Laporan Tugas Akhir Menentukan Damper Rate dan Spring Rate Kendaran

Spring Rate and Damper Rate Calculation of a Vehicle Using Numerical Method

Adamsyah Haryo Saksono 2506565401

Universitas Indonesia

Abstract

kendaraan merupakan sistem transportasi yang sangat penting bagi keberlangsungan hidup manusia. tanpa adanya kendaraan berpergian menjadi sangat susah. agar perjalanan kendaraan menjadi aman dan nyaman diperlukan adanya sistem suspensi yang baik. beberapa cara untuk mengontorl suspensi adalah mengubah spring rate dan damper rate. maka dari itu research ini mencoba untuk mencari spring rate dan damper rate pada kendaraan. menetukan spring rate dan damper rate pada percobaan ini menggunakan metode numerik Newmark beta. dengan digunakannya metode ini spring rate dan damper rate yang cocok untuk digunakan dapat diketahui serta didapatkannya plot dari pergerakan displacement suspensi untuk simulasi pergerakan suspensi terhadap kontur jalan

E. Author Declaration

1. Deep Awareness (of) I

Tuhan yang maha esa adalah yang mengetahui segalanya. tidak ada apapun yang lebih besar dari dia. apapun yang terjadi di muka bumi ini adalah atas kehendaknya.

2. Niat Kegiatan Proyek

penulis membuat suatu sistem untuk menghitung spring rate dan damper rate yang tepat untuk suatu kendaraan dengan menggunakan metode numerik Newmar beta. dengan adanya sistem ini diharapkan dapat digunakan untuk mendapatkan kendaraan yang lebih aman dan nyaman untuk penggunanya.

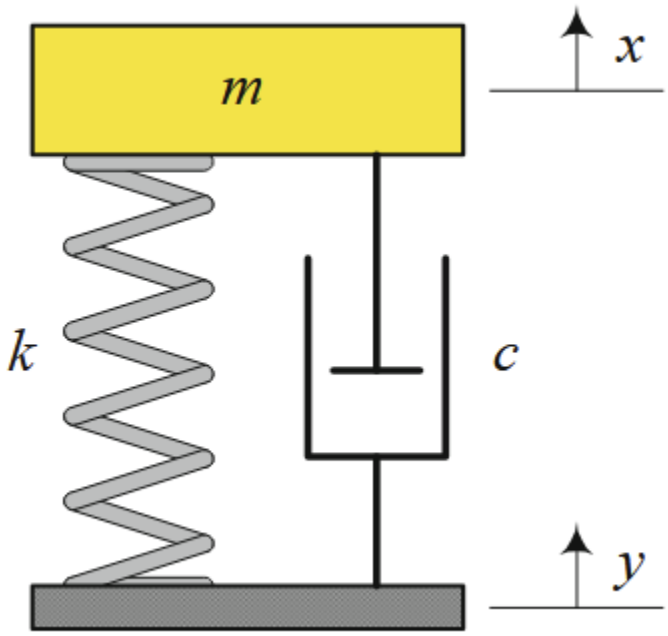
F. Introduction

(include detailed Initial (thinking about the problem)

Sistem suspensi kendaraan merupakan suatu problem yang sangat kompleks sehingga memerlukan proses desain dan perhitungan yang panjang jika dilakukan tanpa komputer ataupun metode numerik. salah satu masalah yang paling sering ditemukan adalah menentukan spring rate dan damper rate suatu kendaraan. untuk menemukan spring rate dan damper rate sebelumnya digunakan metode numerik newmark beta, galerkin, finite difference method oleh Krishnanunni C.G. pada tahun 2019. untuk mempersingkat waktu

komputasi disini dicoba untuk menggunakan hanya newmark beta sehingga perhitungan tidak perlu dilakukan menggunakan daya listrik besar pada komputer yang cepat.

G. Methods & Procedures



berdasarkan buku yang ditulis oleh Jazar R. N. yang berujudul “Vehicle Dynamics”. suspensi dapat di ilustrasikan seperti gambar diatas. sistem suspensi terdiri dari spring(k), damper(c), mass(m), sprung mass(x) dan unsprung mass(y). untuk menghitung pergerakan suspensi digunakan rumus berikut

$$m\ddot{x} + c\dot{x} + kx = c\dot{y} + ky$$

yang di transformasi menjadi:

$$m\ddot{z} + c\dot{z} + kz = -m\ddot{y}$$

dengan z sebagai displacement

$$z = x - y$$

Energi kinetik, energi potential dan disipasi energi dari sistem suspensi dihitung dengan:

$$K = \frac{1}{2}m\dot{x}^2$$
$$V = \frac{1}{2}k\left(x-y\right)^2$$
$$D = \frac{1}{2}c\left(\dot{x}-\dot{y}\right)^2$$

Computational Thinking Framework

pada tugas besar ini digunakan computational thinking framework yang berupa

Decomposition: Pada tahap ini terdapat deskripsi mengenai apa saja komponen-komponen yang berpengaruh pada pemilihan spring rate dan damper rate kendaraan. seperti:

sprung mass

unsprung mass

tinggi road bump

panjang road bump

kecepatan kendaraan

tire stiffness

frekuensi natural dari sprung mass

Pattern recognition: Tahap ini digunakan untuk menemukan pengaruh dari setiap elemen. seperti apa pengaruh spring rate dan damper rate itu sendiri. spring digunakan untuk meredam getaran dari jalan agar tidak tersalur ke badan kendaraan dan damper untuk mengurangi pergerakan dari spring.

Abstraction: pada tahap ini diasumsikan bahwa suspensi dibuat menjadi 1 ban saja dan secara 2 dimensi atau disebut sebagai quarter car model.

Algorithm design: Algoritma yang akan digunakan pada perhitungan ini adalah Newmark Beta Method.

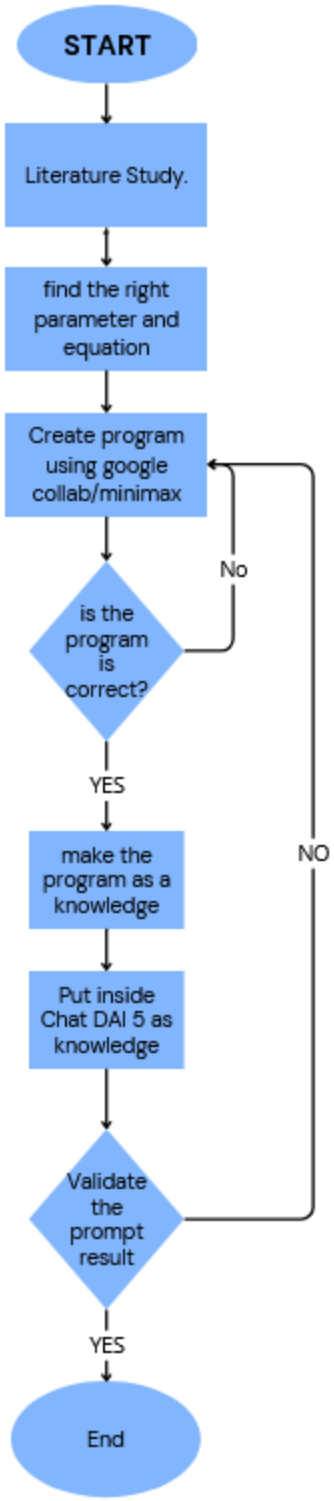
$X\dot{\rightarrow}t+\Delta t=X\dot{\rightarrow}t+(1-\gamma)\Delta tX\ddot{\rightarrow}t+\Delta t\gamma X\ddot{\rightarrow}t+\Delta t$ eq.1

$X\rightarrow t+\Delta t=X\rightarrow t+\Delta tX\dot{\rightarrow}t+(12-\beta)(\Delta t)2X\ddot{\rightarrow}t+\beta(\Delta t)2X\ddot{\rightarrow}t+\Delta t$ eq.2

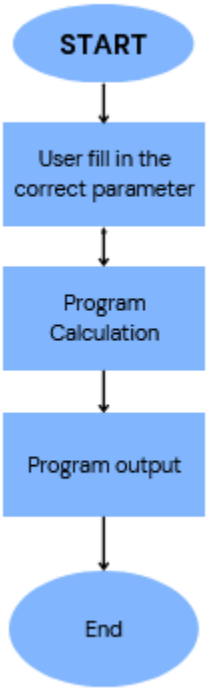
pada metode ini kedua equation digunakan untuk menghitung velocity(eq.1) dan displacement(eq.2) pada suspensi. numerical method ini digunakan karena lebih ringan dari runge kutta, HHT alpha, dan wilson theta. namun masih memiliki hasil yang akurat tanpa memerlukan proses komputasi yang berat.

https://www.sciencedirect.com/topics/engineering/newmark-method

Flowchart pembuatan ptogram

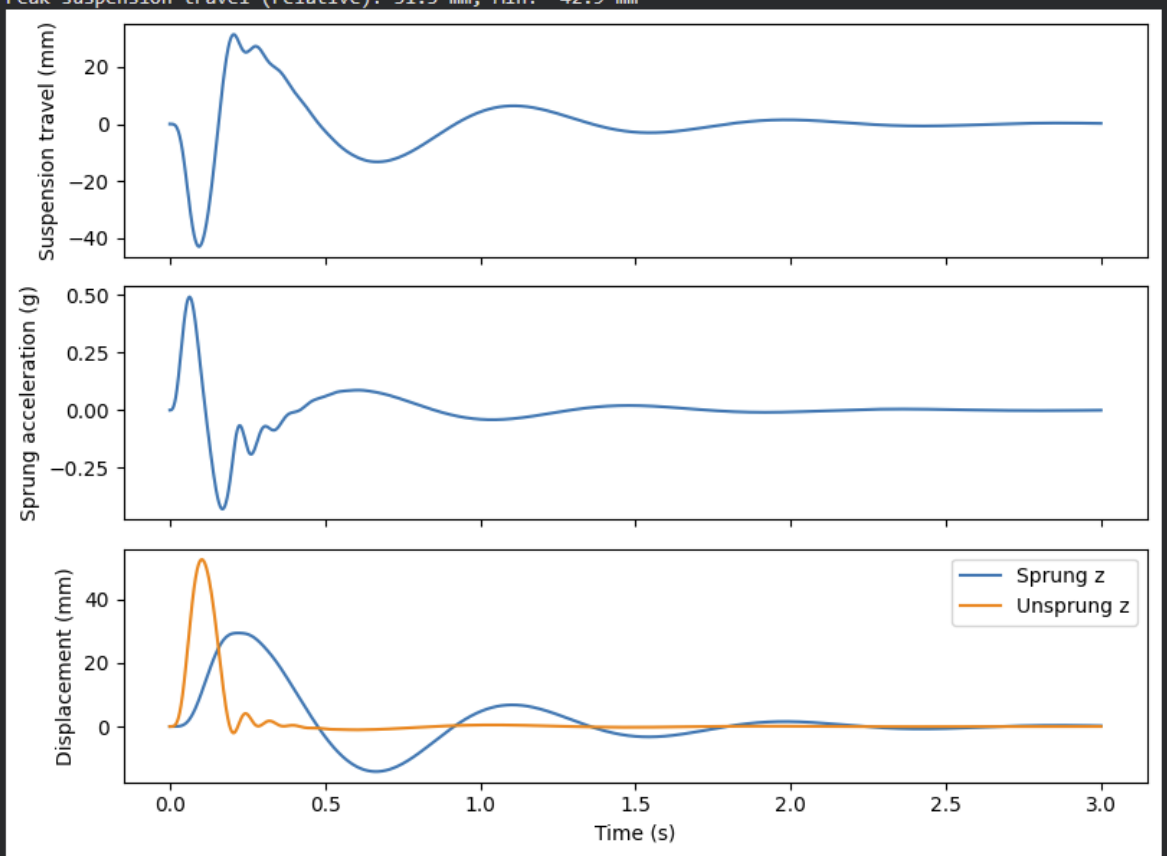

START
Literature Study.
find the right parameter and equation
Create program using google collab/minimax
is the program is correct?
No
YES
make the program as a knowledge
Put inside Chat DAI 5 as knowledge
Validate the prompt result
NO
YES
End

Program flowchart



H. Results & Discussion

```
=== Quarter-Car Newmark Simulation (Half-Sine Bump) ===
Sprung Mass Spring Rate (k_s): 17054.68 N/m
Sprung Mass Damping Rate (c_s): 1130.97 N*s/m
Peak sprung acceleration: 4.82 m/s^2 (~0.49 g)
RMS sprung acceleration over 0–1 s: 1.51 m/s^2
Peak suspension travel (relative): 31.3 mm; Min: -42.9 mm
```



Pada kasus ini digunakan simulasi quarter car model. Hasil dari program yang telah dibuat menunjukan bahwa metode numerik ini bisa digunakan untuk mengetahui spring rate dan damper rate yang dibutuhkan pada kendaraan dengan berat 300 Kg.

I. Conclusion and Recommendation

Metode Newmark Beta bisa digunakan untuk menemukan spring rate dan damper rate yang tepat pada kendaraan. untuk metode ini sebaiknya hanya digunakan ketika komputer yang akan digunakan untuk komputasi memang sudah dipastikan tidak mampu untuk menggunakan metode numerik lainnya.

I. Acknowledgments

Terimakasih kepada Prof. Ir. Ahmad Indra Siswantara, Ph. D selaku dosen komputasi teknik kemudian bang Epsi, Bena, Resya, dan Febryan yang membantu membuat tugas ini

J. (References) Literature Cited _____

Outline Explanation:

I. Conclusion, Closing Remarks, Recommendations

Summarize the key outcomes and their significance. Offer actionable recommendations or next steps for further work.

J. Acknowledgments

Recognize contributions and support from individuals, institutions, or funding sources.

K. (References) Literature Cited

Krishnanunni, C. G., & Rao, B. N. (2019). *Decoupled technique for dynamic response of vehicle-pavement systems*. *Engineering Structures, 191*, 264–279. https://doi.org/10.1016/j.engstruct.2019.04.042

Jazar, R. N. (2014). *Vehicle dynamics: Theory and application* (2nd ed.). Springer. https://doi.org/10.1007/978-1-4614-8544-5

L. Appendices

```
import numpy as np
import matplotlib.pyplot as plt


def run_quarter_car_newmark(
    m_s=300.0,       # Sprung mass (body mass) in kg
    m_u=40.0,        # Unsprung mass (wheel assembly mass) in kg
    f_n_s=1.2,       # Natural frequency of the sprung mass in Hz
    zeta_s=0.25,     # Damping ratio of the sprung mass
    k_tire=250000.0,  # Tire stiffness in N/m (rigid tire assumption)
    h_b=0.05,        # Height of the road bump in meters
    L_b=1.0,         # Length of the road bump in the travel direction in meters
    v_veh=5.0,       # Vehicle speed in m/s
    t_end=3.0        # Total simulation time in seconds
):
    """
    Quarter-car suspension simulation using Newmark-beta method with a half-sine road bump.
    Returns:
        t: time array
        xs: sprung displacement [m]
        xsd: sprung velocity [m/s]
```

```
    xu: unsprung displacement [m]
    xud: unsprung velocity [m/s]
    as_rel: suspension travel relative displacement (z_s - z_u) [m]
    accel_s: sprung absolute acceleration [m/s^2]
    k_s: sprung mass spring rate [N/m]
    c_s: sprung mass damping rate [N*s/m]
"""
# Derived parameters
wn_s = 2*np.pi*f_n_s
k_s = m_s * wn_s**2                # N/m
c_s = 2*zeta_s*np.sqrt(k_s*m_s)       # N*s/m

M = np.array([[m_s, 0.0],
          [0.0, m_u]])
C = np.array([[ c_s, -c_s],
          [-c_s,  c_s]])
K = np.array([[ k_s,    -k_s],
          [-k_s,  k_s + k_tire]])

# Time settings
T_bump = L_b / v_veh               # time to traverse bump rise/fall half-sine path
dt = min(0.001, T_bump/100.0)         # small step for stability/accuracy
t = np.arange(0.0, t_end+dt*0.5, dt)

# Road input functions (half-sine)
def road_z(t):
    if 0 <= t < T_bump:
        return 0.5 * h_b * (1 - np.cos(2*np.pi*t/T_bump))
    else:
        return 0.0

def road_zdot(t):
    if 0 <= t < T_bump:
        return (h_b * np.pi / T_bump) * np.sin(2*np.pi*t/T_bump)
    else:
        return 0.0

# Newmark parameters (gamma = 1/2, beta = 1/4 -> average acceleration method,
```

```
unconditionally stable)
    beta = 1/4
    gamma = 1/2
    dt_sq = dt**2

    # Newmark coefficients for direct integration
    a0 = 1.0 / (beta * dt_sq)
    a1 = gamma / (beta * dt)
    a2 = 1.0 / (beta * dt)
    a3 = (1.0 / (2.0 * beta)) - 1.0
    a4 = (gamma / beta) - 1.0
    a5 = dt * (gamma / (2.0 * beta) - 1.0)

    # Initialization: static equilibrium at t=0 (r(0)=0)
    r0 = road_z(0.0)
    x0 = np.array([r0, r0], dtype=float)        # z_s=z_u=r
    v0 = np.zeros_like(x0)

    M_inv = np.linalg.inv(M)
    P0  = np.array([0.0, k_tire*r0])

    # Initial acceleration at t=0
    acc0 = M_inv @ (P0 - (C @ v0 + K @ x0))

    # Allocate arrays
    nsteps = len(t)
    xs = np.zeros(nsteps)
    xsd= np.zeros(nsteps)
    xu = np.zeros(nsteps)
    xud= np.zeros(nsteps)
    accs = np.zeros((nsteps, 2)) # Store full acceleration vector for x_s and x_u

    as_rel = np.zeros(nsteps)  # z_s - z_u
    accel_s= np.zeros(nsteps)

    # Assign initial conditions (t=0)
    xs[0] = x0[0]
    xsd[0]= v0[0]
```

```
xu[0] = x0[1]
xud[0]= v0[1]
as_rel[0] = xs[0]-xu[0]
accs[0] = acc0 # Store initial acceleration
accel_s[0]= acc0[0]

# Effective stiffness matrix for Newmark direct integration
K_eff = K + a0*M + a1*C

# Time stepping
for i in range(1, nsteps):
  # Values at previous time step (n)
  xn   = np.array([xs[i-1], xu[i-1]], dtype=float)
  vxn  = np.array([xsd[i-1], xud[i-1]], dtype=float)
  accxn = accs[i-1] # Acceleration at previous time step

  # Force at current time step (n+1)
  r_n1 = aroad_z(t[i])
  P_n1 = np.array([0.0, k_tire*r_n1])

  # Calculate effective force P_eff for Newmark direct integration
  # P_eff = P_n1 + M @ (c0*xn + c2*vxn + c3*accxn) + C @ (c1*xn + c4*vxn + c5*accxn)
  P_eff = P_n1 + M @ (a0*xn + a2*vxn + a3*accxn) + C @ (a1*xn + a4*vxn + a5*accxn)

  # Solve for displacement at n+1
  xnp1 = np.linalg.solve(K_eff, P_eff)

  # Calculate acceleration at n+1
  acc_np1 = a0*(xnp1 - xn) - a2*vxn - a3*accxn

  # Calculate velocity at n+1 (using average acceleration method for gamma=0.5)
  # v_np1 = v_n + dt*((1-gamma)*acc_n + gamma*acc_np1)
  vnp1 = vxn + dt*0.5*(accxn + acc_np1)

  # Store results for current time step (n+1)
  xs[i]  = xnp1[0]
  xu[i]  = xnp1[1]
  xsd[i]  = vnp1[0]
```

```
        xud[i]  = vnp1[1]
        accs[i] = acc_np1 # Store for next iteration

        as_rel[i]   = xs[i] - xu[i]
        accel_s[i]  = acc_np1[0] # Sprung absolute acceleration

    return t, xs, xsd, xu, xud, as_rel, accel_s, k_s, c_s

if __name__ == "__main__":
    # Run with default parameters
    t, zs, zsd, zu, zud, as_rel, accel_s, k_s, c_s = run_quarter_car_newmark()

    print("=== Quarter-Car Newmark Simulation (Half-Sine Bump) ===")
    print(f"Sprung Mass Spring Rate (k_s): {k_s:.2f} N/m")
    print(f"Sprung Mass Damping Rate (c_s): {c_s:.2f} N*s/m")
    print(f"Peak sprung acceleration: {np.max(np.abs(accel_s)):.2f} m/s^2
(~{np.max(np.abs(accel_s))/9.81:.2f} g)")
    print(f"RMS sprung acceleration over 0–1 s: {np.sqrt(np.mean((accel_s[t<=1.0])**2)):.2f}
m/s^2")
    print(f"Peak suspension travel (relative): {np.max(as_rel)*1000:.1f} mm; Min:
{np.min(as_rel)*1000:.1f} mm")

    # Simple plot
    fig, axes = plt.subplots(3, 1, figsize=(8,6), sharex=True)
    axes[0].plot(t, as_rel*1000); axes[0].set_ylabel("Suspension travel (mm)")
    axes[1].plot(t, accel_s/9.81); axes[1].set_ylabel("Sprung acceleration (g)")
    axes[2].plot(t, zs*1000, label="Sprung z"); axes[2].plot(t, zu*1000, label="Unsprung z")
    axes[2].set_ylabel("Displacement (mm)"); axes[2].legend()
    plt.xlabel("Time (s)")
    plt.tight_layout(); plt.show()
```