or down it behaves the same.

2 Application of Freudenstein's Equation

Typically the designer will have in mind some function they wish to emulate, and a corresponding interval, such as

$$y = \frac{1}{x^2} \text{ over the range } 1 \le x \le 2 \tag{10}$$

First we must select the **three precision points**.

2.1 Precision Point Selection using Chebyshev Spacing

The **structural error** in a function generator is simply the error between the mathematical function and the actual mechanism, usually expressed as a percentage. A good choice of precision points will help reduce the structural error.

One good choice for the three precision points is using **Chebyshev Spacing**, which is simply a kind of equal spacing around a circle, then projection onto the horizontal bisector of the circle (see text Figure 10.27).

With Freudenstein's Equation we are limited to **three** precision points. We also have the bounds of the interval on x as given in (10).

2.1.1 Solution for Three Precision Points

Let the minimum and maximum values of independent variable x be called x_i and x_f. Our three precision points x_1, x_2, x_3 will fit between x_i and x_f; the sequence will be $\begin{bmatrix} x_i & x_1 & x_2 & x_3 & x_f \end{bmatrix}$.

Text equation (10.22) gives the Chebyshev solution for N points; expressed for 3 points using my notation it is

$$\boxed{x_j = \frac{1}{2}(x_f + x_i) - \frac{1}{2}(x_f - x_i)\cos\frac{(2j-1)\pi}{6},\ \ j = 1,\ 2,\ 3.} \tag{11}$$

For the example defined by (10), the three Chebyshev precision points I obtained are:

$$\begin{aligned} x_1 &= 1.0670 \\ x_2 &= 1.5000 \\ x_3 &= 1.9330 \end{aligned} \tag{12}$$

Note that with three points the middle point will **always** be in the center of the function interval. That is also the point we'll use for building the ADAMS/View model—covered later in Section 4.1.

2.2 Mapping Between $y = f(x)$ and angles θ_2 and θ_4

We cannot **directly** "fit" the mechanism to the function given in (10). For one thing, the ranges of the independent (x) and dependent (y) variables may be unsuitable for use as joint angles. For example, in the function of (10), are the extrema of x (1 and 2) to be interpreted as θ_2 in degrees? radians? What about θ_4? The extrema of y are 1 and 0.25; will these work as joint angles for θ_4?

We must solve for a **linear mapping** between the function $y = f(x)$ and the joint angles θ_2 and θ_4. Specifically, we must solve for scaling parameters $a,\ c$ and bias parameters $b,\ d$ in the linear mappings

$$\theta_2 = ax + b \tag{13}$$

$$\theta_4 = cy + d \tag{14}$$

2.2.1 Initial and Final Joint Angles (starting angles and swing angles)

To find the mapping between function values x and y and joint angles θ_2 and θ_4 we must select the **initial** and **final** values for θ_2 and θ_4. This will specify the configuration of the linkage at the extremes of motion. For our problems, selection of these angles is arbitrary, use common sense (some experience will help).

Define these initial and final joint angles as

$$\begin{bmatrix}\theta_{2i} & \theta_{2f} & \theta_{4i} & \theta_{4f}\end{bmatrix} \tag{15}$$

Units may be either degrees or radians; I will use degrees in my MATLAB function.

For the current example, I chose the following initial and final joint angles:

$$\theta_{2i}\ ...\ \theta_{2f} = 60^\circ\ ...\ 120^\circ \tag{16}$$

$$\theta_{4i}\ ...\ \theta_{4f} = 225^\circ\ ...\ 315^\circ \tag{17}$$

Thus I specified that link 2 has a travel of 60°, while that of link 4 is 90°. Both links 2 and 4 swing "symmetrically" through vertical. This all seemed reasonable to me, at least as a starting point.

2.2.2 Matrix Solution for Mapping Parameters

Consider the mapping of (13) for independent variable x and crank angle θ_2 (the mapping between y and θ_4 in (14) is handled similarly). Substitution of the extremum (max & min) values for both variables yields the two equations

$$ax_i + b = \theta_{2i} \tag{18}$$

$$ax_f + b = \theta_{2f} \tag{19}$$

Equations (18)–(19) can be arranged in matrix/vector form as

$$\begin{bmatrix} x_i & 1 \\ x_f & 1 \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \theta_{2i} \\ \theta_{2f} \end{bmatrix} \tag{20}$$

which is exactly in the form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ and $\mathbf{b}$ are known, and $\mathbf{x} = \begin{bmatrix} a & b \end{bmatrix}^T$ is the unknown. The solution is simply

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

Computing this is what MATLAB is made for, and the way to solve this is

```
>> x = A^{-1}*b;          % This is the "old" way
>> x = inv(A)*b;          % This is the "old" way
>> x = A\b;               % This is the "new" way (preferred)
```

Then of course the scaling parameter $a =$ `x(1)` and the bias parameter $b =$ `x(2)`.

As I indicated earlier, the procedure for dependent variable y and joint angle θ_4 is similar. You will, of course, get different scaling and bias parameters.

For the current example, I obtained the following values:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 60 \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} -120 \\ 345 \end{bmatrix} \tag{21}$$

2.2.3 Creation of "Precision Joint Angles"

With the mappings between function variable space (x, y) and mechanism joint space (θ_2, θ_4) known, we can map the three function **precision points** to corresponding **precision joint angles**.

From the x_i of (12) and the function in (10) you can compute the corresponding y_i. Then using the x_i and y_i you use the mappings of Section 2.2.2 to produce the corresponding **precision joint angles** θ_2 and θ_4. This is given in the text as Table 10.2 (p. 453).

For my example that table is shown below as Table 1. Note that the two joint angles move through their full range of motion.

Position	x	θ_2	y	θ_4
-	1	60°	1	225°
1	1.0670	64.0192°	0.8784	239.5946°
2	1.5000	90.0000°	0.4444	291.6667°
3	1.9330	115.9808°	0.2676	312.8847°
-	2	120°	0.25	315°

Table 1: Function values and precision point angles.

2.3 Finding the Link Lengths r_i

Now we are ready to find the four link lengths r_i, which will define the mechanism.

2.3.1 Matrix/Vector Formatting and the Solution for $K_1 ... K_3$

The first step is to reformat equation (6) using the three **precision point** values from Table 1.

For convenience, I will denote the angles θ_2 and θ_4 corresponding to precision point i (rows 1..3 in the table) as θ_{2i} and θ_{4i}, thus in this example we will have

$$\theta_{21} = 64.0192^\circ \qquad \theta_{41} = 239.5946^\circ \tag{22}$$

$$\theta_{22} = 90.0000^\circ \qquad \theta_{42} = 291.6667^\circ \tag{23}$$

$$\theta_{23} = 115.9808^\circ \qquad \theta_{43} = 312.8847^\circ \tag{24}$$

By substituting (22)—(24) into (6) we are doing a "three position synthesis" of this linkage. Note that we can't use the first and last rows of Table 1; we already used those to get the mapping (parameters a, b, c, d) between function space and joint space (Section 2.2).

The resulting matrix/vector equation is given by

$$\underbrace{\begin{bmatrix} c_{21} & c_{41} & 1 \\ c_{22} & c_{42} & 1 \\ c_{23} & c_{43} & 1 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \cos(\theta_{21} - \theta_{41}) \\ \cos(\theta_{22} - \theta_{42}) \\ \cos(\theta_{23} - \theta_{43}) \end{bmatrix}}_{\mathbf{b}} \tag{25}$$

where $c_{21} = \cos\theta_{21}$, $c_{42} = \cos\theta_{42}$, *etc.* In (25) quantities $\mathbf{A}$ and $\mathbf{b}$ are known, and the solution for vector $\mathbf{x}$ is

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}, \tag{26}$$

When programming this in MATLAB, the following syntax is preferred (as indicated earlier, I believe):

```
>> x = A^{-1}*b;          % This is the "old" way
>> x = inv(A)*b;          % This is the "old" way
>> x = A\b;               % This is the "new" way (preferred)
```

2.3.2 Finding r_i from K

With the three elements of $\mathbf{K}$ known, link lengths r_i are given by equations (7)—(9). Obviously, with only three equations we cannot solve for four quantities.

So just set $r_1 = 1$ and solve for r_2, r_3, and r_4. The kinematical relationship between θ_2 and θ_4 will be the same regardless of the overall **size** of the linkage, as long as the relationship between link lengths is preserved. So the final result can be scaled up or down to a convenient size for construction.

2.4 Next Steps

The two remaining sections are important, and will be presented in the following order:

- Constructing a model—this will probably require some iteration of the linkage synthesis; also required the construction of **measurement scales** for both links 2 and 4; this is effectively a mechanical implementation of the **joint space** $\leftrightarrow$ **function space** mapping of Section 2.2.
- Structural error—how accurate is the function generator? This will required construction of an ADAMS simulation, archiving of the result, and MATLAB analysis of these data to determine error.

3 Constructing a Model

Before constructing a model, we need an acceptable linkage design.

3.1 Design Iteration

In the problem I assign, you will be given the following:

- Function $y = f(x)$ and range $x_i \le x \le x_f$
- Required range of rotation for θ_2 and θ_4

You will be free to select both starting angles θ_{2i} and θ_{4i}. You will find that many starting angle selections will result in awkward values for r_i. It will be very valuable to be able to quickly compute the r_i, that's the reason for writing a MATLAB function to obtain the solution.

You will follow the flowchart shown below in Figure 2:

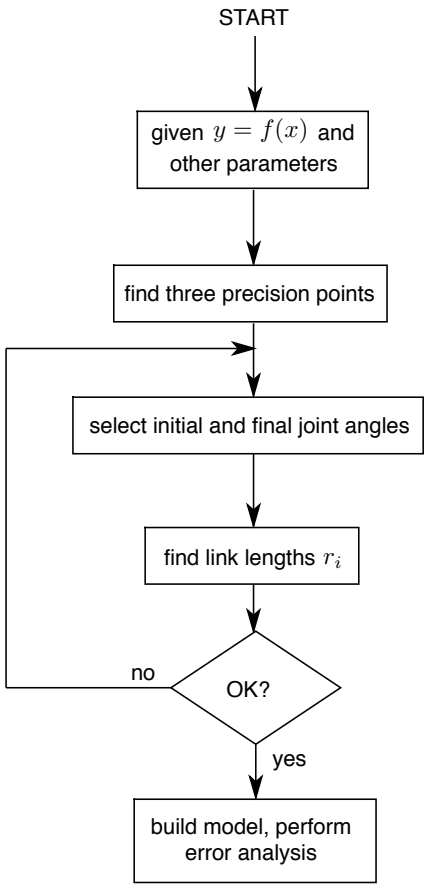


Figure 2: Flowchart for function generator synthesis.

3.2 An Example of Iteration

Following the previous work, I'll assume the following:

- Function is $y = \dfrac{1}{x^2}$ over the range $1 \leq x \leq 2$
- Input link 2 shall rotate 60°
- Output link 4 shall rotate 90°

I wrote MATLAB function `freud.m`, which is given as

```
>> help freud
  [r,c2,c4] = FREUD(fn,xi,xf,t2i,t2f,t4i,t4f)

  This function performs three-point kinematic synthesis of a four-bar
  mechanism using Freudenstein's Equation.  Function handle "fn" computes
  the mathematical function.  xi and xf specify the range of the
  independent variable, and Chebyshev spacing is used to determine the
  three precision points for the dependent variable.  Angles t2i, t2f, t4i,
  and t4f (DEG) specify the limits of motion of links 2 and 4.
  Returned 4x1 array r contains the link lengths, with r(1) = 1, while
  returned arrays c2 and c4 contain the scaling & bias parameters (like
  equation (21)) to relate [x y] space to [t2 t4] space to check the accuracy.
```

Initial Parameters. I'll begin by specifying initial angles such that both links swing their respective angles such that they are vertical at the midpoint. Since the swing angles are 60° and 90°, respectively, this results in initial and final joint angles of:

$$\theta_{2i} = 60^\circ \qquad \theta_{4i} = 225^\circ \tag{27}$$

$$\theta_{2f} = 120^\circ \qquad \theta_{4f} = 315^\circ \tag{28}$$

Here's how to define a "function handle" in MATLAB; the "dots" in the $1/x^2$ expression are to enforce an "element by element" evaluation (which is necessary inside the `freud.m` function):

```
>> fn = @(x) 1./x.^2;  % fn is now a "function handle" pointing to 1/x^2
>> xi = 1;             % starting point of interval
>> xf = 2;             % ending point of interval
```

Here's what happens with these values:

```
>> r = freud(fn,xi,xf,60,120,225,315)

r = 1.0000    5.9295    1.3220    5.4793
```

This doesn't look too great; links 2 & 4 are 5 inches long while the "ground" and coupler link 3 are around 1 inch long. I'll iterate using the starting position of link 2 and see what happens...

```
>> r = freud(fn,xi,xf,50,110,225,315)

r = 1.0000    3.2393    0.7870    3.2829  % These are better

>> r = freud(fn,xi,xf,40,100,225,315)

r = 1.0000    2.3482    0.6935    2.4094  % Better still

>> r = freud(fn,xi,xf,30,90,225,315)

r = 1.0000    1.9311    0.6824    1.9175  % Even better

>> r = freud(fn,xi,xf,20,80,225,315)

r = 1.0000    1.7182    0.7075    1.5855  % Still going

>> r = freud(fn,xi,xf,10,70,225,315)

r = 1.0000    1.6253    0.7624    1.3332  % Stop changing t2i now

>> r = freud(fn,xi,xf,10,70,240,330)  %  Now vary t4i

r = 1.0000    1.5010    1.0283    1.4411  % link 3 is a little longer...good

>> r = freud(fn,xi,xf,10,70,260,350)

r = 1.0000    1.3589    1.6715    1.9384  % Now we have fairly equal lengths.  Try this.
```

Looks to me like the last iteration gives reasonable link lengths: they're all similar magnitudes and link 2 is not as long as it began. So I'll construct this one. BTW, you can imagine how tedious this would be without a computer...

3.3 Actual Model Construction

I used corrugated cardboard for the "frame," and manila folder stock for the links. Thumbtacks served as joints. I measured as accurately as I could.

Oh, and I scaled the whole thing up by a factor of **THREE** so small construction errors are not so critical. Therefore the final link lengths are

$$\boxed{\begin{aligned} r_1 &= 3.0000 \text{ in} \\ r_2 &= 4.0767 \text{ in} \\ r_3 &= 5.0145 \text{ in} \\ r_4 &= 5.8152 \text{ in} \end{aligned}}$$

3.3.1 Basic Linkage

I sketched the linkage in both **INITIAL** and **FINAL** positions to get an idea of the overall size, then cut a "frame" from heavy cardboard (an HP printer box, actually).

After carefully locating points O_2 and O_4 on the cardboard, I measured and cut links 2, 3, and 4 from manila folder stock. Remember to leave a little extra length on links 2 and 4 for a "pointer." I used thumbtacks to assemble everything.

I should have taken a photo of it at this point (before I created the "input" and "output" scales, but I forgot. Here's a photo of the whole thing in Figure 3. The scales are discussed in the next section.

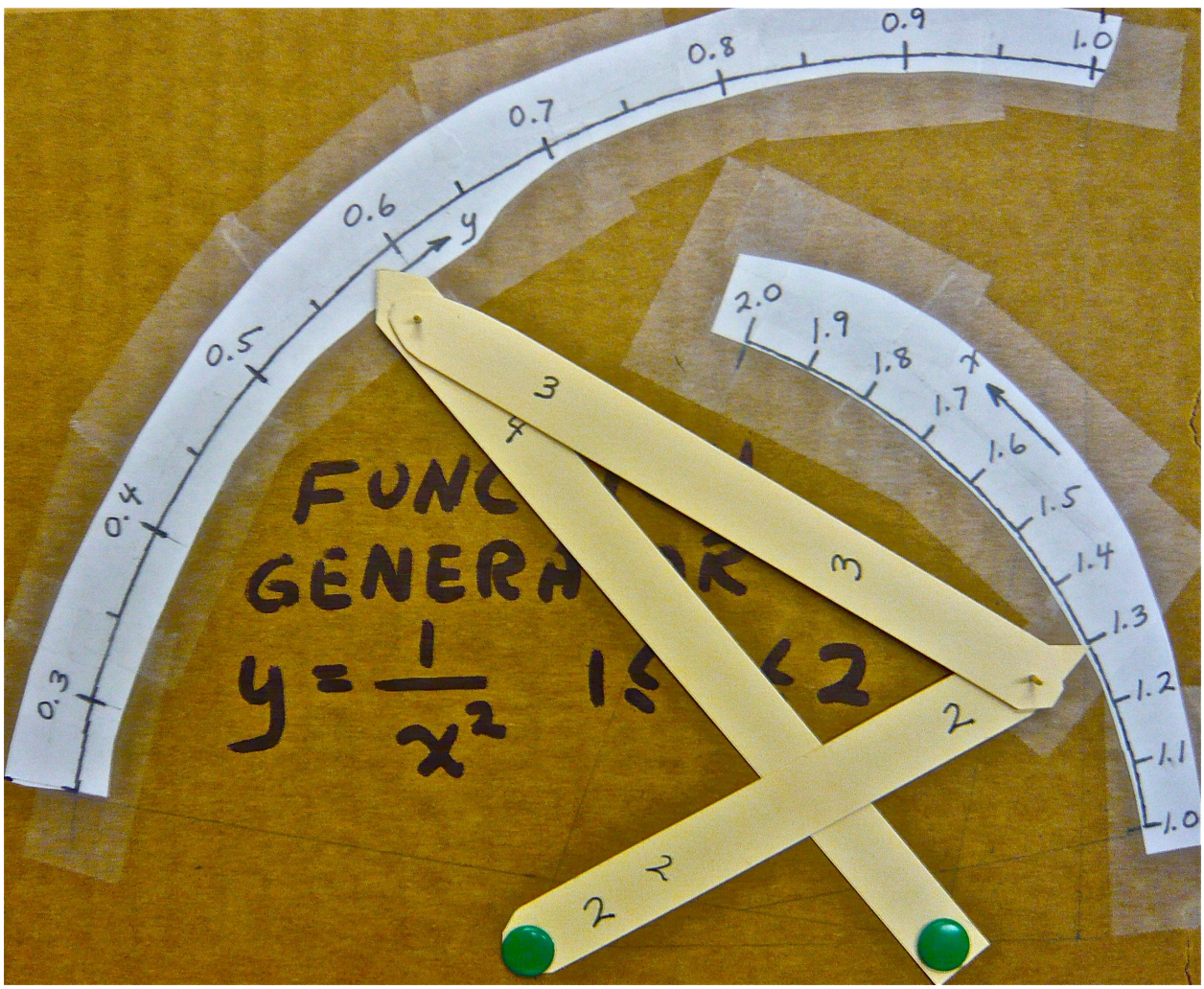


Figure 3: Completed model including input and output x and y scales.

3.3.2 Input and Output Scales

This linkage approximates the mathematical function

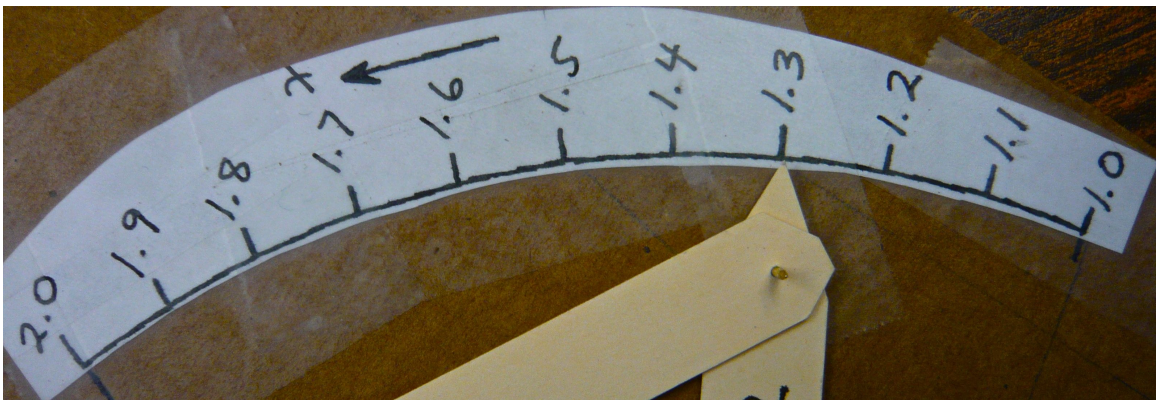
$$y = \frac{1}{x^2} \text{ over the range } 1 \leq x \leq 2$$

but as yet there are no "scales" for x (motion of link 2) or y (motion of link 4). The final step is to create scales which relate the range of the function independent and dependent variables to the angles of links 2 and 4, respectively.

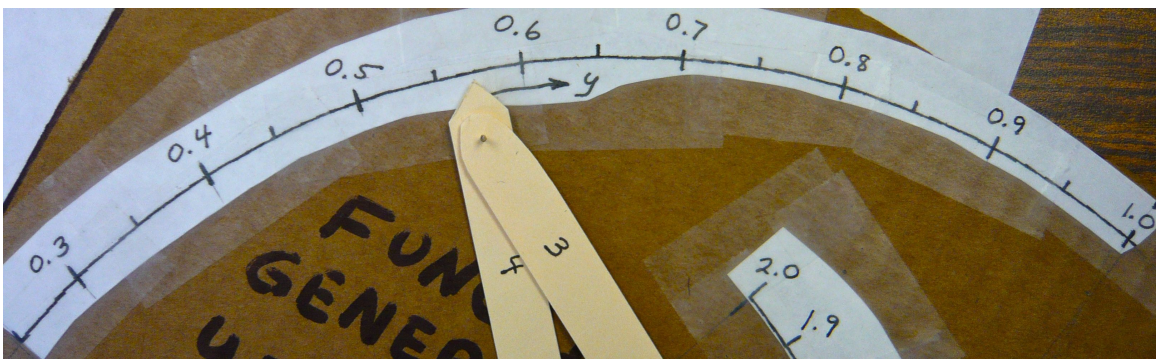
In a sense this a graphical version of the mapping of Section 2.2. Since that mapping is **LINEAR**, both input and output scales will also be **LINEAR**, with equal spacing between tic marks.

All you must do is draw a scale such that the extremal angles of link 2 (10° and 70°) match up with $x = 1$ and $x = 2$ on the x scale. Likewise the extremal angles of link 4 (260° and 350°) match up with $y = 1$ and $y = 0.25$ on the y scale. Then subdivide each scale into convenient divisions.

A closer view of my x and y scales are shown in Figure 4.



(a) Input x scale of 60° from 1 to 2.



(b) Output y scale of 90° from 1 to 0.25.

Figure 4: Closeup of function scales.

4 Structural Error Analysis

The function generator linkage was synthesized using only three points along the range, hence it is bound to have some **structural error** (theoretical difference between the function produced by the linkage and the original mathematical function). To find the structural error we need an **EXACT** model of the linkage, and a means to acquire accurate measurements of angles θ_2 and θ_4.

This is a *PERFECT* application for an ADAMS model! We'll use ADAMS to simulate the linkage and acquire the data, and subsequent MATLAB processing to determine the structural error.

4.1 ADAMS/View Simulation

4.1.1 The Problem

There is a "problem" when creating the ADAMS/View simulation. If you create the ADAMS/View model at the "initial" position—usually by creating links 2 and 4, rotating them into position, then connecting them with link 3—you will find that the length of link 3 from ADAMS/View is **DIFFERENT** than that from the Freudenstein analysis in MATLAB.

> The problem is that there is *ALWAYS* error in the linkage **EXCEPT** at one of the three **precision points**. Therefore, one should really create the ADAMS/View model with the linkage at one of the precision points. The logical precision point to use is #2, which is at the midpoint of the range in x.

4.1.2 The Solution

To create the ADAMS/View model at the midpoint, do the following:

1. Find the midpoint of *independent* variable x: this is simply $x_m = \dfrac{x_i + x_f}{2}$. In my example, this is $x_m = 1.5$.
2. Using the mapping $\theta_2 = ax + b$, find the link 2 angle midpoint θ_{2m}.
3. Find the midpoint of *dependent* variable y: find this using the function you are given: $y_m = f(x_m)$.
4. Using the mapping $\theta_4 = cy + d$, find the link 4 angle midpoint θ_{4m}. It will probably *not* be the midpoint between θ_{4i} and θ_{4f}.
5. Create links 2 and 4 in ADAMS/View, and rotate them into position using θ_{2m} and θ_{4m}.
6. Connect their endpoint markers with link 3. The length of this link *should* equal that from the Freudenstein MATLAB analysis.

A screenshot of my ADAMS/View model is shown in Figure 5 below. The fixed marker out to the right is part of a `ANGLE MEASURE` (discussed in the ADAMS Guide Section 6.2). Since this model was created at the "midpoint," $\theta_2 = 40°$ and $\theta_4 = 326.6667°$.

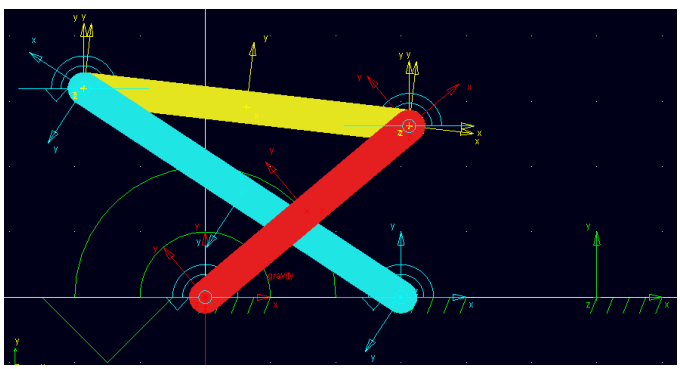


Figure 5: ADAMS/Viewmodel of function generator.

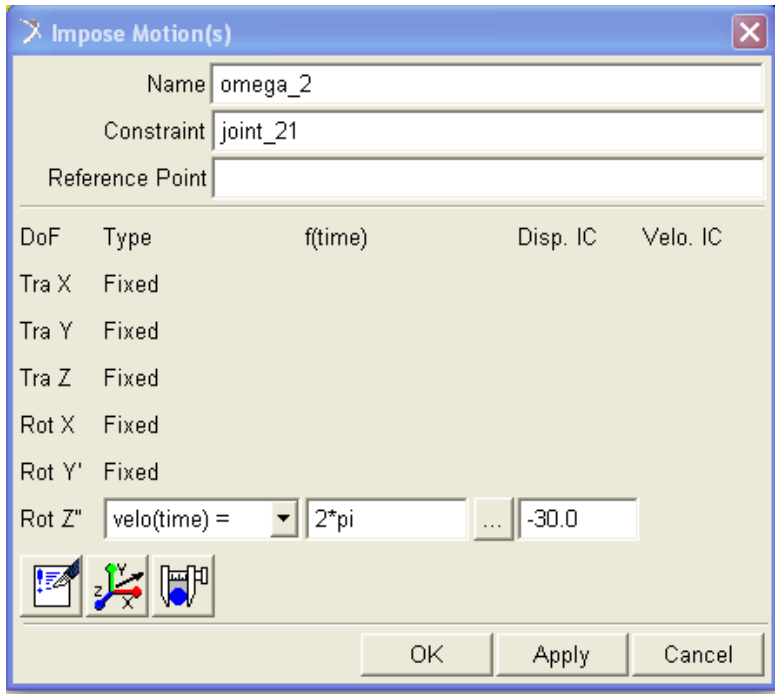
4.1.3 Starting the Simulation at the Initial Position

Construction of the model in this way is straightforward, however there is problem running the simulation: the linkage is not at the **INITIAL** position, it's at the **MIDPOINT!**. How can you start from the initial position?

The easiest way is to specity a **Displacement Initial Condition** when you specify the angular velocity of link 2. After you create the joints, you **Modify** `joint_12` to give it an angular velocity. I typically use 2π rad/sec, since then the link will rotate 360° in 1 second, and simulation durations are easy to calculate.

However, the window used for specifying this angular velocity also gives you the opportunity for a "`Disp. I.C.`" which the key to our initial position. Simply set this `Disp. I.C.` to the value that—when added to the midpoint angle of link 2—will result in the desired initial angle.

For example, my $\theta_{2m} = 40°$, but I want to start at $\theta_{2i} = 10°$. So I want to apply a `Disp. I.C.` of $-30°$. See below:



When you start the simulation the linkage will *instantly* "snap down" to the new initial position and begin moving. If it doesn't, something is wrong.

You must also set the simulation time duration such that it only moves through the range of the function. In my case I set the angular velocity of link 2 to be

$$\omega_2 = 2\pi \text{ rad/s} \tag{29}$$

so link 2 would make a full revolution in 1.00 second. Then since I wish θ_2 to move through an arc of 60° (10° $\rightarrow$ 70°) the time duration must be

$$t = 1/6 \qquad (\texttt{End Time or Duration}) \tag{30}$$

4.2 Exporting Data from ADAMS

You should have `ANGLE MEASURES` (discussed in the ADAMS Guide Section 6) to record angles θ_2 and θ_4. Although not necessary, it might be useful to plot these angles in the ADAMS/Postprocessor window; such a plot is shown in Figure 6. Note that this a plot in which θ_4 is plotted *vs* θ_2, rather than time t. This is a "`data`" plot in the ADAMS/Postprocessor and you must click that button (after which you will be prompted for the variable to use for the horizontal axis).

By the way, I had to add 180° to θ_4 to get the range I wanted, since ADAMS doesn't measure θ_4 like Figure 1.

The whole purpose of creating the ADAMS/View model is to get accurate angle data from your **actual synthesized linkage.** These data will then be analyzed using a MATLAB function for structural error determination.

4.2.1 Linkage Data File

You should **Export** the **Numeric Data** (just θ_2 and θ_4) from the ADAMS/Postprocessor to a file. The procedure for exporting data is covered in the ADAMS Guide Section 7. When you open the file you will see the following:

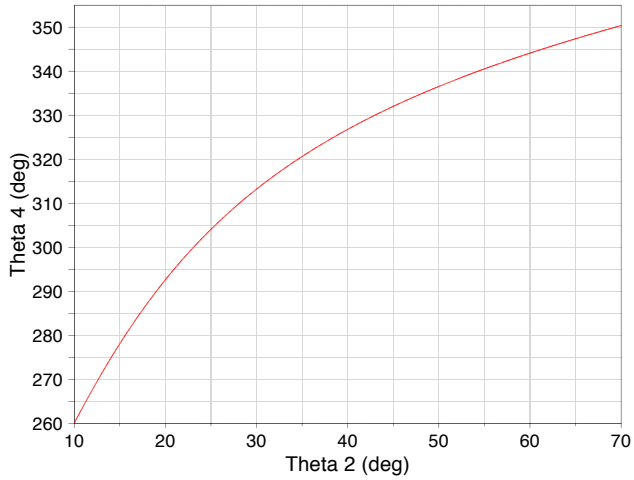


Figure 6: Plot of actual linkage θ_4 *vs* θ_2.

```
A.  .plot_1.curve_1.x_data (deg)
B.  .plot_1.curve_1.y_data (deg)

       A              B
 1.000000E+001  2.597513E+002
 1.012000E+001  2.602262E+002
 1.024000E+001  2.606993E+002
 1.036000E+001  2.611706E+002
    ...            ...
 6.976000E+001  3.501467E+002
 6.988000E+001  3.502165E+002
 7.000000E+001  3.502863E+002
```

You can see that the first column goes from 10° to 70° as expected, while the second column goes from 259.7513° to 350.2863° (instead of from 260° to 350° exactly). There is the first indication of some error.

4.2.2 Formatting for MATLAB Processing

The file shown above cannot be imported into MATLAB directly. You must first delete all the text that appears above the numerical data; that can be done with any text editor.

4.3 Error Determination using MATLAB

From text Section 10.7,

> *Structural error* is defined as the difference between the function produced by your synthesized linkage and the function originally prescribed.

The function originally prescribed is simply $y = f(x)$, in this case it is

$$y = \frac{1}{x^2} \text{ over the range } 1 \le x \le 2 \tag{31}$$

The function produced by the synthesized linkage is *related* to the data in the file you exported (and to the plot of Figure 6. It is related by the inverse of the mapping we found in Section 2.2. This mapping is given by (13)–(14), repeated below:

$$\theta_2 = ax + b$$
$$\theta_4 = cy + d$$

4.3.1 Inverse Mapping

In Section 2.2 we used this mapping to go from $(x, y) \rightarrow (\theta_2, \theta_4)$, now we need to go the other way. We must take the (θ_2, θ_4) data and find the corresponding (x, y) data. This shouldn't be too difficult to figure out.

4.3.2 MATLAB Error Function

You should write a MATLAB function called '`error.m`' to calculate the **percentage** error over the entire range of motion. The syntax should be

```
>> help struc_err
  function [x,y,ya] = struc_err(fn,data,c2,c4) computes the structural
  error of a four-bar function-generating mechanism.  Function handle "fn"
  defines the prescribed mathematical function.  Parameter "data" is a
  2-column array with link 2 angle (DEG) in column 1, and link 4 angle (DEG)
  in column 2.  Parameters "c2" and "c4" are the 2x1 arrays of scaling and
  bias coefficients which map from variable space to link angle space.  The
  "structural error" is simply the difference between the actual function
  y values and the ones from the linkage.  Returned vectors "x", "y", and
  "ys" are the function dependent variable, exact independent variable, and
  approximate independent variable (from your linkage).
```

4.3.3 Application to this Example

Using the ADAMS/View model of the four-bar with link lengths of p. 8 and the initial and final angles for θ_2 and θ_4 from the last execution of "`freud()`" on p. 7, I exported the angle data from ADAMS/Postprocessor into an array called "`data`."

After editing out the text at the top of the file, I loaded it into MATLAB and began the error analysis as shown below (comments typed in after I pasted it here):

```
>> [r,c2,c4] = freud(fn,xi,xf,10,70,260,350);  % Do design, get c2 and c4 mapping parameters
>> load data;  % Load the numeric file containing theta_2 and theta_4 from ADAMS simulation
>> [x,y,ya] = struc_err(fn,data,c2,c4);  % Do error analysis using data, c2, c4
```

Now we have the exact (actual function) and approximate (my linkage) data.

Plot of Exact and Approximate Results. We can plot both the exact and approximate results for dependent variable y versus independent variable x. The plot is shown in Figure 7. It looks like there is good agreement, although some error is evident.

It is interesting to compare the plot of Figure 7 to that of Figure 6—the curves are completely different! That's due to the **mapping** between joint angle space (θ_2, θ_4) and function space (x, y).

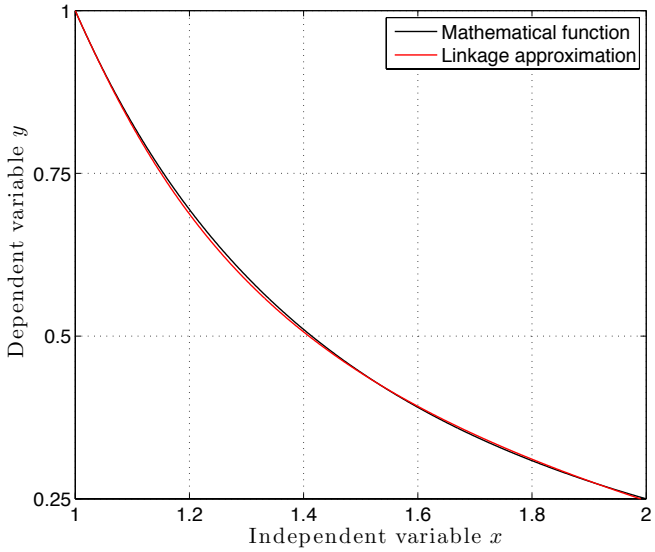


Figure 7: Exact and approximate behavior of $y = \dfrac{1}{x^2}$ over the range $1 \leq x \leq 2$.

Plot of Percentage Error. More meaningful is a plot showing the % error of the function generator linkage. If you have exact vector variable $\mathbf{y}$, approximate vector variable $\mathbf{y}_a$, you know that the % error is given by

$$\% \text{ error} = \frac{\mathbf{y} - \mathbf{y}_a}{\mathbf{y}} * 100 \tag{32}$$

In MATLAB, you must perform an element-by-element operation (you can't divide one vector by another), using the "`dot`" prefix like this:

```
perror =((y-ya)./y)*100;
```

The "`dot`" operator, when prefixed to any MATLAB basic arithmetic operation, performs the operation element-by-element on the vector arguments. That is, it divides element i of the numerator vector by element i of the denominator vector, and returns a result the same length as the arguments. Very useful.

The plot of % error is shown in Figure 8.

The synthesized linkage based on only *three* points has less than **1.2% error** over the range of the function. ***Not bad!***

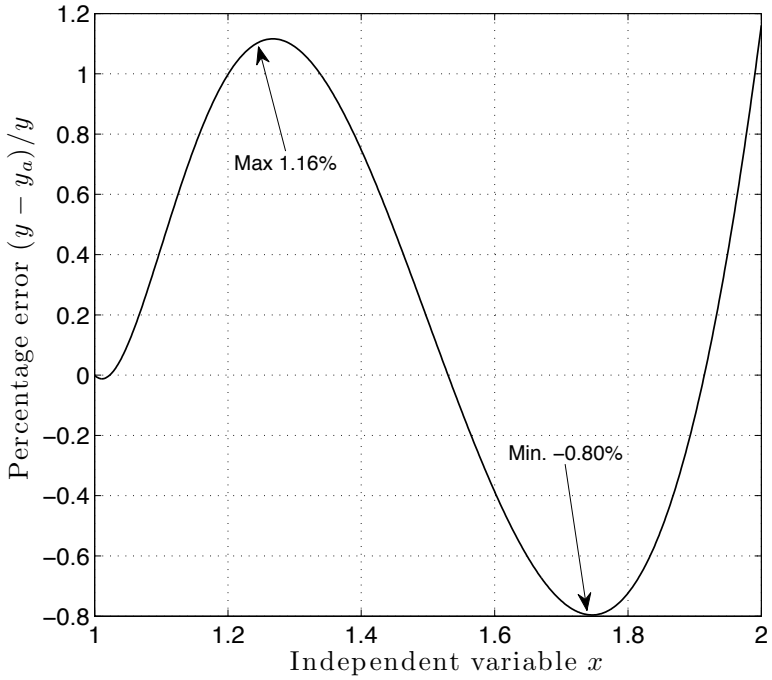


Figure 8: Percentage error of linkage over function interval