(15 pt)Rocket Movement

The equations that describe the motion of a rocket during vertical flight are provided by the following ODE:

$$(m_s + m_p)\frac{d^2h}{dt^2} = -(m_s + m_p)g + \dot{m}_p v_e - \frac{1}{2}\rho\frac{dh}{dt}\left|\frac{dh}{dt}\right| AC_D$$

h is the altitude of the rocket

$m_s = 50kg$ is the weight of the rocket shell

$g = 9.81\frac{m}{s^2}$

$\rho = 1.0\frac{kg}{m^3}$ is the average air density (assumed constant throughout flight)

$A = \pi r^2$ is the maximum cross sectional area of the rocket, where $r = 0.5m$

$v_e = 325\frac{m}{s}$ is the exhaust speed

$C_D = 0.2$ is the drag coefficient

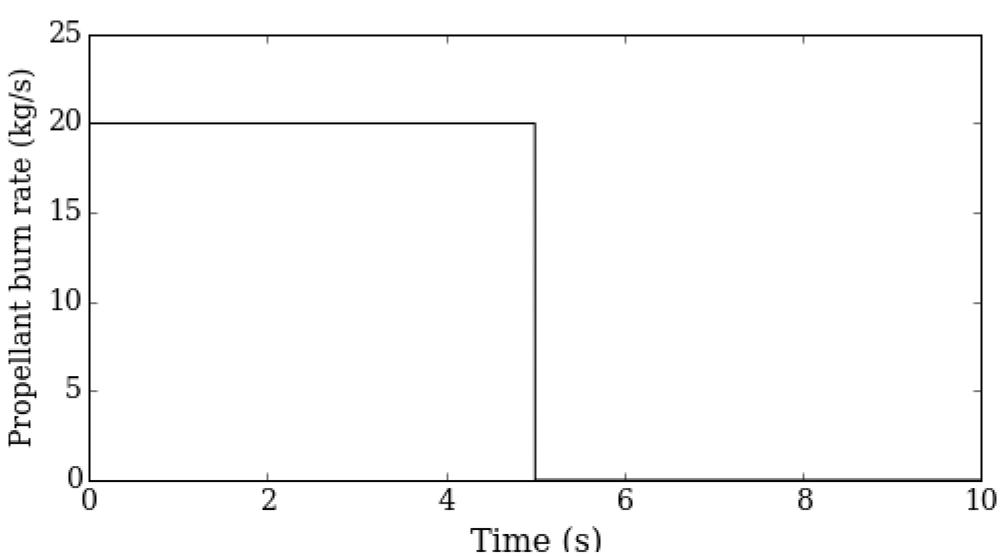
$m_{po} = 100kg$ at time $t = 0$ is the initial weight of the rocket propellant

The mass of the remaining propellant is given by:

$$m_p = m_{po} - \int_0^t \dot{m}_p d\tau$$

where $\dot{m}_p$ is the time-varying burn rate given by the following figure:

Propellant Burn Rate



1) Convert this into a first order ODE system. You must include the ODE system in your submission pdf.

2) Using any numerical ODE method taught in class, create a program to simulate the altitude and velocity of the rocket from launch until crash down. Ensure that reasonble step sizes are chose to obtain a good

simulation result. Plot two things: Altitude of Rocket Over Time and Velocity of Rocket Over Time. You must include these two plots in your submission pdf.

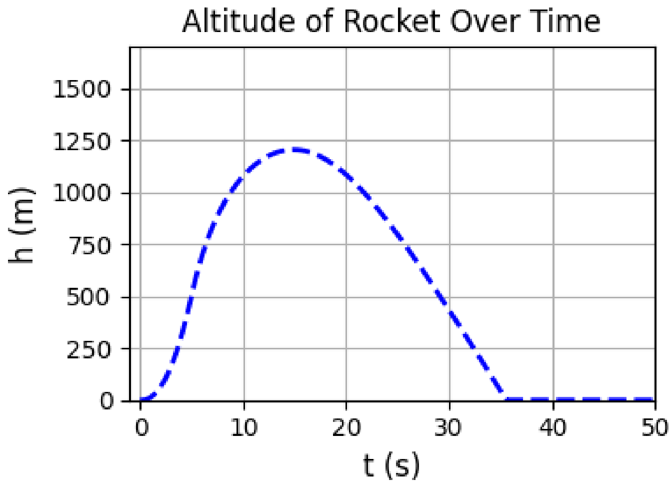
3) Answer the following four questions:

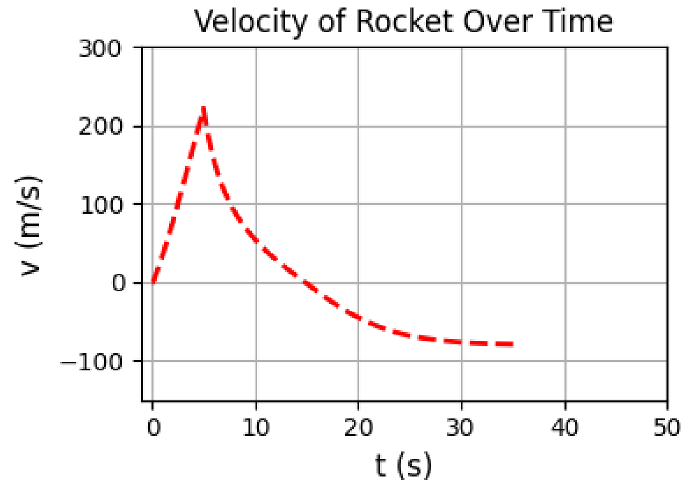
1. (1pt) At $t = 3.6$ seconds, what is the remaining mass of the rocket propellant (in kg)?
2. (3pt) What is the rocket's maximum speed (in $\frac{m}{s}$)?
   - At what time does this maximum speed occur (in seconds)?
   - What is the rocket's altitude at that time (in meters)?
3. (2pt) What is the maximum altitude reached by the rocket during its flight (in meters)?
   - At what time, measured from the launch, does this maximum altitude occur (in seconds)?
4. (2pt) When does the rocket hit the ground (in seconds)?
   - What is the rocket's velocity when it hits the ground (in $\frac{m}{s}$)?

You may use either LaTeX or Word or anything else that allows you to type math. Please submit a single PDF file. No report required, just inlclude the equations of the ODE system (3 pt), the two plots (2pt each), and answers to the 4 questions above.

In [29]:
```
from math import sin, cos, log, ceil
import numpy as np
import sympy
from matplotlib import pyplot as plt
%matplotlib inline
from matplotlib import rcParams



# Model parameters
g = 9.81           # Acceleration due to gravity in m/s^2
ms = 50            # Mass of the rocket shell in kg
rho = 1.0          # Average air density in kg/m^3
r = 0.5            # Maximum radius of the rocket cross section in meters
A = np.pi * r**2 # Maximum cross-sectional area of the rocket in m^2
ve = 325           # Exhaust velocity in m/s
CD = 0.2           # Drag coefficient

# Initial conditions
mp0 = 100          # Initial mass of the rocket propellant in kg
h0 = 0             # Initial height of the rocket in meters
v0 = 0             # Initial velocity of the rocket in m/s
a0 = 0             # Initial acceleration of the rocket in m/s^2

# Time parameters
T = 100                             # Final simulation time in seconds
dt = 0.1                            # Time step in seconds
t = np.arange(0, T + dt, dt)        # Time discretization array
N = len(t)                          # Number of time steps
mp_int = 20 * dt * 0.5              # Integrated mass flow rate over half a time step

# Initialize the solution array
u = np.empty((N, 2))                # Array to store height and velocity at each time step
u[0] = np.array([h0, v0])           # Set initial conditions (height, velocity) at t=0


# Function defining the system of equations
def f(u):
    h = u[0]  # Altitude
    v = u[1]  # Velocity
```

```python
    return np.array([
        v,  # The rate of change of altitude is velocity
        (-(ms + mp) * g + mp_dot * ve - 0.5 * rho * v * np.abs(v) * A * CD) / (ms + mp)
    ])

# Euler method to advance the solution by one time step
def euler_step(u, f, dt):
    return u + dt * f(u)

# Function to determine the rate of change of propellant mass
def dot_mp(t):
    if t < 5:
        return 20  # Constant propellant burn rate for the first 5 seconds
    else:
        return 0   # No propellant burn after 5 seconds

# Time loop - Euler Method
for i, tt in enumerate(t[1:(N-1)]):
    n = i + 1

    # Calculate the current mass flow rate of the propellant
    mp_dot = dot_mp(tt)

    # Update the integrated propellant mass and the remaining propellant mass
    mp_int = mp_int + mp_dot * dt
    mp = mp0 - mp_int

    # Use the Euler method to compute the new state (altitude and velocity)
    u[n+1] = euler_step(u[n], f, dt)

# Extract altitude and velocity for plotting
h = u[:, 0]
v = u[:, 1]
# After it reaches the ground, altitute and speed stays 0
t_impact = np.where(h < 0)[0][0]
v[t_impact+1:] = np.nan
h[t_impact+1:] = 0

# Create a figure with 1 row and 2 columns of subplots
fig, axs = plt.subplots(1, 2, figsize=(8, 3))  # figsize controls the size of the entire

# Plotting the altitude over time on the first subplot (left)
axs[0].grid(True)
axs[0].set_title('Altitude of Rocket Over Time')
axs[0].set_xlabel(r't (s)', fontsize=12)
axs[0].set_ylabel(r'h (m)', fontsize=12)
axs[0].plot(t, h, 'b--', lw=2)
axs[0].set_xlim(-1, 50)
axs[0].set_ylim(0, 1700)

# Plotting the velocity over time on the second subplot (right)
axs[1].grid(True)
axs[1].set_title('Velocity of Rocket Over Time')
axs[1].set_xlabel(r't (s)', fontsize=12)
axs[1].set_ylabel(r'v (m/s)', fontsize=12)
axs[1].plot(t, v, 'r--', lw=2)
axs[1].set_xlim(-1, 50)
axs[1].set_ylim(-150, 300)

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```





In [30]:
```
T1 =  3.6                     # final time
dt1 = 0.1                     # time increment
t1 = np.arange(0,T1,dt1)     # time discretization
N1 = len(t1)
mp_int1 = 0
mp01= 100

#time loop
for i1, tt1 in enumerate(t1):
    mp_dot1 = dot_mp(tt1)
    mp_int1 = mp_int1 + mp_dot1*dt1
    mp1 = mp01 - mp_int1


print("Q1: %.2f kg of rocket propellant remain at 3.2s" %mp1)
```

```
Q1: 28.00 kg of rocket propellant remain at 3.2s
```

In [31]:
```
maxv = max(v)
vmax_loc = np.where(v == maxv)[0][0]
"Q2: Maximum Velocity, Time, Altitude: (%.2f m/s, %s s, %.2f m)"  %(maxv, t[vmax_loc], h
```

Out[31]:
```
'Q2: Maximum Velocity, Time, Altitude: (221.71 m/s, 5.0 s, 503.50 m)'
```

In [32]:
```
maxh = max(h)
hmax_loc = np.where(h == maxh)[0][0]
"Q3: Maximum Height, Time: (%.2f m, %s s)"  %(maxh, t[hmax_loc])
```

Out[32]:
```
'Q3: Maximum Height, Time: (1205.11 m, 14.9 s)'
```

In [33]:
```
"Time of Impact and Velocity: (%.2f s, %.2f m/s)" %(t[t_impact], v[t_impact])
```

Out[33]:
```
'Time of Impact and Velocity: (35.60 s, -78.89 m/s)'
```

Part 2 Written

1. (1pt) Use Newton's method to write down an iterative formula for finding the root of $f(x) = x^3 - a$ for any constant a. If you start with the initial guess $x_0 = \frac{1}{3}a$, then what is x_1?

2. (1pt) The root of $x^3 - 2$ is $\sqrt[3]{2}$, which is located in the interval $[1, 2]$. If we use the bisection method to find this root, starting with the endpoints $a = 1$ and $b = 2$, then what is the worst case error in our estimate for the root after 10 steps?

3. (0.5pt each) To find the interpolating polynomial for the points $(x, y) = (0, 0), (1, 1)$, and $(4, 2)$, write down the 3 Lagrange cardinal functions $l_i(x)$.

4. (0.5pt) Write down the interpolating polynomial in 3?

5. (1pt) Write down the Vandermonde matrix system for these same points $(0, 0), (1, 1), (4, 2)$ to find the coefficients of the interpolating polynomial in the standard basis. You don't need to solve the Vandermonde matrix system.

6. (1.5pt) The formula for Simpson's method (not composite) with only 3 points on an interval is

$$\int_a^b f(x)dx \approx \frac{h}{3}(f(a) + 4f(m) + f(b))$$

where $h = \frac{b-a}{2}$ and $m = \frac{a+b}{2}$. Use this rule to approximate area under the function $y = e^x$ on the interval $[0, 2]$ and estimate the error in the approximation.

7. (1.5pt) Consider the numerical quadrature rule to approximate $\int_0^1 f(x)dx$ given by $\int_0^1 f(x)dx \approx w_1 f(0) + w_2 f(x_1)$. Find the highest possible degree of precision for this rule and find the values of the unknowns in this quadrature rule, namely w_1, w_2, and x_1. Approximate $\int_0^1 x^3 dx$ with the quadrature rule found.

8. (2pt) Find the condition number of the matrix: $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

9. (3pt) The SOR iteration for solving $Ax = b$ can be written as:

$$x^{k+1} = (D + wL)^{-1}b + (D + wL)^{-1}[(1 - w)D - wU]x^k = y_{\text{SOR}} + M_{\text{SOR}}\, x^k$$

where

$$y_{SOR} = (D + wL)^{-1}b, \quad M_{SOR} = (D + wL)^{-1}[(1 - w)D - wU].$$

If

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix},$$

and $w = 1.2$, what is M_{SOR}? What is the error bound (in ℓ_2) at the k-th iteration given that the error of the initial guess, $\|e^0\|$, is 1.

10. (1pt) Convert this ODE:

$$u'' + 4uu' + t^2u + t = 0, \quad u(0) = 1, \quad u'(0) = 2$$

into a system of first order ODEs.

11. (2pt) Derive the shooting method for

$$y''(x) = u(x) + v(x)y(x) + w(x)y'(x), \quad y(a) = \alpha, \quad y'(b) = \beta.$$

Namely, when we express the solution as

$$y(x)=\lambda \cdot \bar{y}(x)+(1-\lambda) \cdot \tilde{y}(x),$$

where

$$\bar{y}''(x)=u(x)+v(x)\bar{y}(x)+w(x)\bar{y}'(x), \quad \bar{y}(a)=\alpha, \quad \bar{y}'(a)=0$$

$$\tilde{y}''(x)=u(x)+v(x)\tilde{y}(x)+w(x)\tilde{y}'(x), \quad \tilde{y}(a)=\alpha, \quad \tilde{y}'(a)=1,$$

what is λ?

12. (1pt) Given

$$\frac{d^2 f}{dx^2}=6x-0.5x^2, f(0)=0, f(12)=0$$

the value of $\frac{d^2 f}{dx^2}$ at $f(4)$ using the finite difference method and a step size of $h=4$ can be approximated by

(A) $\frac{f(8)-f(0)}{8}$

(B) $\frac{f(8)-2f(4)+f(0)}{16}$

(C) $\frac{f(12)-2f(8)+fy(4)}{16}$

(D) $\frac{f(4)-f(0)}{4}$

1. (3pt) Given

$$\frac{d^2 y}{dx^2}=6x-0.5x^2, y(0)=0, y(12)=0,$$

what is the value of $y(4)$ using the finite difference method (using the second order central difference like we did in class) and a step size of $h=4$? (Hint: Set up the FDM method $A\vec{y}=\vec{b}$ first, and then solve for $\vec{y}$. A should be 4 by 4 as we have 4 grid points $0,4,8,12$)

Solutions:

1. The Newton's method iteration formula is

$$x_{n+1}=x_n-\frac{f\left(x_n\right)}{f'\left(x_n\right)}.$$

In this case, $f(x)=x^3-a$ and $f'(x)=3x^2$, so

$$x_1=\frac{1}{3}a-\frac{\left(\frac{1}{3}a\right)^3-a}{3\left(\frac{1}{3}a\right)^2}.$$

You can simplify this to

$$x_1=\frac{2}{9}a-\frac{3}{a}.$$

2. Each step in the bisection method reduces the error by a factor of 2. Initially the solution could be anywhere in the interval $[1, 2]$, so our initial error could be as big as 1 (since 1 is the length of the interval). After 10 steps, the worst case error would be

$$\frac{1}{2^{10}} = \frac{1}{1024}.$$

1. $l_0(x) = \frac{1}{4}(x-1)(x-4) \quad l_1(x) = -\frac{1}{3}x(x-4) \quad l_2(x) = \frac{1}{12}x(x-1)$

1. $P(x) = -\frac{1}{3}x(x-4) + \frac{1}{6}x(x-1)$

1. $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 4 & 16 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}.$

1. In this example, $a = 0, b = 2, m = 1$ and $h = 1$. So the approximate area is $\frac{1}{3}\left(1 + 4e + e^2\right) = 6.421$. The error in this approximation is $-\frac{h^5}{90}f^{(4)}(\xi) = -\frac{1}{90}e^{\xi}$. Since e^{ξ} is at most e^2 on the interval, the absolute value of the error is at most $\frac{e^2}{90} = 0.0821$.

2. We want the formula

$$\int_0^1 f(x)dx = w_1 f(0) + w_2 f(x_1)$$

to hold for polynomials $1, x, x^2, \ldots$. We have 3 unknowns, so we need 3 equations. For the equation to hold for $1, x, x^2$, we have:

$$\begin{aligned} f(x) = x^0 &\quad \int_0^1 1dx = x\Big|_0^1 = 1 = w_1 \cdot 1 + w_2 \cdot 1, \\ f(x) = x^1 &\quad \int_0^1 xdx = \frac{x^2}{2}\Big|_0^1 = \frac{1}{2} = w_1 \cdot 0 + w_2 \cdot x_1, \\ f(x) = x^2 &\quad \int_0^1 x^2dx = \frac{x^3}{3}\Big|_0^1 = \frac{1}{3} = w_1 \cdot 0 + w_2 \cdot x_1^2. \end{aligned}$$

We have 3 equations in 3 unknowns:

$$\begin{aligned} w_1 + w_2 &= 1, \\ w_2 x_1 &= \frac{1}{2}, \\ w_2 x_1^2 &= \frac{1}{3}, \end{aligned}$$

or

$$\begin{aligned} w_2 &= 1 - w_1, \\ x_1(1 - w_1) &= \frac{1}{2}, \\ x_1^2(1 - w_1) &= \frac{1}{3}. \end{aligned}$$

Multiplying the second equation by x_1 and subtracting the third equation, we obtain $x_1 = \frac{2}{3}$. Then, $w_2 = \frac{3}{4}$ and $w_1 = \frac{1}{4}$. Thus, the quadrature formula is

$$\int_0^1 f(x)dx = \frac{1}{4}f(0) + \frac{3}{4}f\left(\frac{2}{3}\right)$$

The accuracy/degree of precision of this quadrature formula is $m = 2$, since this formula holds for polynomials $1, x, x^2$. Applying this quadrature rule to approximate $\int_0^1 x^3 dx$:

$$\int_0^1 x^3 dx = \frac{1}{4} \cdot 0 + \frac{3}{4} \cdot \frac{8}{27} = \frac{2}{9} = 0.2222.$$

3.
$$A^T A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 10 & 14 \\ 14 & 20 \end{pmatrix}$$

Solve the characteristic polynomial$\det(A^T A - \lambda I) = 0$:

$$\det \begin{pmatrix} 10-\lambda & 14 \\ 14 & 20-\lambda \end{pmatrix} = (10-\lambda)(20-\lambda) - 14^2$$

$$= \lambda^2 - 30\lambda + 4$$

The eigenvalues are $15 \pm \sqrt{221}$. The singular values are the square roots of the eigenvalues:

$$\sigma_1 = \sqrt{15 - \sqrt{221}} \approx 0.365966$$

$$\sigma_2 = \sqrt{15 + \sqrt{221}} \approx 5.464985$$

Compute Condition Number

$$\frac{5.464985}{0.365966} \approx 14.933$$

1.
$$M_{SOR} = \begin{pmatrix} -0.2 & 0.6 & 0 \\ -0.12 & 0.16 & 0.6 \\ -0.072 & 0.096 & 0.16 \end{pmatrix},$$

and

$$\left\|e^k\right\| \leq \|M_{SOR}\|^k \left\|e^0\right\| = 0.746^k$$

2. By the variable change

$$x_1 = u(t), \quad x_2 = u'(t)$$

we have

$$\begin{cases} x_1' = x_2 \\ x_2' = -4x_1x_2 - t^2x_1 - t \end{cases}$$

with initial conditions

$$\begin{cases} x_1(0) = u(0) = 1 \\ x_2(0) = u'(0) = 2 \end{cases}$$

3. For the boundary condition at $x = b$, we must require

$$y'(b) = \lambda \bar{y}'(b) + (1-\lambda)\tilde{y}'(b) = \beta, \quad \Rightarrow \quad \lambda = \frac{\beta - \tilde{y}'(b)}{\bar{y}'(b) - \tilde{y}'(b)}.$$

4. B

1.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 256 \\ 256 \\ 0 \end{bmatrix}$$

which gives

$$\begin{aligned} &y(0) = y_0 = 0 \\ &y(4) \approx y_1 = -256 \\ &y(8) \approx y_2 = -256 \\ &y(12) = y_3 = 0 \end{aligned}$$

Hence

$$y(4) \approx -256$$

Part 3: BBBDACCDAA