## Supplementary Material

# Harmonic Identifiability and Singular Conditioning in Quadratic Jerk Systems with PID/PIDD Control

Robert Vrabel
Slovak University of Technology in Bratislava, Slovakia
robert.vrabel@stuba.sk

## S1. Computational settings

The Python script in Section S3 generates both figures and the associated numerical data. The plant coefficients are $(a_0, a_1, a_2) = (1, 0.6, 2)$. In the gain order $(K_p, K_i, K_d, K_{dd})$, the controllers are $K_0 = (3, 1, 5.4, 2)^\top$ and $K_1 = (3.5, 1.5, 5.9, 2.5)^\top$.

| Setting | Value |
|---|---|
| Frequency sweep | $K_0$; $Q = x_1^2 + x_2^2$ and $Q = x_1^2 + 2x_2^2 + x_3^2$; $\varepsilon = 2^{-j}$, $j = 3, \ldots, 6$; $\omega = 1 \pm 2^{-k}$, $k = 3, \ldots, 8$. Exact degeneracy: $\omega = 1$, $\varepsilon = 2^{-3}$. |
| Energy experiment | $K_0, K_1$; $Q = x_1^2$; $\omega = 1$; $\varepsilon = 2^{-j}$, $j = 3, \ldots, 10$. |
| Fourier modes | $-N, \ldots, N$, $N = 32, 64$; zero output mean; linear convolution with truncation. |
| Iteration | Zero initial correction and sensitivities; stopping threshold $3 \times 10^{-17}$ (extended) or $3 \times 10^{-14}$ (double); 300-iteration cap. |
| Precision | NumPy longdouble/clongdouble for Fourier iteration where supported; float64 for singular value decomposition. |

**Execution.** Save the Python source as `verify_harmonics.py` in a writable directory with Python 3, NumPy and Matplotlib installed. Run:

```
python -m pip install numpy matplotlib
python verify_harmonics.py
```

Output: `singular_scaling.eps`, `waveform_energy.eps`, `singular_scaling.csv` (96 rows) and `waveform_energy.csv` (8 rows). The console reports residuals, truncation and precision comparisons, and the fitted slope. The accompanying archive includes the executable source and complete CSV files. Reference execution: Python 3.13.5, NumPy 2.3.5, Matplotlib 3.10.8; the platform provides a 63-bit stored mantissa for `longdouble`.

## S2. Reference results

| Diagnostic | Reference value |
|---|---|
| Maximum scaled fixed-point residual | $2.115 \times 10^{-19}$ |
| Maximum number of iterations | 9 |
| Maximum relative precision difference in singular values | $2.777 \times 10^{-8}$ |
| Relative difference, $N = 32$ and $N = 64$ (singular values) | 0 at reported precision |
| Relative difference, $N = 32$ and $N = 64$ (energy) | 0 at reported precision |
| Fitted energy slope | 1.999989784 |
| Analytical limit of $E(\varepsilon)/\varepsilon^2$ | $1.8680555556 \times 10^{-4}$ |

**Table S1.** Smallest singular value at $\varepsilon = 2^{-6}$ and $|\omega - 1| = 2^{-8}$.

| Quadratic form | $\omega = 1 - 2^{-8}$ | $\omega = 1 + 2^{-8}$ |
|---|---|---|
| $x_1^2 + x_2^2$ | $2.07017592 \times 10^{-10}$ | $1.97692019 \times 10^{-10}$ |
| $x_1^2 + 2x_2^2 + x_3^2$ | $1.61416914 \times 10^{-12}$ | $1.54748836 \times 10^{-12}$ |

Normalized singular-value limits at $(\varepsilon, \omega) \to (0, 1)$: $\sqrt{2}/32$ (twice), $3/(10000\sqrt{2})$, and $3/(20000\sqrt{2})$.

**Table S2.** Relative energy of the omitted harmonics, $E = \sum_{n=3}^{N} |\Delta Y_n|^2 / \sum_{n=1}^{2} |\Delta Y_n|^2$.

| $\varepsilon$ | $E$ $(N = 32)$ | $E$ $(N = 64)$ | $E/\varepsilon^2$ |
|---|---|---|---|
| $2^{-3}$ | $2.91863122 \times 10^{-6}$ | $2.91863122 \times 10^{-6}$ | $1.86792398 \times 10^{-4}$ |
| $2^{-4}$ | $7.29696352 \times 10^{-7}$ | $7.29696352 \times 10^{-7}$ | $1.86802266 \times 10^{-4}$ |
| $2^{-5}$ | $1.82426497 \times 10^{-7}$ | $1.82426497 \times 10^{-7}$ | $1.86804733 \times 10^{-4}$ |
| $2^{-6}$ | $4.56067749 \times 10^{-8}$ | $4.56067749 \times 10^{-8}$ | $1.86805350 \times 10^{-4}$ |
| $2^{-7}$ | $1.14017031 \times 10^{-8}$ | $1.14017031 \times 10^{-8}$ | $1.86805504 \times 10^{-4}$ |
| $2^{-8}$ | $2.85042637 \times 10^{-9}$ | $2.85042637 \times 10^{-9}$ | $1.86805543 \times 10^{-4}$ |
| $2^{-9}$ | $7.12606630 \times 10^{-10}$ | $7.12606630 \times 10^{-10}$ | $1.86805552 \times 10^{-4}$ |
| $2^{-10}$ | $1.78151660 \times 10^{-10}$ | $1.78151660 \times 10^{-10}$ | $1.86805555 \times 10^{-4}$ |

**CSV fields.** In `singular_scaling.csv`, `case` is 1 for the simple root and 2 for the double root; `offset` is $|\omega - 1|$; `gamma` is $\Gamma_Q(\omega)$; `sigma1`–`sigma4` are singular values in decreasing order; `norm1`–`norm4` are their normalized values. The final columns give the scaled residual and iteration count. In `waveform_energy.csv`, the energy columns give $E$ at the indicated truncations, and the final column gives $E/\varepsilon^2$ at $N = 32$.

# S3. Python source code

verify_harmonics.py

```
"""Scaled Fourier-Galerkin solver and differentiated fixed-point equations.
Run with numpy and matplotlib. Generates EPS and CSV directly.
"""
from pathlib import Path
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
OUT = Path(__file__).resolve().parent
K0 = [3, 1, 5.4, 2]
K1 = [3.5, 1.5, 5.9, 2.5]

def solve(K, e, w, q, N=32, high=True):
    R = np.longdouble if high else np.float64
    C = np.clongdouble if high else np.complex128
    K = np.array(K, dtype=R)
    e = R(e)
    w = R(w)
    q = np.array(q, dtype=R)
    s = np.array(1j * w * np.arange(-N, N + 1), dtype=C)
    ss = s.copy()
    ss[N] = 1
    H = 1 / (ss ** 3 + (2 + K[3]) * ss ** 2 + (R('0.6') + K[2]) * ss + 1 + K[0] + K[1] / ss)
    H[N] = 0
    dH = -H ** 2 * np.array([np.ones_like(s), 1 / ss, s, s * s])
    g = q[0] - q[1] * w * w + q[2] * w ** 4
    if tuple(q) == (1, 2, 1):
        g = (1 - w * w) ** 2

    def bil(a, b):
        return sum((q[j] * np.convolve(s ** j * a, s ** j * b)[N:3 * N + 1] for j in range(3) if
    q[j] != 0))
    f = np.zeros(2 * N + 1, dtype=C)
    f[N - 1] = f[N + 1] = R('.5')
    l = H * f
    dl = dH * f
    base = np.zeros_like(l)
    db = np.zeros_like(dl)
    for k in (N - 2, N + 2):
        j = N + (k - N) // 2
        base[k] = l[j] ** 2
        db[:, k] = 2 * l[j] * dl[:, j]
    v = np.zeros_like(l)
    dv = np.zeros_like(dl)
    b = e * e * g
    for it in range(300):
        rhs = base + 2 * e * bil(l, v) + b * bil(v, v)
        vn = H * rhs
        dn = np.array([
            dH[j] * rhs + H * (db[j] + 2 * e * (bil(dl[j], v)
                + bil(l, dv[j])) + 2 * b * bil(v, dv[j]))
            for j in range(4)])
        err = max(np.max(abs(vn - v)), np.max(abs(dn - dv)))
        v, dv = (vn, dn)
        if err < (3e-17 if high else 3e-14):
            break
    else:
        raise RuntimeError('iteration failed')
    res = float(np.max(abs(v - H * (base + 2 * e * bil(l, v) + b * bil(v, v)))))
    y = e * l + b * v
```

```
    dy = e * dl + b * dv
    J = np.array([dy[:, N + 1].real, dy[:, N + 1].imag,
                  dy[:, N + 2].real, dy[:, N + 2].imag], dtype=float)
    return (y, np.linalg.svd(J, compute_uv=False), float(g), res, it + 1)

def main():
    plt.rcParams.update({'font.size': 11, 'legend.fontsize': 8, 'ps.fonttype': 42})
    amps = 2.0 ** (-np.arange(3, 7))
    off = 2.0 ** (-np.arange(3, 9))
    ref = np.array([np.sqrt(2) / 32] * 2
                   + [3 / (10000 * np.sqrt(2)), 3 / (20000 * np.sqrt(2))])
    rows = []
    maxdiff = maxprec = maxres = maxit = 0
    fig, ax = plt.subplots(2, 2, figsize=(7, 5), sharex=True)
    for col, q in enumerate([(1, 1, 0), (1, 2, 1)]):
        for ai, e in enumerate(amps):
            for sign in [-1, 1]:
                ns = []
                for delta in off:
                    w = 1 + sign * delta
                    y, s, g, r, it = solve(K0, e, w, q)
                    _, s64, _, r64, it64 = solve(K0, e, w, q, 64)
                    _, sd, _, _, _ = solve(K0, e, w, q, 32, False)
                    norm = s / [e, e, e * e * abs(g), e * e * abs(g)]
                    ns.append(norm)
                    rows.append([col + 1, e, w, delta, g, *s, *norm, r, it])
                    maxdiff = max(maxdiff, max(abs(s / s64 - 1)))
                    maxprec = max(maxprec, max(abs(s / sd - 1)))
                    maxres = max(maxres, r, r64)
                    maxit = max(maxit, it, it64)
                ns = np.array(ns)
                for row, ids in [(0, [0, 1]), (1, [2, 3])]:
                    for j in ids:
                        ax[row, col].plot(
                            off, ns[:, j], '-' if sign == 1 else '--',
                            marker=['o', 's', '^', 'd'][ai], ms=3,
                            color=f'C{ai}',
                            label=f'$\\varepsilon=2^{{-{ai + 3}}}$'
                            if row == 0 and col == 0 and (sign == 1)
                            and (j == 0) else None)
        y, s, g, r, it = solve(K0, amps[0], 1, q)
        print('degenerate', col + 1, abs(y[34]), s)
        for row, ids in [(0, [0]), (1, [2, 3])]:
            for j in ids:
                ax[row, col].axhline(ref[j], color='0.25', ls=':', lw=1)
            ax[row, col].set_xscale('log')
            ax[row, col].grid(True, which='both', color='0.9')
        ax[0, col].set_title(['Simple root', 'Double root'][col])
        ax[1, col].set_xlabel('$|\\omega-1|$')
    ax[0, 0].set_ylabel('$\\sigma_{1,2}/\\varepsilon$')
    ax[1, 0].set_ylabel('$\\sigma_{3,4}/(\\varepsilon^2|\\Gamma_Q|)$')
    h, l = ax[0, 0].get_legend_handles_labels()
    fig.legend(h, l, ncol=4, loc='upper center', frameon=False)
    fig.tight_layout(rect=(0, 0, 1, 0.92))
    fig.savefig(OUT / 'singular_scaling.eps', format='eps', bbox_inches='tight')
    plt.close(fig)
    np.savetxt(OUT / 'singular_scaling.csv', rows, delimiter=',',
               header='case,epsilon,omega,offset,gamma,sigma1,sigma2,'
                      'sigma3,sigma4,norm1,norm2,norm3,norm4,'
                      'scaled_residual,iterations', comments='')
    er = []
    for e in 2.0 ** (-np.arange(3, 11)):
        vals = []
        for N in [32, 64]:
```

```
            d = (solve(K0, e, 1, (1, 0, 0), N)[0]
                 - solve(K1, e, 1, (1, 0, 0), N)[0])
            vals.append(float(np.sum(abs(d[N + 3:]) ** 2) / np.sum(abs(d[N + 1:N + 3]) ** 2)))
        er.append([e, *vals, vals[0] / e ** 2])
    a = np.array(er)
    slope = np.polyfit(np.log(a[:, 0]), np.log(a[:, 1]), 1)[0]
    fig, ax = plt.subplots(figsize=(6.2, 3.6))
    ax.loglog(a[:, 0], a[:, 1], 'o-', label='Fourier--Galerkin')
    ax.loglog(a[:, 0], a[-1, 3] * a[:, 0] ** 2, 'k--', label='Reference slope 2')
    ax.set_xlabel('$\\varepsilon$')
    ax.set_ylabel('Relative omitted energy')
    ax.grid(True, which='both', color='0.9')
    ax.legend(frameon=False)
    fig.tight_layout()
    fig.savefig(OUT / 'waveform_energy.eps', format='eps', bbox_inches='tight')
    plt.close(fig)
    np.savetxt(OUT / 'waveform_energy.csv', a, delimiter=',',
               header='epsilon,relative_energy_N32,relative_energy_N64,'
                      'ratio_over_epsilon_squared', comments='')
    print('N32/N64 max relative sv difference', maxdiff,
          'precision difference', maxprec, 'scaled residual', maxres,
          'max iterations', maxit)
    print('energy slope', slope, 'ratio/eps^2', a[-1, 3],
          'N32/N64 energy relative difference',
          max(abs(a[:, 1] / a[:, 2] - 1)))
    print('reference', ref)
    print('closest rows', [r for r in rows if r[1] == amps[-1] and r[3] == off[-1]])
if __name__ == '__main__':
    main()
```