Homework 1

Wednesday, January 7, 2026 2:47 PM

1) Roller Chain
   a. Used for transmitting power from one sprocket to another (even multiple others). Links different systems together. There is no slippage or creep since its solid metal links, leading to long life.
   b. Vendor-usarollerchain.com
   c. They have standard chains, ANSI or Metric. They also have double chains, corrosion resistant chains, all sorted by pitch, width, pin sizing, plate sizing, etc.
2) McMaster-Carr
   a. Low Carbon

Dia.	Dia. Tolerance	Yield Strength, psi	Fabrication	Hardness	Max. Hardness After Heat Treatment	Specifications Met		1/2 ft. Lg.	1 ft. Lg.	2 ft. Lg.	3 ft. Lg.	4 ft. Lg.	6 ft. Lg.	8 ft. Lg.	12 ft. Lg.
1"	-0.002" to 0"	54,000	Cold Worked	Rockwell B70 (Medium)	Rockwell C60	ASTM A108	8920K231	—	9.77	16.88	24.43	32.42	44.41	55.51	75.50

   b. Tight Tolerance

Dia.	Dia. Tolerance	Mechanical Finish	Straightness Tolerance		1/2 ft. Lg.	1 ft. Lg.	2 ft. Lg.	3 ft. Lg.	4 ft. Lg.	6 ft. Lg.	8 ft. Lg.	12 ft. Lg.
a. 1"	-0.0005" to 0"	Precision Ground	1/16" per 3 ft.	5227T322	$21.06	35.10	56.17	81.44	102.51	140.42	189.57	—

   c. It looks to me that the Tight tolerance is significantly pricier. This would affect my decision making because I would take into account the tolerance that I want for the part or machine and make sure that It is right.
3) Shigley's 2.25

#3) rod in tensile force

Tungsten carbide

Zinc alloy

Polycarb

Aluminum alloy

use Ashby chart

Performance Metrics

looking to ↑ material strength

↓ weight

P_1 = weight

$= A \cdot l \cdot \rho$ (1)

$P_1 =$ (Functional req)(Geometry req)(Material properties)

Tensile stress eq = $\sigma = \frac{F}{A}$

At failure by strength (yield)

$S = \frac{F}{A}$ where S is yield strength

$A = \frac{F}{S}$ (2)

Sub (2) → (1)

$P_1 = \frac{F}{S} l \rho = (F)(l)\left(\frac{\rho}{S}\right)$

how to have low weight?

min F X can't do

min l x can't do

$\frac{\rho}{S}$ ✓ look into this

Minimize $\frac{\rho}{S}$ or Maximize $\frac{S}{\rho}$

Aluminum Alloy for lightweight rod limited by stress

HW 4

Cantilever beam

failure by deflection
min weight

Fig 2-24 pg 82

$P = A l \rho$

constrained by stiffness

$$k = \frac{F}{\delta} = \frac{P}{\frac{PL^3}{3EI}} = \frac{3EI}{L^3}$$

$F = k\delta$

$I = cA^2$

$$k = \frac{3EcA^2}{L^3} \quad \text{or} \quad \frac{kL^3}{3Ec} = A^2$$

$$P_1 = \sqrt{\frac{kL^3}{3Ec}} \cdot l \cdot \rho$$

$$P_1 = (\ k\ )\left(\ \sqrt{L^3 \cdot l}\ \right)\left(\ \frac{\rho}{\sqrt{3E \cdot c}}\ \right)$$

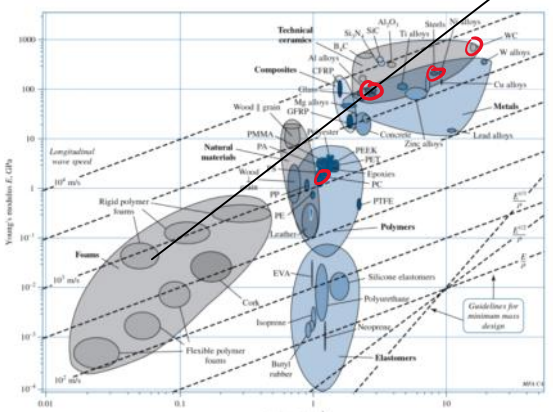
either ↑ stiffness
or
↓ density

Looking at Ashby charts for tungsten carbide, high-carbon heat-treated steel, polycarbonate polymer, and aluminum alloy.

Polycarb = highest

WC and steel ≈ same

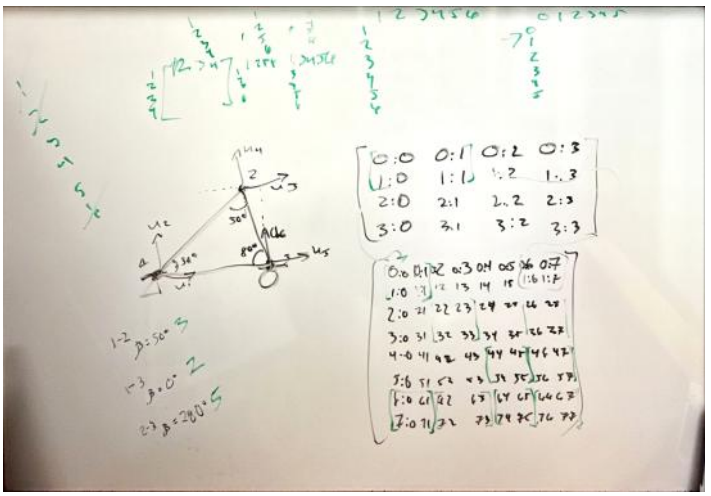
Al alloys lowest



Direct Stiffness 2 DOF Truss Element

Total global stiffness matrix K:
[[3.2395 1.4772 -1.2395 -1.4772 -2.   0.   ]
 [1.4772 1.7605 -1.4772 -1.7605 0.   0.   ]
 [-1.2395 -1.4772 1.3903 0.6222 -0.1508 0.8551]
 [-1.4772 -1.7605 0.6222 6.6097 0.8551 -4.8492]
 [-2.   0.   -0.1508 0.8551 2.1508 -0.8551]
 [0.   0.   0.8551 -4.8492 -0.8551 4.8492]]

Non-zero entries in K_reduced:
[[1.3903 0.6222 -0.1508]
 [0.6222 6.6097 0.8551]
 [-0.1508 0.8551 2.1508]]



```python
import numpy as np

#define coefficients and angles in radians
coeff = np.array([3, 2, 5])
beta = np.deg2rad(np.array([50, 0, 280]))
#function to calculate global stiffness matrix for each element
def calc_stiffness_matrix(coeff, beta):
    """Calculate the global stiffness matrices for structural elements.
    This function computes the stiffness matrix for each element in a structural
    system, transforming local element stiffness matrices to the global
coordinate
    system using rotation angles.
    :param coeff: List of stiffness coefficients (EA/L) for each element
    :type coeff: list of float
    :param beta: List of orientation angles (in radians) for each element
    :type beta: list of float
    :return: List of global stiffness matrices (4x4 numpy arrays), one per
element
    :rtype: list of numpy.ndarray
    Example:
        >>> coeff = [1000, 1500]
        >>> beta = [0, .7854]
        >>> k_matrices = calc_global_stiffness_matrix(coeff, beta)
    """
    k = []
    for i in range(len(coeff)):
        c, s = np.cos(beta[i]), np.sin(beta[i])
        k_i = coeff[i] * np.array([[c**2, c*s, -c**2, -c*s],
                                   [c*s, s**2, -c*s, -s**2],
                                   [-c**2, -c*s, c**2, c*s],
                                   [-c*s, -s**2, c*s, s**2]])
        k.append(k_i)
    return k
k_matrices = calc_stiffness_matrix(coeff, beta)

def global_stiffness_matrix(k_matrices):
    """Assemble the total global stiffness matrix from individual element
stiffness matrices.
    This function combines the stiffness matrices of individual structural
elements into a
    single global stiffness matrix, taking into account the connectivity of the
elements.
    :param k_matrices: List of global stiffness matrices (4x4 numpy arrays) for
each element
    :type k_matrices: list of numpy.ndarray
    :return: Total global stiffness matrix
    :rtype: numpy.ndarray
```

```
    Example:
        >>> k_matrices = [k1, k2]
        >>> K_total = assemble_global_stiffness_matrix(k_matrices)
    """
    # map each sub matrix to an expanded 8x8 matrix
    K_total = np.zeros((6, 6))
    for i, k in enumerate(k_matrices):
        # Split k into 4 quadrants
        k_1 = k[0:2, 0:2]  # top left
        k_2 = k[0:2, 2:4]  # top right
        k_3 = k[2:4, 0:2]  # bottom left
        k_4 = k[2:4, 2:4]  # bottom right
        if i == 0:
            K_total[0:2, 0:2] += k_1
            K_total[0:2, 2:4] += k_2
            K_total[2:4, 0:2] += k_3
            K_total[2:4, 2:4] += k_4
        elif i == 1:
            K_total[0:2, 0:2] += k_1
            K_total[0:2, 4:6] += k_2
            K_total[4:6, 0:2] += k_3
            K_total[4:6, 4:6] += k_4
        elif i == 2:
            K_total[2:4, 2:4] += k_1
            K_total[2:4, 4:6] += k_2
            K_total[4:6, 2:4] += k_3
            K_total[4:6, 4:6] += k_4
    return K_total

K_global = global_stiffness_matrix(k_matrices)
np.set_printoptions(precision=4, suppress=True, linewidth=100)
print(f"Total global stiffness matrix K:\n{K_global}")

#reduced global stiffness matrix
k_boundary = np.zeros((6, 6))
k_boundary[2:5, 2:5] = 1
K_reduced = K_global * k_boundary
# print(f"\nReduced global stiffness matrix K_reduced:\n{K_reduced}")
non_zero_mask = K_reduced != 0
non_zero_indices = np.where(non_zero_mask)
min_row, max_row = non_zero_indices[0].min(), non_zero_indices[0].max()
min_col, max_col = non_zero_indices[1].min(), non_zero_indices[1].max()
K_reduced_trimmed = K_reduced[min_row:max_row+1, min_col:max_col+1]
print(f"\nNon-zero entries in K_reduced:\n{K_reduced_trimmed}")
```