Drivetrain Hub (https://drivetrainhub.com)

Geometry / Planetary Gears

Authors: Chad Glinsky (https://www.linkedin.com/in/chad-glinsky-a1840b13/)

Description: Review the geometric design of planetary geartrain layouts.

Table of Contents

Notebook imports and settings

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
from math import pi, radians, degrees, isclose, sqrt, cos, sin, asin, floor, ceil
from cmath import exp
import numpy as np
import pandas as pd
import qgrid
from IPython.display import display, HTML

# notebook modules
import planetary

# settings
FIGSIZE = (6, 6)  # size of plots
plt.rcParams.update({'font.size': 12})  # set default font size

# usage mode
DEV_MODE = False
```

In [2]:

```
# DEVELOPMENT USE: %autoreload 1
# PRODUCTION USE: %autoreload 0
%load_ext autoreload
%autoreload 1 if DEV MODE else %autoreload 0
%aimport planetary
```

$\LaTeX$ commands

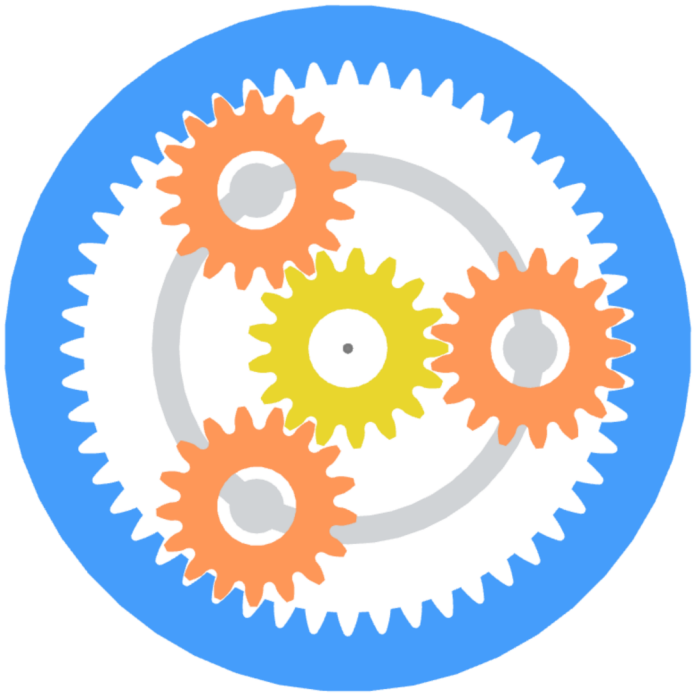
```
$\newcommand{\mm}{\text{mm}}$
$\newcommand{\deg}{^\circ}$
$\newcommand{\degt}{\text{deg}}$
```

Introduction

The geometry of planetary geartrain layouts is reviewed here. Planetary geartrains can be configured in several ways, with the most common architectures reviewed here. Attractive characteristics of planetary geartrains include:

- Mechanical power is transferred between concentric bodies (ring, sun, carrier)
- Power density is high, providing great mechanical power in a small package
- Multiple gear meshes act on central bodies, with self-centering tendencies
- Planet gears may precess with the carrier reference frame
- Cylindrical involute gears are used to transmit motion
- Multiple power inputs or outputs can be defined

The visualization of a planetary geartrain clearly demonstrates its unique geometric characteristics.



Planetary geartrain modeled in Gears App (https://drivetrainhub.com/gears)

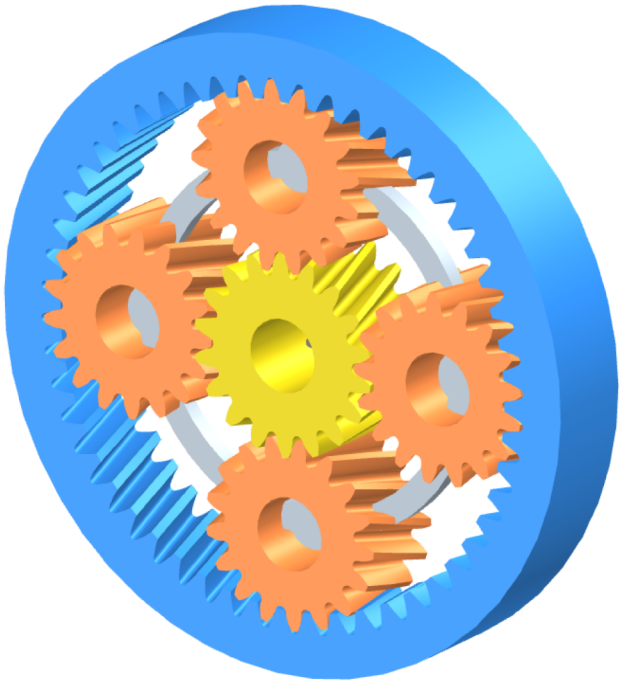
Most commonly, planetary geartrains have a sun gear (yellow), ring gear (blue), and multiple planet gears (orange) on a carrier (gray). Spur or helical gears are used for a planetary gearset, with each type having advantages. This notebook reviews the unique geometric attributes of planetary gears and their significance to planetary gear mesh operation.

The assembly of a planetary geartrain is dependent on various geometric attributes, such as tooth counts.

The various conditions that must be satisfied for a planetary geartrain to successfully assemble are reviewed in this notebook. To understand the geometry of the spur or helical gears that makeup a planetary geartrain, see the respective notebooks on gear geometry.

Standard Planetary

This section reviews the design layout of standard planetary geartrains, i.e. planetaries with a single sun, ring, and set of planets like shown in the image below. Working examples are provided for spur and helical planetary geartrains.



Standard planetary modeled in Gears App (https://drivetrainhub.com/gears)

Nomenclature

This table provides a set of parameters commonly used to define the geometry of planetary geartrains.

Symbol	Description
i	Transmission ratio
u	Base ratio
ω	Angular velocity
m_n	Normal module
z	Number of teeth
β	Helix angle
d_a	Tip diameter
p_t	Transverse pitch

d_w	Pitch diameter
a	Center distance
l	Planet distance
N	Number of planets

For a planetary geartrain, the subscripts $_1$, $_2$, $_3$, and $_V$ are used for the sun, planet, ring, and carrier, respectively.

Ratios

When designing a planetary geartrain, a transmission ratio is typically used to define a target or goal. The design space is largely determined from the target ratio, so an allowable deviation is often used depending on the application requirements. The general equation for transmission ratio is expessed as:

$$i = \frac{\omega_{\text{in}}}{\omega_{\text{out}}}$$

A *base ratio* is defined as the transmission ratio between the sun and ring gear, with a fixed carrier. This is effectively an idler geartrain, with the planets behaving as idler gears. The ratio is calculated as:

$$u = \frac{z_3}{z_1}$$

where sign convention for number of ring gear teeth, z_3, is negative. The transmission ratio between the sun gear and carrier, with the ring gear constrained to ground, is expressed as:

$$i_{1V} = 1 - u$$

Additionally, the transmission ratio between the carrier and ring gear, with the sun gear constrained to ground, is expressed as:

$$i_{3V} = 1 - \frac{1}{u}$$

Other transmission ratios of interest may be solved for by rearranging the equations above.

Planetary geartrains are capable of having non-zero angular velocity for all components.

To learn more about the fundamentals of rotational kinematics and how gears transfer motion, refer to our kinematics notebook. The current notebook is focused on planetary design layouts.

In [3]:

```
# NOTE: Negative sign convention is used for number of ring gear teeth, z3.

def std_ratio_base(z1, z3):
    """Compute the base transmission ratio between sun and ring, with the carrier constrain

    return z3 / z1


def std_ratio_1v(ratio_base):
    """Compute the transmission ratio between sun and carrier, with the ring gear constrain

    return 1 - ratio_base


def std_ratio_3v(ratio_base):
    """Compute the transmission ratio between the ring and carrier, with the sun gear const

    return 1 - 1 / ratio_base
```

Constraints

To design a *valid* planetary geartrain, one that can be assembled and operate in real-world applications, multiple constraints must be adhered to. All the constraints relate to geometrical relationships of planetary geartrains.

> **Geometric layouts are used to study the feasible design space for planetary geartrains.**
>
> As will be shown, only certain combinations of tooth counts, planet count, and planet spacing are possible.

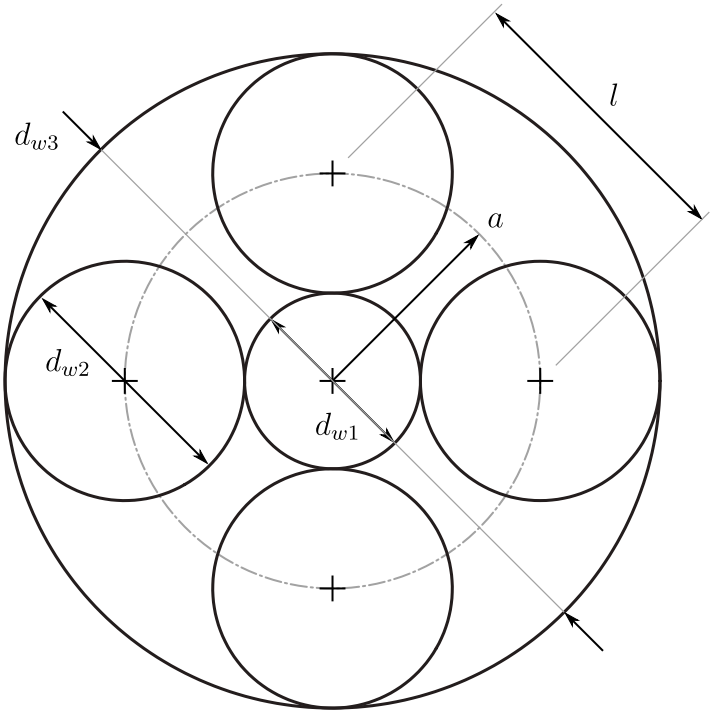
Center Distance

The geometric condition that requires the center distance for the sun-planet and planet-ring gear meshes to be the same is obvious when illustrated. The equation to express this relationship can be expanded as a function of the working pitch diameters:

$$|a_{12}| = |a_{23}|$$

$$|d_{w1} + d_{w2}| = |d_{w2'} + d_{w3}|$$

where $d_{w2'}$ specifically refers to the working pitch diameter of the planet-ring gear mesh, which may differ from the sun-planet mesh due to profile shifts, tooth counts, and center distance differing from nominal (a condition with zero profile shift or backlash). Below illustrates a layout diagram for a 4-planet geartrain:



Planetary layout diagram of 4-planet geartrain

> **Analogous to an idler gear, the planet gear has no effect on the planetary transmission ratios.**
>
> When considering the detailed tooth form of the meshing gears, this means multiple values of planet tooth count are possible for a given design. The design space is somewhat complicated by this, but it is more applicable to the detailed tooth design, so does not require immediate consideration.

In [4]:

```
def reference_planet_size(d1, d3):
    """Compute the reference planet size; diameter or tooth counts may be used.  Actual pla

    return (abs(d3) - d1) / 2
```

Planet Interference

Upon viewing the enveloping geometry, notice the condition of planet-to-planet interference must be considered. Clearance must exist between the tip diameters of each planet. The equation to express this constraint is:

$$l > d_{a2}$$

where l is the center distance between planets, expressed as:

$$l = 2a \sin\left(\frac{\pi}{N}\right)$$

> In the conceptual design stage, tip diameters are usually unknown, so the pitch diameter may be used as an initial approximation.

In [5]:

```
def get_max_planets(tip_diameter_planet, center_distance):
    """Compute the maximum number of planets to avoid interference."""

    return floor(pi / (asin(tip_diameter_planet / (2 * center_distance))))
```

Meshing Teeth

Despite a given enveloping diagram appearing as a valid design, remember that an integer number of teeth is required for all the gears. Adding to this requirement, the teeth of each planet must mesh with both the sun and ring. Determining valid conditions for teeth to mesh depends on the tooth counts, number of planets, and planet spacing.

To understand this requirement, let's imagine assembling a planetary geartrain. First, position the sun at the origin with any angle of rotation. Next, add the first planet to a desired carrier position, rotating its teeth to mesh with the sun. Then, position the ring at the origin, rotating its teeth to mesh with the planet. Now, imagine adding the second planet somewhere on the carrier, not being allowed to rotate the sun or ring. Only certain carrier positions will allow the second planet to mesh with both the sun and ring.

Equal Spacing

For *equally-spaced planets*, the following equation is used to identify valid meshing conditions for a given number of planets and tooth counts:

$$k = \frac{z_1 - z_3}{N}$$

where k must be an integer for the gears to mesh.

Unequal Spacing

It is also possible for the gears to mesh with *unequal planet spacing*. In such cases, the smallest angular unit pertaining to the allowable planet positions is used to find the planet positions nearest equal spacing. This unit angle of planet position, or so-called *tick angle*, may be calculated as:

$$\hat{\theta} = \frac{2\pi}{z_1 - z_3}$$

Typically, unuequal spacing is only used in 4-planet geartrains where the planet gears can be positioned with *diametrically-opposed* spacing. Sometimes this is called an *X* configuration. A diametrically-opposed design is allowed if the remainder of k is 0.5. Designs capable of being equally-spaced can also be modified to be diametrically-opposed.

In [6]:

```
# NOTE: Negative sign convention is used for number of ring gear teeth, z3.

def get_planet_tick_angle(z1, z3):
    """Compute the unit angle to satisfy the planet meshing conditions."""

    return 2 * pi / (z1 - z3)


def get_planet_equal_ticks(z1, z3, number_of_planets):
    """Compute the number of tick angles that correspond to equal planet spacing.  Must be

    return (z1 - z3) / number_of_planets


def allows_equally_spaced(equal_ticks):
    """Check if equally spaced planets are allowed based on the tick count required for equ

    return isclose(equal_ticks, round(equal_ticks))


def allows_diametrically_opposed(equal_ticks):
    """Check if diametrically opposed planet spacing is allowed based on the ticks required

    return isclose(equal_ticks % 1, 0.5)


def get_planet_angles(z1, z3, number_of_planets):
    """Compute the planet angles that are nearest equal-spacing."""

    tick_angle = get_planet_tick_angle(z1, z3)
    equal_ticks = get_planet_equal_ticks(z1, z3, number_of_planets)
    equal_angle = 2 * pi / number_of_planets

    if allows_equally_spaced(equal_ticks):
        thetas = [i * equal_angle for i in range(number_of_planets)]
    elif allows_diametrically_opposed(equal_ticks):
        thetas = [0]
        for i in range(1, number_of_planets):
            nearest_whole_tick = ceil(i * equal_ticks)
            theta = nearest_whole_tick * tick_angle
            thetas.append(theta)
    else:
        raise ValueError('Invalid planetary design.')

    return thetas
```

Mesh Phasing

Once the geometric conditions are met to ensure a valid planetary geartrain can be assembled, it is also of interest to consider the phasing of the planetary gear meshes. Phasing refers to the cyclic behavior of multiple gear meshes with the same frequency.

Three types of planetary phasing exist for nominal conditions.

1. In-phase
2. Sequential-phase

3. Counter-phase

Before explaining each phasing type, it must understood that phasing specifically refers to the phase angle for the harmonics of gear mesh excitation caused by imperfect conjugate action, commonly referred to as transmission error. These excitations are caused by numerous things, including manufacturing and assembly errors, but also due to the deflection of gear teeth under operating loads.

> **Gear mesh excitations can be viewed as a series of sinusoidal signals, called harmonics, each with an amplitude and phase angle.**
>
> Fourier transform is used to analyze a time signal as a series of sinusoids, useful for harmonic analysis.

Phase Angles

To understand the different types of planetary phasing, we must first consider the phasing of each gear mesh in the planetary geartrain. Let's individually examine the gear meshes for the ring and sun.

> Only the *relative* phase angles are of significance for planetary phasing analysis.

Planet-Ring

To determine the phase angle of a planet-ring mesh, we must define a relationship between the position of ring gear teeth and the position of the planet gear, e.g. equally-spaced or diametrically-opposed. Successive ring gear teeth are separated by the ring transverse pitch angle, defined as:

$$\psi_{t3} = \frac{2\pi}{z_3}$$

By writing a fraction that divides the i^{th} planet angle by the ring transverse pitch angle,

$$q_{3i} = \frac{\theta_i}{\psi_{t3}}$$

the mesh phase angle is determined by the remainder of q_{3i}, the fraction of mesh cycle, according to:

$$\phi_{3i} = 2\pi * (q_{3i} \bmod 1)$$

Sun-Planet

Following a similar logic as above, we can compute the sun-planet gear mesh phase angles as with equations:

$$\psi_{t1} = \frac{2\pi}{z_1}$$

$$q_{1i} = \frac{\theta_i}{\psi_{t1}}$$

$$\phi_{1i} = 2\pi * (q_{1i} \bmod 1)$$

In [7]:

```
def pitch_angle_transverse(number_of_teeth):
    """Spur or helical gear transverse pitch angle."""

    return 2 * pi / abs(number_of_teeth)


def mesh_cycle_fraction(planet_position_angle, transverse_pitch_angle):
    """Calculate the fraction of mesh cycle for a given planet position.

    Notes:
        1. Only the relative values of phasing are important.
        2. It is assumed that a planet at zero angle is at the start of a mesh cycle.

    About relative phasing of sun and ring for a given planet:
        1. It depends on the planet tooth count.
        2. It depends on the backlash of each gear mesh.
        3. It does not affect the geartrain phasing type.
    """

    return planet_position_angle / transverse_pitch_angle % 1


def mesh_phase_angle(planet_position_angle, number_of_teeth):
    """Calculate the mesh phase angle in radians."""

    transverse_pitch_angle = pitch_angle_transverse(number_of_teeth)

    return 2 * pi * mesh_cycle_fraction(planet_position_angle, transverse_pitch_angle)


def summed_phasing(planet_angles, phase_angles):
    """Calculate the sum of gear mesh excitations, reporting a normalized metric for design

    fx = 0
    fy = 0
    mz = 0

    for theta, phi in zip(planet_angles, phase_angles):
        signal = exp(1j * phi)
        fx += cos(theta) * signal
        fy += sin(theta) * signal
        mz += signal

    # normalize metrics; 1.0 is fully reinforced
    n_planets = len(planet_angles)
    fx_sum = abs(fx) / (n_planets / 2)
    fy_sum = abs(fy) / (n_planets / 2)
    trq_sum = abs(mz) / n_planets

    return fx_sum, fy_sum, trq_sum


def get_phasing_type(fx_sum, fy_sum, trq_sum):
    """Return a string to indicate the type of planetary phasing.

    Phasing types:
        1. 'I' - in-phase
        2. 'C' - counter-phase
        3. 'S' - sequential-phase
```

```
        4. 'M' - mixed-phase
    """

    f_sum = fx_sum + fy_sum
    f_cancelled = isclose(f_sum, 0, abs_tol=1e-3)
    f_reinforced = isclose(f_sum, 2)

    trq_cancelled = isclose(trq_sum, 0, abs_tol=1e-3)
    trq_reinforced = isclose(trq_sum, 1)

    if f_cancelled and trq_reinforced:
        typ = 'I'
    elif f_cancelled and trq_cancelled:
        typ = 'C'
    elif f_reinforced and trq_cancelled:
        typ = 'S'
    else:
        typ = 'M'

    return typ
```

In [8]:

```
if DEV_MODE:
    print('----------------')
    print('--- DEV ONLY ---')
    print('----------------')

    models = [
        (4, -48, 16),  # in-phase, equally-spaced
        (4, -51, 17),  # sequential-phase, equally-spaced
        (4, -54, 18),  # counter-phase, equally-spaced
        (4, -52, 18),  # mixed-phase, diametrically-opposed
        (5, -52, 18),  # counter-phase, equally-spaced
    ]

    for N, zr, zs in models:
        print('N, ZR, ZS:', N, zr, zs)

        # planet angles
        thetas = get_planet_angles(zs, zr, N)
        print('PLANET ANGLES:', [round(degrees(theta), 3) for theta in thetas])

        # phase angles, ring
        phis_ring = [mesh_phase_angle(theta, zr) for theta in thetas]
        print('PHASE ANGLES, RING:', [round(degrees(phi), 3) for phi in phis_ring])

        # phase angles, sun
        phis_sun = [mesh_phase_angle(theta, zs) for theta in thetas]
        print('PHASE ANGLES, SUN:', [round(degrees(phi), 3) for phi in phis_sun])

        # sum of forces and torques; ring & sun
        fx_ring, fy_ring, trq_ring = summed_phasing(thetas, phis_ring)
        fx_sun, fy_sun, trq_sun = summed_phasing(thetas, phis_sun)
        print('FORCES-X:', round(fx_ring, 3), round(fx_sun, 3))
        print('FORCES-Y:', round(fy_ring, 3), round(fy_sun, 3))
        print('TORQUE:', round(trq_ring, 3), round(trq_sun, 3))

        # phasing type; ring & sun
        phase_type_ring = get_phasing_type(fx_ring, fy_ring, trq_ring)
        phase_type_sun = get_phasing_type(fx_sun, fy_sun, trq_sun)
        print('TYPE:', phase_type_ring, phase_type_sun)
        print('------')
```

Phasing Types

The phasing classification for a planetary geartrain is based on the first harmonic of its gear mesh excitations. Higher harmonics may have different phasing. The following equation can be used to determine phasing with *equally-spaced planets*:

$$k_\phi = |z_3| \bmod N$$

In particular, the phasing is of interest to understand which gear mesh excitations reinforce or cancel in the system. Excitation reinforcements and cancellations are studied for the following summed components:

1. Torques and axial forces, i.e. thrusts (helical only)
2. Radial forces

In-Phase

In-phase refers to having the same phase angle for each gear mesh. In-phase planetaries have reinforced torques and thrusts but cancelled radial forces. The condition for in-phase is:

$$k_\phi = 0$$

Sequential-Phase

Sequential-phase refers to having equally separated phase angles for subsequent gear meshes. Sequential-phase planetaries have cancelled torques and thrusts but reinforced radial forces. The condition for sequential-phase:

$$k_\phi = 1, N-1$$

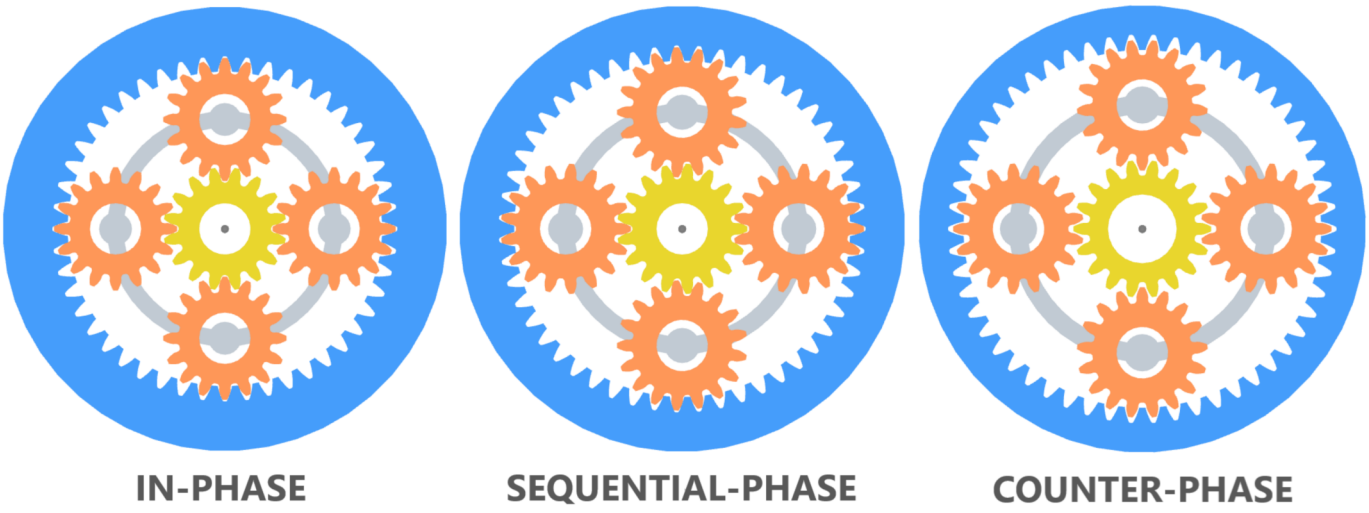
Counter-Phase

Counter-phase refers to having opposite phase angles for opposing planets. Counter-phase planetaries have all summed components of excitation cancelled. The condition for counter-phase is:

$$k_\phi = 2, 3, \ldots, N-2$$

Mixed-Phase

Mixed-phase refers to having a combination of phasing types. This can only occur with *unequally-spaced planets*, as found in diametrically-opposed designs. In such cases, the gear mesh phase angles must be calculated and used in the sum of harmonics to determine which reinforcements and cancellations occur.



Planetary geartrain phasing types modeled in Gears App (https://drivetrainhub.com/gears)

Perfect planetary phasing conditions are not observed in real-world conditions.

Errors caused by manufacturing, assembly, and operation can cause gear mesh phasing deviations and low frequency modulations that will lead to more vibration frequencies, increased vibrations, and undesireable noise characteristics. Planetary phasing should be considered during the design-phase.

In [9]:

```
# NOTE: Sign convention for number of ring gear teeth is insignificant here.

def mod_phasing(z3, number_of_planets):
    """Modulo operation to compute gear mesh phasing for equally-spaced planets."""

    return abs(z3) % number_of_planets


def is_in_phase(k_phi):
    """Check if equally-spaced gear meshes are in-phase."""

    return k_phi == 0


def is_sequential_phase(k_phi):
    """Check if equally-spaced gear meshes are sequential-phase."""

    return k_phi == 1 or k_phi == number_of_planets - 1


def is_counter_phase(k_phi):
    """Check if equally-spaced gear meshes are counter-phase."""

    return k_phi == 2 or k_phi == 3 or k_phi == number_of_planets - 2
```

Helical Gears

To extend the planetary design rules to helical gearing, we need to consider the helix angle and the transverse geometric properties, such as module and pitch diameters. The transverse module of a helical gear is most easily understood as the millimeters of theoretical pitch diameter per tooth:

$$m_t = \frac{d}{z}$$

which can be expressed as a function of the normal module and helix angle according to:

$$m_t = \frac{m_n}{\cos \beta}$$

It can be deduced from the equations above that for a given normal module and number of teeth, increasing the helix angle causes an increase in theoretical pitch diameter. This is important to consider for planetary layout design.

Phasing

For the fundamentals of planetary phasing, the introduction of a helix angle does not change a lot. The concept is still the same, i.e. each planetary gear mesh produces a dynamic signal at the mesh frequency and these signals may cancel or reinforce in various ways. Since a helix angle introduces axial forces into the gear mesh, it may also be considered in the cancellations and reinforcements.

- The sum of axial forces cancel / reinforce the same as torques.

> To learn more about the detailed attributes of helical gears, see our notebook on helical gears and gear meshes.

In [10]:

```
# EXAMPLE HELPERS
# ---------------
cols_candidate = ['#', 'z1', 'z2', 'z3', 'ratio', 'beta', 'N', 'space', 'phase', 'fx', 'fy'


def compute_candidate(i_candidate, z1, z3, beta, number_of_planets, spacing):
    # compute actual ratio
    ratio_base = std_ratio_base(z1, z3)
    ratio_1v = std_ratio_1v(ratio_base)

    # compute reference planet teeth; actual tooth count may vary depending on profile shif
    z2 = reference_planet_size(z1, z3)

    # compute tooth size and center distance
    mt = ring_pitch_dia_target / z3
    mn = mt * cos(beta)
    reference_center_distance = mt * (z1 + z2) / 2

    # compute phasing data
    thetas = get_planet_angles(z1, z3, number_of_planets)
    phis = [mesh_phase_angle(theta, z3) for theta in thetas]
    fx_sum, fy_sum, trq_sum = summed_phasing(thetas, phis)
    phasing_type = get_phasing_type(fx_sum, fy_sum, trq_sum)

    return [i_candidate, z1, f'~{z2}', z3, round(ratio_1v, 2), degrees(beta), number_of_pla


def get_candidate_dict(candidate_row):
    return {key: val for key, val in zip(cols_candidate, candidate_row)}


def is_candidate_diametrically_opposed(candidate_dict):
    return candidate_dict['space'] == 'X'


def candidate_center_distance(candidate_dict):
    return candidate_dict['CD']
```

Example | Standard Planetary

GIVEN

1. Target transmission ratio, with +/- range.
2. Target ring gear diameter.
3. Target number of planets.
4. Range of helix angles.

FIND

Standard planetary designs that satisfy the input constraints.

SOLUTION

See below.

In [11]:

```
# EXAMPLE | STANDARD PLANETARY
# ---------------------------
ratio_1v_target = 4  # omega_sun / omega_carrier, omega_ring=0
ratio_1v_delta = 0.03  # decimal percentage

ring_pitch_dia_target = -104
n_planets_target = 4

helix_angle_min = 15  # (+) degrees
helix_angle_max = 15  # (+) degrees

z3_min = -40
z3_max = -150

# NOTES
# -----
# Alternatively, tooth size could be constrained, allowing envelope size to vary.

# TARGET RATIOS
# -------------
ratio_1v_lower = (1 - ratio_1v_delta) * ratio_1v_target
ratio_1v_upper = (1 + ratio_1v_delta) * ratio_1v_target
base_ratio_target = 1 - ratio_1v_target

# TARGET HELIX
# ------------
helix_angles_deg = np.arange(helix_angle_min, helix_angle_max + 1)
helix_angles_rad = [radians(beta) for beta in helix_angles_deg]

# TARGET DIAMETERS
# ----------------
sun_pitch_dia_target = ring_pitch_dia_target / base_ratio_target
planet_pitch_dia_target = reference_planet_size(sun_pitch_dia_target, ring_pitch_dia_target
carrier_pitch_dia_target = sun_pitch_dia_target + planet_pitch_dia_target
center_distance_target = carrier_pitch_dia_target / 2

# MAX PLANETS
# -----------
tip_diameter_planet = planet_pitch_dia_target  # CAUTION
number_of_planets_max = get_max_planets(tip_diameter_planet, center_distance_target)
number_of_planets = min([n_planets_target, number_of_planets_max])

if n_planets_target > number_of_planets_max:
    print(f"\x1b[31mWARNING: The number of planets is too high, using {number_of_planets} i

# DESIGN CANDIDATES
# -----------------
candidates = []
i_candidate = 1
z3_range = range(z3_min, z3_max - 1, -1)

# compute designs within constraints
for z3 in z3_range:
    # range of z1 to satisfy ratio constraints
    z1_high = z3 / (1 - ratio_1v_lower)  # higher z1 for lower ratio
    z1_low = z3 / (1 - ratio_1v_upper)  # lower z1 for higher ratio
    z1_values = list(range(ceil(z1_low), floor(z1_high) + 1))

    for z1 in z1_values:
```

```
        # determine planet spacing
        equal_ticks = get_planet_equal_ticks(z1, z3, number_of_planets)

        if allows_equally_spaced(equal_ticks):
            spacing = 'E'
        elif allows_diametrically_opposed(equal_ticks):
            spacing = 'X'
        else:
            continue

        # candidate for each helix angle
        for helix_angle in helix_angles_rad:
            candidates.append(compute_candidate(i_candidate, z1, z3, helix_angle, number_of
            i_candidate += 1

# DISPLAY TARGETS
# ---------------
cols = ['Description', 'Symbol', 'Value', 'Units']
data = [
    ['Target ratio', '$i_{1v_o}$', ratio_1v_target, '-'],
    ['Target ratio deviation', '$\\Delta i_{1v_o}$', 100 * ratio_1v_delta, '%'],
    ['Target pitch diameter, ring', '$d_{w3_o}$', ring_pitch_dia_target, '$\mm$'],
    ['Target pitch diameter, sun', '$d_{w1_o}$', round(sun_pitch_dia_target, 3), '$\mm$'],
    ['Target pitch diameter, planets', '$d_{w2_o}$', round(planet_pitch_dia_target, 3), '$\
    ['Target pitch diameter, carrier', '$d_{wV_o}$', round(carrier_pitch_dia_target, 3), '$
    ['Target reference center distance', '$a_{0_o}$', round(center_distance_target, 3), '$\
    ['Helix angle, min', '$\\beta_\\text{min}$', helix_angle_min, '$\deg$'],
    ['Helix angle, max', '$\\beta_\\text{max}$', helix_angle_max, '$\deg$'],
    ['Number of planets, target', '$N_o$', n_planets_target, '-'],
    ['Number of planets, max', '$N_\\text{max}$', number_of_planets_max, '-'],
]
df = pd.DataFrame(data=data, columns=cols)
display(HTML(df.to_html(index=False)))

if not candidates:
    print(f"\x1b[31mNo candidates found in the design space.\x1b[0m")  # red color formatti
```

Description	Symbol	Value	Units
Target ratio	i_{1v_o}	4.000	-
Target ratio deviation	Δi_{1v_o}	3.000	%
Target pitch diameter, ring	d_{w3_o}	-104.000	mm
Target pitch diameter, sun	d_{w1_o}	34.667	mm
Target pitch diameter, planets	d_{w2_o}	34.667	mm
Target pitch diameter, carrier	d_{wV_o}	69.333	mm
Target reference center distance	a_{0_o}	34.667	mm
Helix angle, min	$\beta_{\min}$	15.000	°
Helix angle, max	$\beta_{\max}$	15.000	°
Number of planets, target	N_o	4.000	-
Number of planets, max	$N_{\max}$	5.000	-

In [12]:

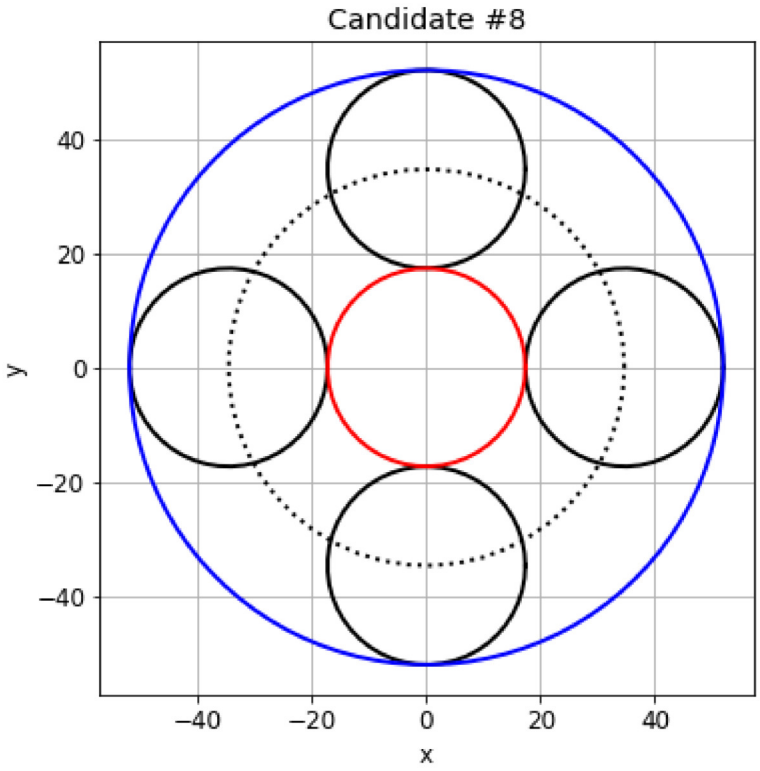
```
# DISPLAY CANDIDATES
# ------------------

if DEV_MODE:
    # qgrid approach:
    df = pd.DataFrame(data=candidates, columns=cols_candidate)
    df = df.set_index('#')
    qgridwidget = qgrid.show_grid(df)
else:
    # html approach:
    df = pd.DataFrame(data=candidates, columns=cols_candidate)
    display(HTML(df.to_html(index=False, max_rows=10)))

qgridwidget if DEV_MODE else None
```

#	z1	z2	z3	ratio	beta	N	space	phase	fx	fy	trq	mn	CD
1	14	~14.0	-42	4.00	15.0	4	E	C	0.00	0.0	0.00	2.392	34.667
2	15	~15.0	-45	4.00	15.0	4	E	S	1.00	1.0	0.00	2.232	34.667
3	16	~16.0	-48	4.00	15.0	4	E	I	0.00	0.0	1.00	2.093	34.667
4	17	~16.0	-49	3.88	15.0	4	X	M	1.03	1.0	0.00	2.050	35.020
5	17	~17.0	-51	4.00	15.0	4	E	S	1.00	1.0	0.00	1.970	34.667
...	...	...	...	...	...	...	...	...	...	...	...	...	...
136	50	~49.0	-148	3.96	15.0	4	X	M	0.00	0.0	0.39	0.679	34.784
137	49	~50.0	-149	4.04	15.0	4	X	M	1.01	1.0	0.00	0.674	34.550
138	51	~49.0	-149	3.92	15.0	4	E	S	1.00	1.0	0.00	0.674	34.899
139	50	~50.0	-150	4.00	15.0	4	E	C	0.00	0.0	0.00	0.670	34.667
140	52	~49.0	-150	3.88	15.0	4	X	M	0.00	0.0	0.92	0.670	35.013

In [13]:

```
# PLOT CANDIDATE
# --------------
candidate_number = 8

# candidate attributes
candidate_row = candidates[candidate_number-1]
candidate = get_candidate_dict(candidate_row)
z1 = candidate['z1']
z2 = candidate['z2']
z3 = candidate['z3']
ratio = candidate['ratio']
beta = radians(candidate['beta'])
N = candidate['N']
space_type = candidate['space']
phase_type = candidate['phase']
mn = candidate['mn']
cd = candidate['CD']
carrier_pcd = 2 * cd

# computed data
mt = mn / cos(beta)
d_w1 = z1 * mt
d_w3 = z3 * mt
d_w2 = reference_planet_size(d_w1, d_w3)
planet_angles = get_planet_angles(z1, z3, N)
phis_ring = [mesh_phase_angle(theta, z3) for theta in planet_angles]
phis_sun = [mesh_phase_angle(theta, z1) for theta in planet_angles]

# initialize plot
fig = plt.figure(figsize=FIGSIZE)
ax = fig.add_subplot(1, 1, 1)

# carrier pitch circle
x, y = planetary.circle_curve(carrier_pcd / 2)
ax.plot(x, y, 'k:', linewidth=2)

# planet pitch diameters
for i in range(number_of_planets):
    xc, yc = planetary.polar_to_cartesian(cd, planet_angles[i])
    x, y = planetary.circle_curve(d_w2 / 2, x_center=xc, y_center=yc)
    ax.plot(x, y, 'k', linewidth=2)

# ring pitch diameter
x, y = planetary.circle_curve(d_w3 / 2, np.linspace(0, 2*pi, 100))
ax.plot(x, y, 'b', linewidth=2)

# sun pitch diameter
x, y = planetary.circle_curve(d_w1 / 2)
ax.plot(x, y, 'r', linewidth=2)

# plot format
ax.set_aspect('equal')
plt.title(f'Candidate #{candidate_number}')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.show()

# display candidate table
```

```
cols = ['Description', 'Symbol', 'Value', 'Units']
data = [
    ['Candidate', '#', candidate_number, '-'],
    ['Sun tooth count', '$z_1$', z1, '-'],
    ['Ring tooth count', '$z_3$', z3, '-'],
    ['Planet tooth count', '$z_2$', z2, '-'],
    ['Transmission ratio', '$i_{1V}$', ratio, '-'],
    ['---', '', '', ''],
    ['Number of planets', '$N$', N, '-'],
    ['Planet spacing', '-', space_type, '-'],
    ['Planet angles', '$\\theta_i$', [round(degrees(theta), 5) for theta in planet_angles],
    ['---', '', '', ''],
    ['Phasing type', '-', phase_type, '-'],
    ['Phase angles, ring', '$\\phi_{3i}$', [round(degrees(phi), 3) for phi in phis_ring], '
    ['Phase angles, sun', '$\\phi_{1i}$', [round(degrees(phi), 3) for phi in phis_sun], '$\
    ['---', '', '', ''],
    ['Helix angle', '$\\beta$', round(degrees(beta), 3), '$\deg$'],
    ['Normal module', '$m_n$', round(mn, 3), '$\mm$'],
    ['Transverse module', '$m_t$', round(mt, 3), '$\mm$'],
    ['Reference center distance', '$a_0$', round(cd, 3), '$\mm$'],
]
df_candidate = pd.DataFrame(data=data, columns=cols)
display(HTML(df_candidate.to_html(index=False)))
```



Description	Symbol	Value	Units
Candidate	#	8	-
Sun tooth count	z_1	18	-
Ring tooth count	z_3	-54	-
Planet tooth count	z_2	~18.0	-
Transmission ratio	i_{1V}	4.0	-

Number of planets	N	4	-
Planet spacing	-	E	-
Planet angles	θ_i	[0.0, 90.0, 180.0, 270.0]	°

Phasing type	-	C	-
Phase angles, ring	ϕ_{3i}	[0.0, 180.0, 0.0, 180.0]	°
Phase angles, sun	ϕ_{1i}	[0.0, 180.0, 0.0, 180.0]	°

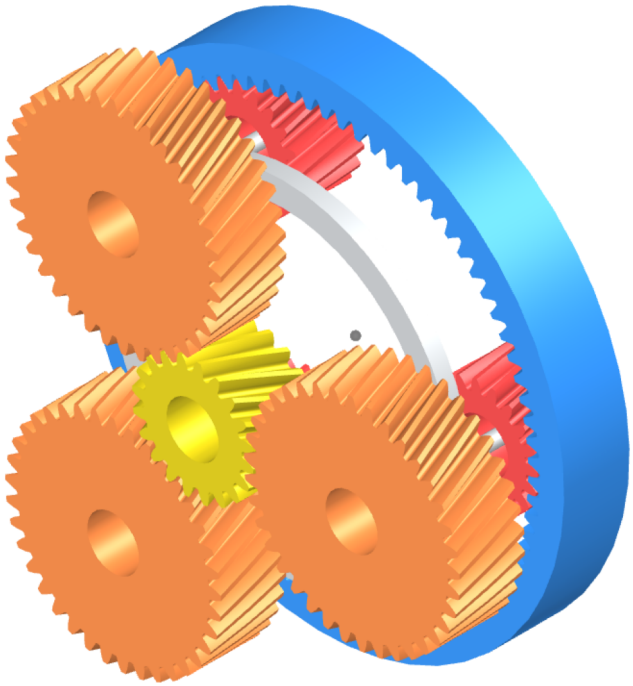
Helix angle	β	15.0	°
Normal module	m_n	1.86	mm
Transverse module	m_t	1.926	mm
Reference center distance	a_0	34.667	mm

In [14]:

```
# SIZE ADJUSTMENT
# ---------------
if DEV_MODE:
    module_normal = 2
    module_transverse = module_normal / cos(beta)
    z2 = reference_planet_size(z1, z3)
    reference_center_distance = module_transverse * (z1 + z2) / 2

    print('MODULE, NORMAL:', module_normal)
    print('MODULE, TRANSVERSE:', module_transverse)
    print('CENTER DISTANCE:', reference_center_distance)
    print('CARRIER PCD:', 2 * reference_center_distance)
```

Compound Planetary

This section reviews the design layout of compound planetary geartrains with a sun, ring, and planet shafts each with two planet gears. Reference the image below. Working examples are provided for spur and helical planetary geartrains.



Compound planetary modeled in Gears App (https://drivetrainhub.com/gears)

Nomenclature

This table provides a set of parameters commonly used to define the geometry of compound planetary geartrains.

TODO need to add parameters pertaining to the planet gear / pinion

Symbol	Description
i	Transmission ratio
u	Base ratio
ω	Angular velocity
m_n	Normal module
z	Number of teeth
β	Helix angle
d_a	Tip diameter
p_t	Transverse pitch
d_w	Pitch diameter
a	Center distance
l	Planet distance
N	Number of planets

For a planetary geartrain, the subscripts $_1$, $_2$, $_{2'}$, $_3$, and $_V$ are used for the sun, planet gear, planet pinion, ring, and carrier, respectively.

Ratios

TODO define ratios for compound planetary.

Constraints

TODO define constraints for compound planetary design.

Example | Compound Planetary

GIVEN

1. Target transmission ratio, with +/- range.
2. Target ring gear diameter.
3. Target number of planets.
4. Range of helix angles.

FIND
Compound planetary designs that satisfy the input constraints.

SOLUTION
See below.

TODO Solution for compound planetary.

In [15]:

```
# TODO Solution for helical compound planetary.
# Balance the axial forces of planet shaft.
```

Model Gears

Gears App (https://drivetrainhub.com/gears) software is used to accurately model, analyze, and build planetary geartrains entirely in your web browser.

Learn More

Notebook Series (https://drivetrainhub.com/notebooks/) is free to learn and contribute knowledge about gears, such as geometry, manufacturing, strength, and more.

Edit Notebook

GitHub repos (https://github.com/drivetrainhub/notebooks/) are used to publicly host our notebooks, allowing anyone to view and propose edits.

References

1. Gears and Gear Drives, 1st Edition. Damir Jelaska (https://www.wiley.com/en-us/Gears+and+Gear+Drives-p-9781119941309)
2. Handbook of Practical Gear Design and Manufacture, 1st Edition. Darle W. Dudley