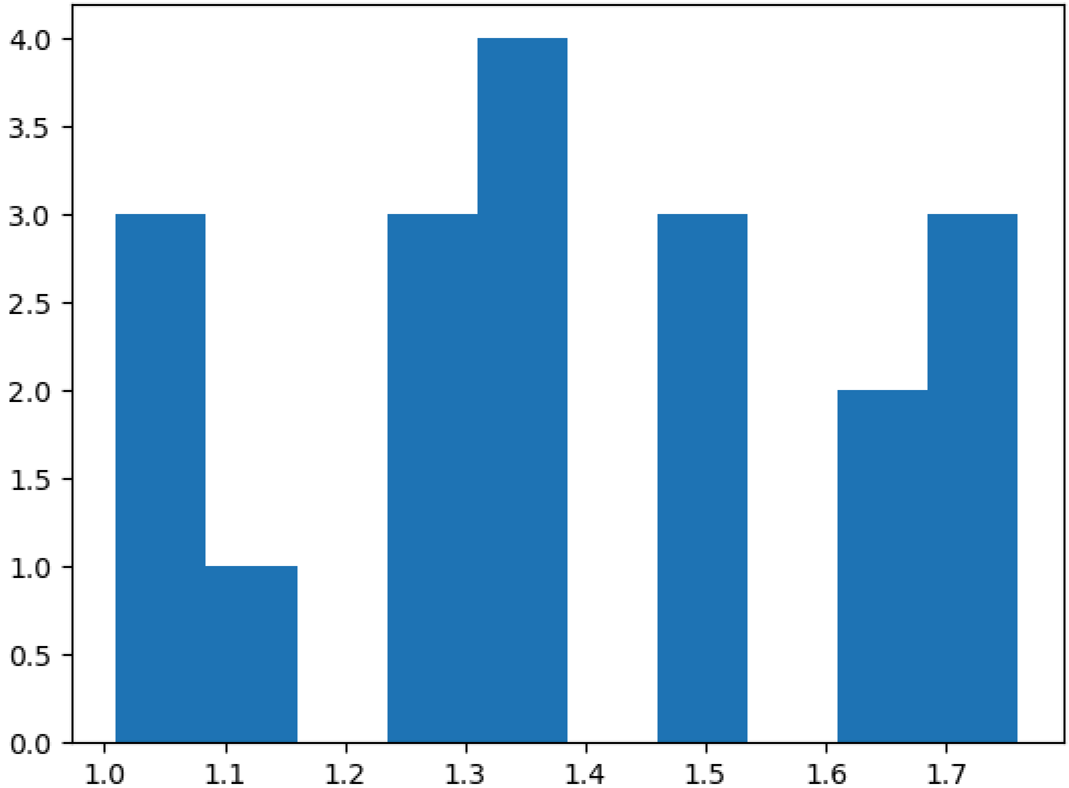
In [14]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [15]:
```
trial = np.arange(1, 21, 1)
times = np.array([1.36, 1.76, 1.10, 1.76, 1.46, 1.48, 1.73, 1.48, 1.33, 1.01, 1.28,
```

In [16]:
```
plt.hist(times)
plt.show()
```



In [17]:
```
s = np.std(times, ddof=1)
u = s / np.sqrt(len(times))
print(f"Standard Deviation: {s:.3f}")
print(f"Uncertainty: {u:.3f}")
```

```
Standard Deviation: 0.234
Uncertainty: 0.054
```

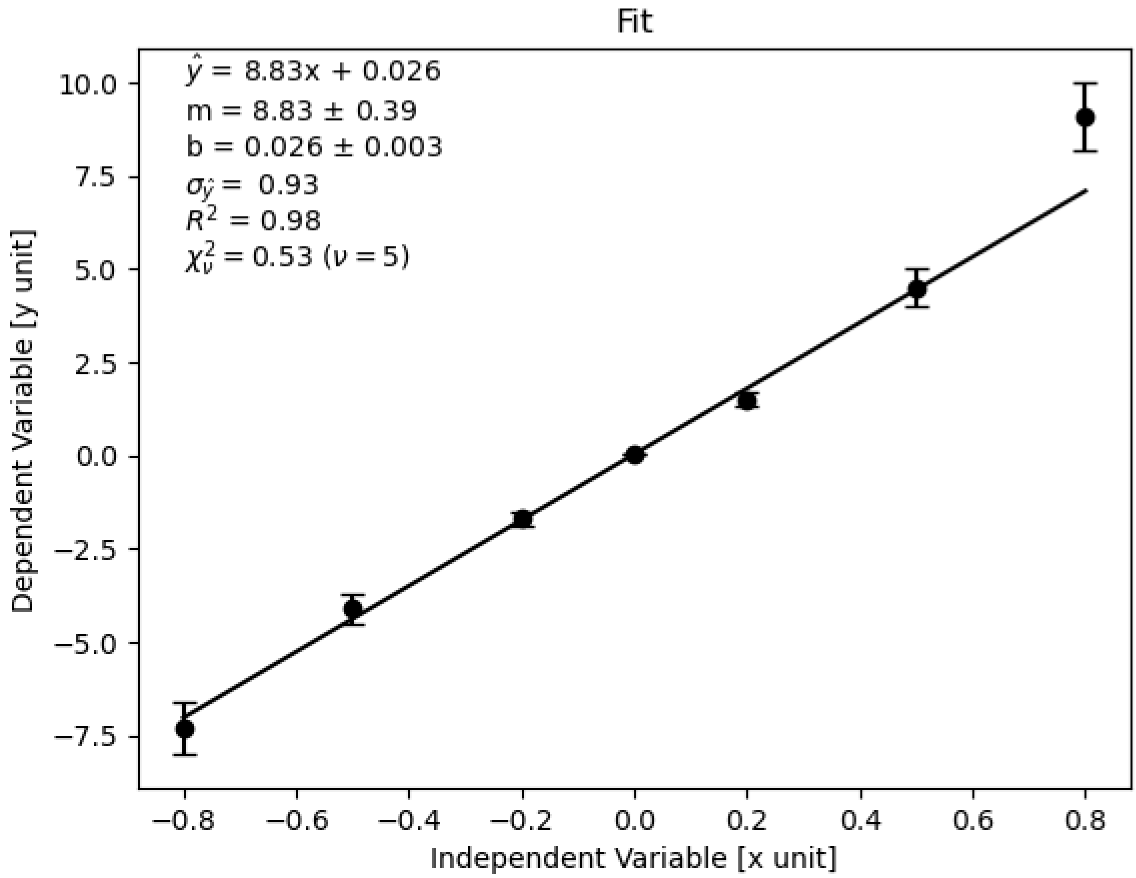
Seventh Computing Task

In [18]:
```
import scipy.optimize as opt
```

In [28]:
```
x = np.array([-0.8, -0.5, -0.2, 0.0, 0.2, 0.5, 0.8])      # Independent variable
y = np.array([-7.3, -4.1, -1.7, 0.026, 1.5, 4.5, 9.1])    # Dependent variable
sigmay = np.array([0.7, 0.4, 0.2, 0.003, 0.2, 0.5, 0.9]) # Uncertainties on y

# Parametize model (function) that should fit data; here linear:   y = mx + b
def f(x, m, b):
        return m*x + b
```

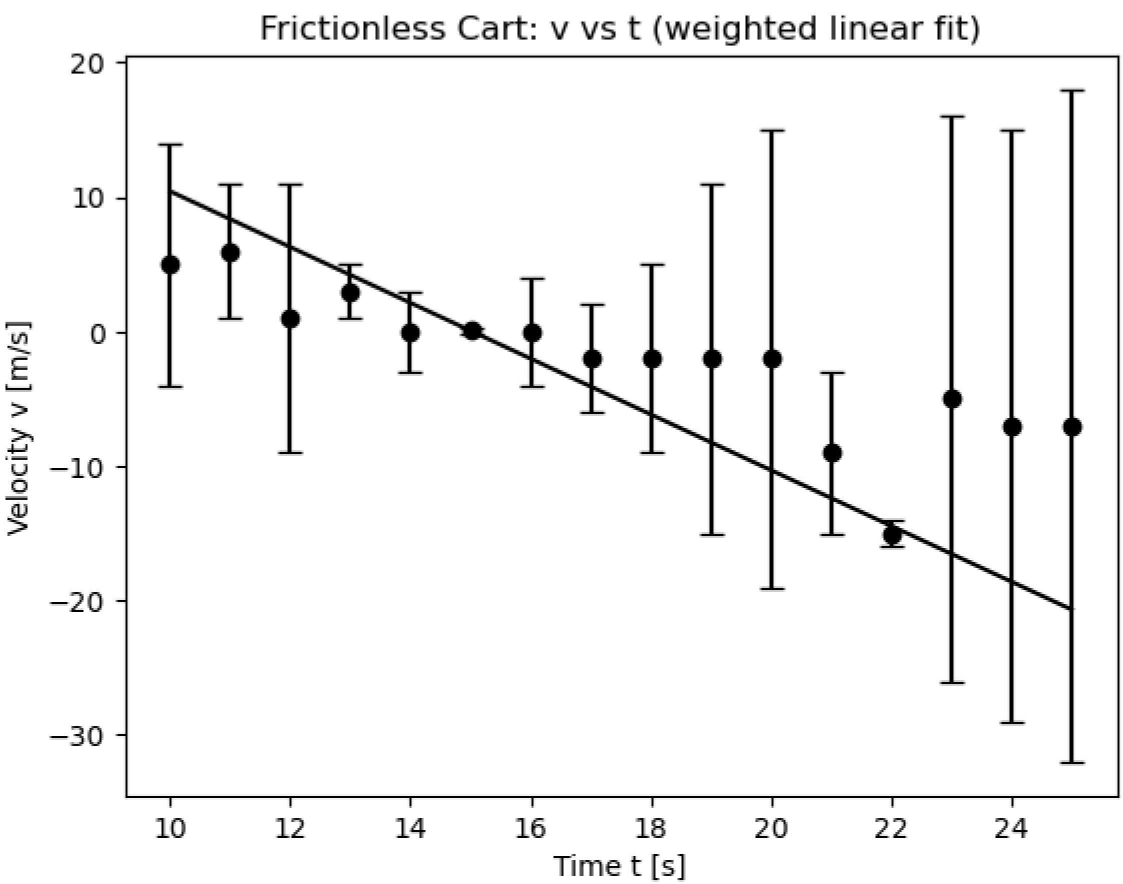
```
# Get the fit parameters (here, m and b) and covariance matrix from curve_fit
# make initial guess for fit parameters and assign to p0
# Weight the dependent variable by sigmay, the array of standard uncertainties
p0 = 8, 0
(params, covmat) = opt.curve_fit(f, x, y, p0, sigma=sigmay, absolute_sigma=True)

# Model's prediction for dependent variable
# The * before params means that this is a list with an arbitrary number of
# elements
fitequation = f(x, *params)

# First plot the data as black dots ('ko')
# include dependent variable uncertainties as error bars with caps
plt.errorbar(x, y, yerr=sigmay, fmt='ko', capsize=4)

# Overlay model, the fit result, as a straight black line ('k-')
plt.plot(x, fitequation, 'k-')

plt.xlabel('Independent Variable [x unit]')
plt.ylabel('Dependent Variable [y unit]')
plt.title('Fit')

eq = r"$\hat{y}$ = " + str(round(params[0],2)) + "x + " + str(round(params[1],4))
plt.text(-0.8, 10, eq)
sl = "m = " + str(round(params[0],2)) + r" $\pm$ " + \
     str(round(np.sqrt(covmat[0,0]),2))
plt.text(-0.8, 9, sl)
interc = "b = " + str(round(params[1],4)) + r" $\pm$ " + \
     str(round(np.sqrt(covmat[1,1]),4))
plt.text(-0.8, 8, interc)


# Print the fit function equation, the uncertainties of the parameters,
print("y = {0:3.2f} x + {1:6.4f}" . format(params[0], params[1]))
print('m = {0:3.2f} +/- {1:3.2f}' . format(params[0], np.sqrt(covmat[0,0])))
print('b = {0:6.4f} +/- {1:6.4f}' . format(params[1], np.sqrt(covmat[1,1])))
# Calculate and print out the uncertainty of fit
ufit = np.sqrt(sum((y - f(x, *params))**2)/(len(y) - 2))
print('Fit uncertainty = {0:4.3f}'.format(ufit))
plt.text(-0.8, 7, r"$\sigma_{\hat{y}} =$ " + str(round(ufit,2)))
# Calculate and print out the coefficient of determination
r2 = 1. -  sum((y - f(x, *params))**2)/sum((y - y.mean())**2)
print("R^2 = {0:4.3f}". format(r2))
plt.text(-0.8, 6, r"$R^2$ = " + str(round(r2, 2)))
# Calculate and print out the reduced chi-squared.
rchi2 = sum(((y - f(x, *params))**2)/(ufit**2 + sigmay**2))/(len(x) - \
        len(params))
print("Reduced chi-squared = {0:4.3f}".format(rchi2))
plt.text(-0.8, 5, r"$\chi^2_{\nu} = $" + str(round(rchi2, 2)) + \
        r" $(\nu = $" + str(len(x) - len(params)) + r"$)$")
plt.show()
```

```
y = 8.83 x + 0.0260
m = 8.83 +/- 0.39
b = 0.0260 +/- 0.0030
Fit uncertainty = 0.927
R^2 = 0.976
Reduced chi-squared = 0.532
```



In [33]:
```
# Data from the handout (t, v, sigma_v)
tt = np.arange(10, 26, 1)
vv = np.array([ 5,  6,  1,  3,  0, 0.1,  0, -2, -2, -2, -2, -9, -15, -5, -7, -7])
sigmav = np.array([ 9,  5, 10,  2,  3, 0.2,  4,  4,  7, 13, 17,  6,   1, 21, 22, 25

# Linear model: v(t) = a*t + b  (a is acceleration, b is intercept)
def f(tt, a, b):
    return a*tt + b

# Initial guess
p0 = (-1, 10)

# Weighted fit (absolute_sigma=True means sigmav are true 1-sigma uncertainties)
(params, covmat) = opt.curve_fit(f, tt, vv, p0=p0, sigma=sigmav, absolute_sigma=Tru
a_fit, b_fit = params
sa, sb = np.sqrt(np.diag(covmat))

# Fit predictions
vhat = f(tt, *params)

# Fit uncertainty (as in your example)
ufit = np.sqrt(np.sum((vv - vhat)**2) / (len(v) - len(params)))
```

```
# R^2
r2 = 1.0 - np.sum((vv - vhat)**2) / np.sum((vv - np.mean(v))**2)

# Reduced chi-squared (same structure as your example)
nu = len(tt) - len(params)
rchi2 = np.sum(((vv - vhat)**2) / (ufit**2 + sigmav**2)) / nu

# ---- Plot ----
plt.errorbar(tt, vv, yerr=sigmav, fmt='ko', capsize=4)
plt.plot(tt, vhat, 'k-')
plt.xlabel('Time t [s]')
plt.ylabel('Velocity v [m/s]')
plt.title('Frictionless Cart: v vs t (weighted linear fit)')


plt.show()

# ---- Print required outputs ----
print(f"Fit function: v(t) = a t + b")
print(f"a = {a_fit:.6f} +/- {sa:.6f}  (m/s^2)")
print(f"b = {b_fit:.6f} +/- {sb:.6f}  (m/s)")
print(f"R^2 = {r2:.6f}")
print(f"Reduced chi-squared = {rchi2:.6f}")
print(f"Degrees of freedom nu = {nu}")
print(f"Fit uncertainty ufit = {ufit:.6f}")
```



```
Fit function: v(t) = a t + b
a = -2.074207 +/- 0.139158  (m/s^2)
b = 31.183600 +/- 2.130959  (m/s)
R^2 = -0.535033
Reduced chi-squared = 0.160250
Degrees of freedom nu = 14
Fit uncertainty ufit = 6.909850
```

1. Is the assumption of constant acceleration reasonable?

The reduced chi-squared value is $\chi_\nu^2 = 0.16$, which is less than 1. This indicates the linear model is consistent with the data within the measurement uncertainties, so the assumption of constant acceleration is reasonable.

2. What was the acceleration?

From the weighted linear fit $v(t) = at + b$, the slope gives the acceleration: $a = -2.07 \pm 0.14\ \mathrm{m/s^2}$. Therefore, the cart's acceleration while moving up the incline was $-2.07\ \mathrm{m/s^2}$.

3. What was the cart's velocity before it started up the incline?

Using the fit equation, the velocity at $t = 10s$ (start of the incline) is $v(10) = a(10) + b = -2.07(10) + 31.18 \approx 10.5\ \mathrm{m/s}$. Thus, the cart's velocity at the start of the incline was approximately $10.5 m/s$.

4. Is the measured acceleration consistent with expectation?

For a frictionless incline at 25°, the expected acceleration is $a = -g\sin(25^\circ) \approx -4.15\ \mathrm{m/s^2}$. Since the measured value $-2.07 \pm 0.14\ \mathrm{m/s^2}$ differs significantly from this, the result is not consistent with a purely frictionless incline.