Vehicle Vibrations on a Bumpy Road

Constructing and running the `bumpy_road` example

Manuel A. Diaz

InspiraSTEM Internship

July 21, 2026

Outline

This worked example

- generates (or loads) road-height profiles as CSV data,
- applies those profiles as forcing for a mass–spring–damper ODE,
- solves the ODE with a centered finite-difference method,
- visualizes displacements, spectra, and an animation of the ride.

Skills illustrated

- NumPy arrays, finite differences, and ODE time stepping
- reading/writing CSV and pickling results
- command-line options with `argparse`
- Matplotlib plotting and FFT spectra
- SymPy manufactured solutions and `pytest`
- packaging the workflow with `uv run`

Sources live in `bumpy_road/` at https://github.com/wme7/kinematics.

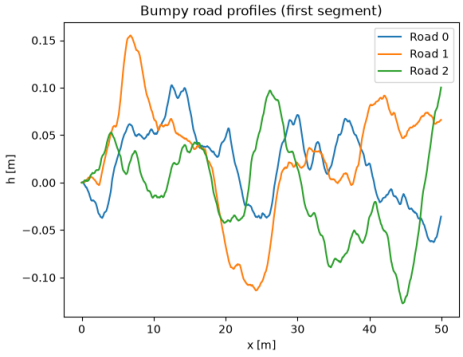
Suggested student workflow

```
cd bumpy_road
uv run python generate_road_profiles.py # -> bumpy.csv
uv run python plot_roads.py # -> road_profiles.png
uv run python bumpy.py # -> bumpy.res
uv run python explore.py 180 # plots / spectra
uv run python bumpy_road_anim.py # GIF / MP4
```

Outline

Physical setup

- A vehicle (mass m) rides at constant speed v over a bumpy road.
- Road height $h(x)$ varies with distance x along the road.
- The suspension is modeled as a mass–spring–damper system.
- We compute the vertical displacement $u(t)$ of the vehicle.



Mathematical model

General vibration ODE:

$$mu'' + f(u') + s(u) = F(t), \qquad u(0) = I, \quad u'(0) = V.$$

- Input: mass m, damping force $f(u')$, spring force $s(u)$, external force $F(t)$, initial data I, V
- Output: vertical displacement $u(t)$

Specialization used in the app

In `bumpy.py` we usually take **linear** spring and damper:

$$mu'' + bu' + ku = F(t).$$

- k: spring stiffness
- b: damping coefficient
- Typical bicycle defaults: $m = 60\,\text{kg}$, $k = 60\,\text{N/m}$, $b = 80\,\text{Ns/m}$, $v = 5\,\text{m/s}$

The solver also supports *quadratic* damping $f(u') = b|u'|u'$ (see next section).

From road height to forcing

Constant speed: $x = v\,t$. Vertical acceleration of the road contact:

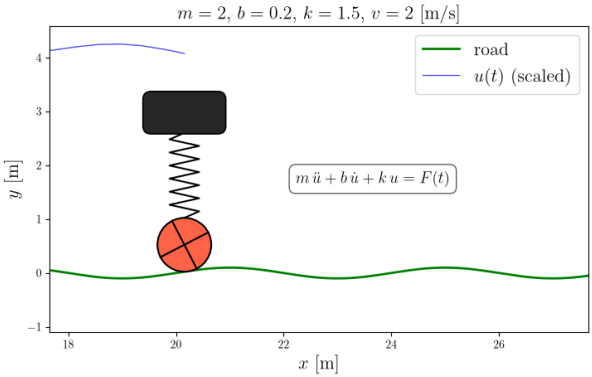
$$a(t) = \frac{d^2}{dt^2} h(x) = v^2\, h''(x).$$

Force on the mass:

$$F(t) = -m\, a(t) = -m\, v^2\, h''(x).$$

So the road curvature (and speed) set the excitation of the suspension.

Animation of the ride



Snapshot from `bumpy_road_anim.py` (writes `bumpy_road.gif` / `.mp4` in `bumpy_road/`).

Outline

Finite-difference idea

- Centered differences in time
- u^n: approximation to $u(t_n)$, $t_n = n\Delta t$
- First: linear damping $f(u') = bu'$

Update formula (linear damping)

$$u^{n+1} = \frac{2mu^n + (\frac{b}{2}\Delta t - m)u^{n-1} + \Delta t^2(F^n - s(u^n))}{m + \frac{b}{2}\Delta t}$$

Special starter step for $n = 0$:

$$u^1 = u^0 + \Delta t\, V + \frac{\Delta t^2}{2m}(-bV - s(u^0) + F^0).$$

Quadratic damping

For $f(u') = b|u'|u'$, linearize with a geometric mean:

$$|u'|u'\Big|^n \approx |u'|^{n-\frac{1}{2}}\,(u')^{n+\frac{1}{2}}.$$

Resulting update (sketch):

$$u^{n+1} = \frac{2mu^n - mu^{n-1} + bu^n|u^n - u^{n-1}| + \Delta t^2\big(F^n - s(u^n)\big)}{m + b\,|u^n - u^{n-1}|}.$$

Implemented in `solver.py` via `damping='quadratic'`.

Simple linear solver

Teaching version: `solver_linear_damping`

```
from numpy import *

def solver_linear_damping(I, V, m, b, s, F, t):
    N = t.size - 1 # No of time intervals
    dt = t[1] - t[0] # Time step
    u = zeros(N+1) # Result array
    u[0] = I
    u[1] = u[0] + dt*V + dt**2/(2*m)*(-b*V - s(u[0]) + F[0])

    for n in range(1, N):
        u[n+1] = 1./(m + b*dt/2)*(2*m*u[n] +
                 (b*dt/2 - m)*u[n-1] + dt**2*(F[n] - s(u[n])))
    return u
```

Using the linear solver

```
from solver import solver_linear_damping
from numpy import *

def s(u):
    return 2*u

T = 10*pi
dt = 0.2
N = int(round(T/dt))
t = linspace(0, T, N+1)
F = zeros(t.size)
I = 1; V = 0
m = 2; b = 0.2
u = solver_linear_damping(I, V, m, b, s, F, t)
```

Free vibration: $F \equiv 0$, linear spring $s(u) = 2u$.

Full solver (API)

Improvements over the teaching version:

- linear *and* quadratic damping
- F may be a function or a measured array
- docstrings; raise exceptions on bad input

Full solver (setup and u^1)

```
import numpy as np

def solver(I, V, m, b, s, F, t, damping='linear'):
    """Solve m*u'' + f(u') + s(u) = F on the mesh t."""
    N = t.size - 1
    dt = t[1] - t[0]
    u = np.zeros(N+1)
    if callable(F):
        F = F(t)
    else:
        F = np.asarray(F)

    u[0] = I
    if damping == 'linear':
        u[1] = u[0] + dt*V + \
               dt**2/(2*m)*(-b*V - s(u[0]) + F[0])
    elif damping == 'quadratic':
        u[1] = u[0] + dt*V + \
               dt**2/(2*m)*(-b*V*abs(V) - s(u[0]) + F[0])
```

Full solver (time stepping)

```
for n in range(1, N):
    if damping == 'linear':
        u[n+1] = (2*m*u[n] + (b*dt/2 - m)*u[n-1] +
                  dt**2*(F[n] - s(u[n])))/(m + b*dt/2)
    elif damping == 'quadratic':
        u[n+1] = (2*m*u[n] - m*u[n-1] +
                  b*u[n]*abs(u[n] - u[n-1]) -
                  dt**2*(s(u[n]) - F[n])) / \
                 (m + b*abs(u[n] - u[n-1]))
return u, t
```

Nonlinear spring + quadratic damping

```
import numpy as np
from numpy import sin, pi
from solver import solver

def F(t):
    return A*sin(pi*t) # sinusoidal forcing

def s(u):
    return k*(0.2*u + 1.5*u**3) # nonlinear spring

A = 0.25
k = 2
t = np.linspace(0, 100, 10001)
u, t = solver(I=0.1, V=0, m=2, b=0.5, s=s, F=F, t=t,
              damping='quadratic')
```

Outline

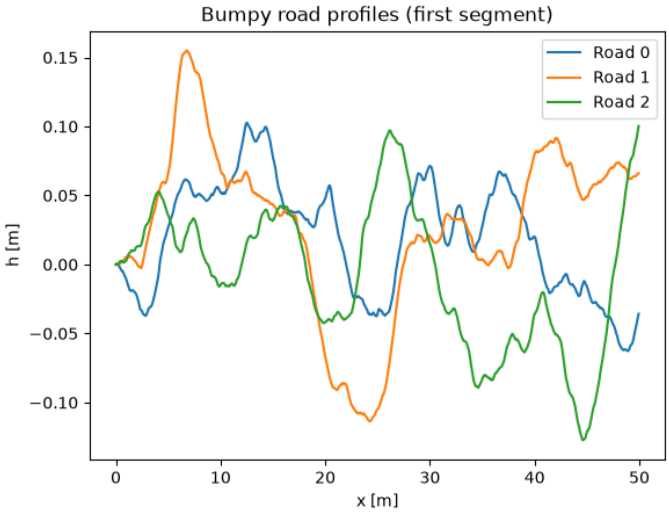
Road data as CSV

- `draw_bumpy.csv`: short sample, columns `x,h`
- `bumpy.csv`: generated file, columns `x,h0,h1,...` (several realizations)

Generate the full file:

```
uv run python generate_road_profiles.py
uv run python plot_roads.py
```

Road profiles



Loading the CSV

```python
import numpy as np

def load_road_csv(filename):
    """Load CSV with columns x, h0, h1, ..."""
    data = np.loadtxt(filename, delimiter=',', skiprows=1)
    if data.ndim != 2 or data.shape[1] < 2:
        raise ValueError(
            f'Expected CSV with columns x,h0,... ; got {data.shape}')
    x = data[:, 0]
    h_data = data[:, 1:].T # shape (nroads, npoints)
    return x, h_data
```

- First column → distance x
- Remaining columns → height profiles (rows of `h_data`)

2D array layout (quick reminder)

```
>>> h_data.shape # (nroads, npoints)
(3, 10001)
>>> h = h_data[0, :] # first road profile
>>> h_data[1, 10:15] # slice of second profile
```

Acceleration from curvature

$$F(t) \sim -m\, v^2\, h''(x)$$

Approximate $h''(x)$ by centered finite differences on the road mesh.

Scalar and vectorized acceleration

```python
import numpy as np

def acceleration(h, x, v):
    """Compute a = v**2 * h''(x) by centered differences."""
    d2h = np.zeros(h.size)
    dx = x[1] - x[0]
    for i in range(1, h.size - 1):
        d2h[i] = (h[i-1] - 2*h[i] + h[i+1]) / dx**2
    d2h[0] = d2h[1]
    d2h[-1] = d2h[-2]
    return d2h * v**2

def acceleration_vectorized(h, x, v):
    d2h = np.zeros(h.size)
    dx = x[1] - x[0]
    d2h[1:-1] = (h[:-2] - 2*h[1:-1] + h[2:]) / dx**2
    d2h[0] = d2h[1]
    d2h[-1] = d2h[-2]
    return d2h * v**2
```

Looping over road realizations

```
t = x / v # time corresponding to x
data = [x, t]
for i in range(h_data.shape[0]):
    h = h_data[i, :]
    a = acceleration(h, x, v)
    F = -m * a
    u, t = solver(I=0, V=0, m=m, b=b, s=s, F=F, t=t,
                  damping='linear')
    data.append([h, F, u])
```

Resulting list: `[x, t, [h,F,u], [h,F,u], ...]`.

Outline

High-level function `bumpy_road`

1. Resolve `roadfile`: local CSV path or URL (`urllib.request` download if needed)
2. `load_road_csv` $\rightarrow x$, h-profiles
3. Set $t = x/v$; optional stability check on Δt
4. For each profile: $a = h''v^2$, $F = -ma$, call `solver`
5. Return `data = [x, t, [h,F,u], ...]`

Noise level: RMS displacement

$$u_{\text{rms}} = \sqrt{T^{-1}\int_0^T u^2\,dt} \approx \sqrt{\frac{1}{N+1}\sum_{n=0}^{N}(u^n)^2}.$$

Compact list comprehension in `bumpy.py`:

```
u_rms = [np.sqrt((1.0/len(u))*np.sum(u**2))
         for h, F, u in data[2:]]
```

Pickling results

Persist the whole `data` list for later plotting:

```
import pickle
import numpy as np

# After bumpy_road(...):
u_rms = [np.sqrt((1.0/len(u))*np.sum(u**2))
         for h, F, u in data[2:]]
print('u_rms:', u_rms)

with open('bumpy.res', 'wb') as outfile:
    pickle.dump(data, outfile)
```

Use binary modes `'wb'`/`'rb'` (Python 3).

Command-line options

```
import argparse

def command_line_options():
    parser = argparse.ArgumentParser(
        description='Bumpy-road vehicle vibration demo')
    parser.add_argument('--m', '--mass', type=float, default=60)
    parser.add_argument('--k', '--spring', type=float, default=60)
    parser.add_argument('--b', '--damping', type=float, default=80)
    parser.add_argument('--v', '--velocity', type=float, default=5)
    parser.add_argument('--roadfile', type=str, default='bumpy.csv',
                        help='local CSV (or URL) with road data')
    args = parser.parse_args()
    return args.roadfile, args.m, args.b, args.k, args.v
```

Running a simulation

```
uv run python bumpy.py --help
uv run python bumpy.py --v 8 --b 100
uv run python bumpy.py --roadfile bumpy.csv
```

Prints `u_rms` values and writes `bumpy.res`.

Outline

Goals of `explore.py`

After a successful `bumpy.py` run, plot

- $u(t)$ and $u'(t)$ for $t \geq t_s$
- FFT spectra of u and F (dominant frequencies)
- stacked panels of $h(x)$, $F(t)$, and $u(t)$ for each road realization

```
uv run python explore.py 180
```

Loading pickled data

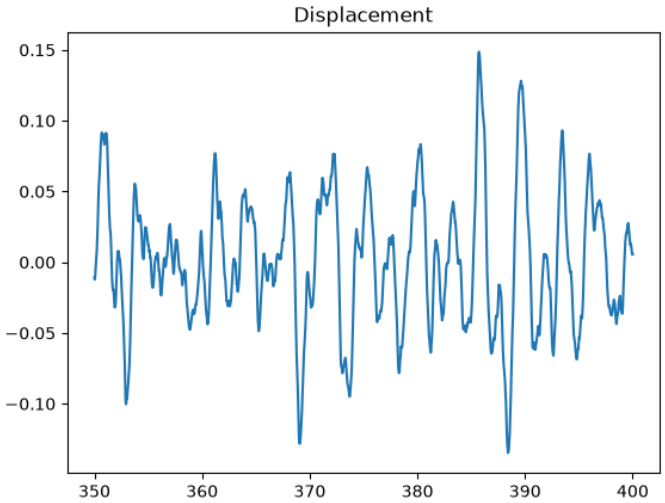
```
import pickle
from numpy import *
from matplotlib.pyplot import *

with open('bumpy.res', 'rb') as outfile:
    data = pickle.load(outfile)

x, t = data[0:2]
t_s = 180
indices = t >= t_s
t = t[indices]
x = x[indices]
```

Boolean mask `t >= t_s` selects the late-time window.

Late-time displacement



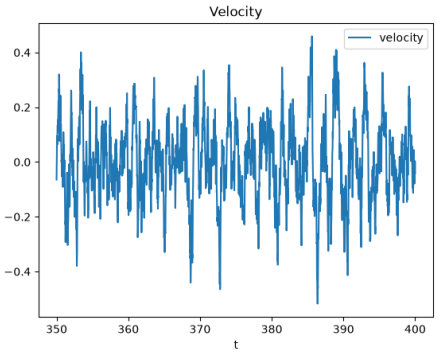
Velocity by centered differences

$$v^n = (u')^n \approx \frac{u^{n+1} - u^{n-1}}{2\Delta t} \quad (1 \leq n \leq N-1),$$

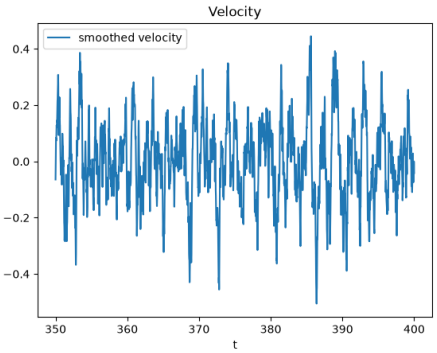
with one-sided formulas at the ends. Vectorized NumPy slicing is much faster than a Python loop.

Velocity plots

Raw



Smoothed

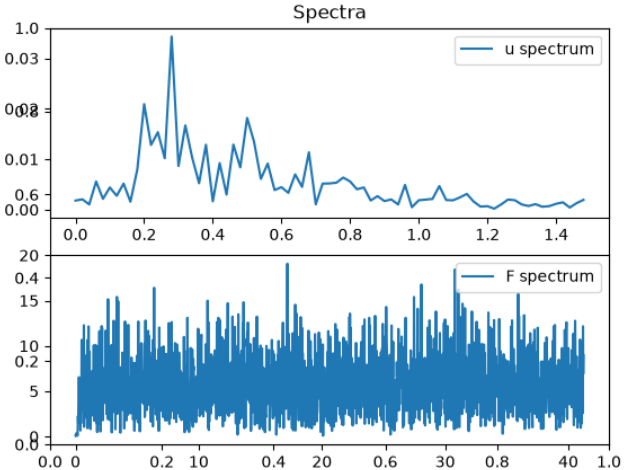


Frequency analysis (FFT)

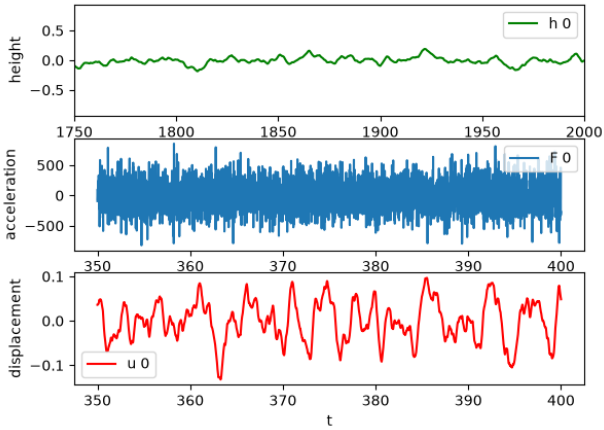
```
from numpy import *

def frequency_analysis(u, t):
    A = fft.fft(u)
    A = 2*A
    dt = t[1] - t[0]
    N = t.size
    freq = arange(N/2, dtype=float)/N/dt
    A = abs(A[0:freq.size])/N
    tol = 0.05*A.max()
    for i in range(len(A) - 1, 0, -1):
        if A[i] > tol:
            break
    return freq[:i+1], A[:i+1]
```

Spectra



Road, force, and displacement



Animation options

```
uv run python bumpy_road_anim.py --help
uv run python bumpy_road_anim.py --frames 60 --fps 10
uv run python bumpy_road_anim.py --roadfile draw_bumpy.csv
uv run python bumpy_road_anim.py --roadfile bumpy.csv --profile 0
```

Default road is a built-in sine profile; CSV roads are interpolated.

Outline

Why verify?

- Finite-difference bugs are easy to introduce
- A *manufactured solution*: pick a simple exact $u(t)$, compute the forcing $F(t)$ that makes the ODE hold, then check that the numerical solver recovers u
- SymPy builds F; `pytest` asserts the error is tiny

SymPy sketch of the ODE

```
import sympy as sp

t, m, b, k = sp.symbols('t m b k', real=True, positive=True)
u = sp.Function('u')

ode = m*sp.diff(u(t), t, 2) + b*sp.diff(u(t), t) + k*u(t)
print(ode)

# Manufactured forcing for a chosen exact solution
I, V, q = 1.2, 3.0, 2.0
u_exact = I + V*t + q*t**2
F = sp.lambdify(t, ode.subs(u(t), u_exact).doit(), 'numpy')
```

(Modernized Python 3 style; see also `symbolic.py` and `tests/test_bumpy.py`.)

Manufactured solution idea

Choose e.g. $u(t) = I + Vt + qt^2$ (linear damping).

1. Form LHS $= mu'' + bu' + s(u)$ symbolically
2. Set $F(t) =$ LHS evaluated on that u
3. Call `solver(..., F, ...)`
4. Compare numerical u^n to the exact $u(t_n)$

For quadratic damping, a linear exact solution $u = I + Vt$ is convenient.

Test function (excerpt)

```
import numpy as np
import sympy as sp
from solver import solver

def lhs_eq(t, m, b, s, u, damping='linear'):
    v = sp.diff(u, t)
    d = b*v if damping == 'linear' else b*v*sp.Abs(v)
    return m*sp.diff(u, t, t) + d + s(u)

def test_solver():
    I, V, m, b, k = 1.2, 3.0, 2.0, 0.9, 4.0
    s = lambda u: k*u
    t = sp.Symbol('t')
    time = np.linspace(0, 2.0, 11)
    u_exact = I + V*t + 2*t**2
    F = sp.lambdify(t, lhs_eq(t, m, b, s, u_exact), 'numpy')
    u, t_ = solver(I, V, m, b, s, F, time, 'linear')
    assert abs(sp.lambdify(t, u_exact, 'numpy')(t_) - u).max() < 1e-13
```

Running the tests

```
# from the repository root (pyproject.toml)
uv run pytest bumpy_road/tests/test_bumpy.py -q
```

Also covers scalar vs vectorized `acceleration`.

Optional extras

- `-cython` flag in `bumpy.py`: optional compiled solver (not required for the first run)
- `timings.py`: longer runs / performance checks
- `./clean.sh`: remove generated `bumpy.csv`, `bumpy.res`, plots

Takeaways

- Road curvature + speed → forcing of a mass–spring–damper ODE
- Centered FD time stepping in `solver.py`
- CSV roads, `uv` workflow, pickle + explore + FFT
- Manufactured solutions + `pytest` for confidence

Folder: `bumpy_road/` in the kinematics repository.