Practical Assignment 5
Fermi–Pasta–Ulam–Tsingou Problem

Computational Physics IV
August 26, 2025
Shubham Pandey

Section 1: Algorithm (natural language)

1. **System & variables**: $N = 32$ oscillators with fixed ends ($q_0 = q_{N+1} = 0$). Mass $= 1$. Nonlinearity parameter α. Displacements q_j and momenta p_j.

2. **Equations of motion**:

$$\ddot{q}_j = (q_{j+1} - 2q_j + q_{j-1}) + \alpha\big[(q_{j+1} - q_j)^2 - (q_j - q_{j-1})^2\big], \quad j = 1, \ldots, N.$$

3. **Time integration**: Velocity Verlet (symplectic, second order). Update positions q_j, compute forces $f_j = \ddot{q}_j$, update momenta p_j. Fixed time step Δt.

4. **Initial condition**: Only the first mode excited:

$$q_j(0) = A \sin \frac{\pi j}{N+1}, \qquad p_j(0) = 0, \quad A = 0.1.$$

5. **Modal projection**: Linear normal modes

$$\phi_{j,k} = \sqrt{\tfrac{2}{N+1}} \sin\left(\tfrac{\pi k j}{N+1}\right),$$

with frequencies $\omega_k = 2 \sin \frac{\pi k}{2(N+1)}$. Modal coordinates: $Q_k(t) = \sum_j \phi_{j,k} q_j(t)$, $P_k(t) = \sum_j \phi_{j,k} p_j(t)$. Modal energy:

$$E_k(t) = \tfrac{1}{2}\big(P_k^2 + \omega_k^2 Q_k^2\big).$$

6. **Diagnostics**:

   - Plot $E_1(t), E_2(t), E_3(t)$ vs time.
   - Heatmap of $E_k(t) / \sum_j E_j(t)$.
   - Recurrence measure

$$R(t) = \frac{\sum_k \big(Q_k(t)Q_k(0) + P_k(t)P_k(0)\big)}{\sum_k \big(Q_k(0)^2 + P_k(0)^2\big)}.$$

7. **Outputs**: Chosen $\Delta t = 0.05$, long total time $T_{\text{total}} \sim$ 20000–100000 to see recurrences.

Section 2: Code

C++ Integrator (fput_alpha.cpp)

```cpp
// fput_alpha.cpp
// Compile: g++ -O2 fput_alpha.cpp -o fput_alpha
// Runs    FPUT    with Velocity Verlet. Outputs:
//  - energies_123.txt : t  E1 E2 E3
//  - modes_time.txt   : t  E1 E2 ... E_N  (full modal energies at each output time)
//  - heatmap_norm.txt : matrix (rows: k=1..N, cols: time-steps) of normalized E_k(t)
//  - R_vs_t.txt       : t R(t)

#include <bits/stdc++.h>
using namespace std;

int main(int argc, char** argv){
    // PARAMETERS (user can tune)
    int N = 32;                     // number of oscillators
    double alpha = 0.25;            // nonlinearity    (change to study TR dependence)
    double A = 0.1;                 // initial amplitude for mode 1
    double dt = 0.05;               // time step
    double T_total = 20000.0;       // total simulation time (increase if recurrence not observed)
    int out_every = 40;             // write outputs every 'out_every' steps
    // parse optional CLI args: alpha, dt, T_total, out_every
    if(argc >= 2) alpha = atof(argv[1]);
    if(argc >= 3) dt = atof(argv[2]);
    if(argc >= 4) T_total = atof(argv[3]);
    if(argc >= 5) out_every = atoi(argv[4]);

    int steps = int(T_total / dt);
    int write_steps = steps / out_every + 2;

    vector<double> q(N+2,0.0), p(N+2,0.0), q_new(N+2,0.0), force(N+2,0.0);

    // MODE FUNCTIONS: phi[j][k] where j=1..N, k=1..N
    vector<vector<double>> phi(N+1, vector<double>(N+1,0.0));
    for(int k=1;k<=N;k++){
        double norm = sqrt(2.0/(N+1));
        for(int j=1;j<=N;j++){
            phi[j][k] = norm * sin(M_PI * k * j / double(N+1));
        }
    }
    // linear frequencies
    vector<double> omega(N+1,0.0);
    for(int k=1;k<=N;k++){
        omega[k] = 2.0 * sin(M_PI * k / (2.0 * (N+1)));
    }

    // Initial condition: excite only first mode
    for(int j=1;j<=N;j++){
```

```
47          q[j] = A * sin(M_PI * j / double(N+1)); // same as phi[j][1] *
                (A / phi_norm)
48          p[j] = 0.0;
49      }
50      q[0]=q[N+1]=0.0;
51
52      // Precompute Qk(0), Pk(0) and denominators for R(t)
53      vector<double> Q0(N+1,0.0), P0(N+1,0.0);
54      for(int k=1;k<=N;k++){
55          double Qk=0.0, Pk=0.0;
56          for(int j=1;j<=N;j++){
57              Qk += phi[j][k] * q[j];
58              Pk += phi[j][k] * p[j];
59          }
60          Q0[k]=Qk; P0[k]=Pk;
61      }
62      double denomR = 0.0;
63      for(int k=1;k<=N;k++) denomR += Q0[k]*Q0[k] + P0[k]*P0[k];
64
65      // Prepare output files
66      ofstream f123("energies_123.txt");
67      ofstream fmodes("modes_time.txt");
68      ofstream fheat("heatmap_norm.txt");
69      ofstream fR("R_vs_t.txt");
70
71      // Write headers
72      f123 << "# t E1 E2 E3\n";
73      fmodes << "# t";
74      for(int k=1;k<=N;k++) fmodes << " E" << k;
75      fmodes << "\n";
76      fR << "# t R\n";
77
78      // temporary storage for normalized energy heatmap columns
79      vector<vector<double>> heat(N+1, vector<double>(write_steps, 0.0));
80
81      // FUNCTION to compute forces f_j from q (indices 1..N)
82      auto compute_force = [&](const vector<double>& qq, vector<double>&
            ff){
83          // fixed ends q0,qN+1 assumed 0 in qq[0] and qq[N+1]
84          for(int j=1;j<=N;j++){
85              double lap = qq[j+1] - 2.0*qq[j] + qq[j-1];
86              double nonlin = alpha * ( (qq[j+1]-qq[j])*(qq[j+1]-qq[j]) -
                    (qq[j]-qq[j-1])*(qq[j]-qq[j-1]) );
87              ff[j] = lap + nonlin;
88          }
89          ff[0]=ff[N+1]=0.0;
90      };
91
92      // initial forces
93      compute_force(q, force);
94
95      int write_idx = 0;
96      // compute initial modal energies and R, write t=0
97      auto compute_modal = [&] (const vector<double>& qq, const vector<
            double>& pp, vector<double>& Ek, vector<double>& Qk, vector<
            double>& Pk){
98          for(int k=1;k<=N;k++){
99              double Q=0.0, P=0.0;
```

```
            for(int j=1;j<=N;j++){
                Q += phi[j][k]*qq[j];
                P += phi[j][k]*pp[j];
            }
            Qk[k]=Q; Pk[k]=P;
            Ek[k] = 0.5 * (P*P + omega[k]*omega[k] * Q*Q);
        }
    };

    vector<double> Ek(N+1,0.0), Qk(N+1,0.0), Pk(N+1,0.0);
    compute_modal(q,p,Ek,Qk,Pk);
    double Esum = 0.0; for(int k=1;k<=N;k++) Esum += Ek[k];
    // write initial lines
    fmodes << 0.0;
    for(int k=1;k<=N;k++) fmodes << " " << Ek[k];
    fmodes << "\n";
    f123 << 0.0 << " " << Ek[1] << " " << Ek[2] << " " << Ek[3] << "\n"
        ;
    double R0 = 0.0;
    for(int k=1;k<=N;k++) R0 += Qk[k]*Q0[k] + Pk[k]*P0[k];
    R0 /= denomR;
    fR << 0.0 << " " << R0 << "\n";
    // fill heat column 0
    for(int k=1;k<=N;k++) heat[k][write_idx] = Ek[k]/Esum;
    write_idx++;

    // MAIN TIME LOOP (Velocity Verlet)
    for(int step=1; step<=steps; ++step){
        // positions half-step update (full-step positions in VV
            formula)
        for(int j=1;j<=N;j++){
            q[j] += p[j]*dt + 0.5 * force[j] * dt*dt; // since mass =1,
                acceleration = force
        }
        // Enforce fixed boundaries
        q[0]=0.0; q[N+1]=0.0;

        // compute new forces at q
        compute_force(q, q_new); // reuse q_new as temporary force
            storage
        // update momenta (full-step)
        for(int j=1;j<=N;j++){
            p[j] += 0.5*(force[j] + q_new[j]) * dt;
        }
        // set force = q_new for next iteration
        force = q_new;

        // outputting
        if(step % out_every == 0){
            double t = step * dt;
            compute_modal(q,p,Ek,Qk,Pk);
            double Esum_now = 0.0; for(int k=1;k<=N;k++) Esum_now += Ek
                [k];
            fmodes << t;
            for(int k=1;k<=N;k++) fmodes << " " << Ek[k];
            fmodes << "\n";
            f123 << t << " " << Ek[1] << " " << Ek[2] << " " << Ek[3]
                << "\n";
```

```
            double R = 0.0;
            for(int k=1;k<=N;k++) R += Qk[k]*Q0[k] + Pk[k]*P0[k];
            R /= denomR;
            fR << t << " " << R << "\n";
            // heat column
            int col = write_idx;
            for(int k=1;k<=N;k++) heat[k][col] = Ek[k] / Esum_now;
            write_idx++;
        }
    } // end time loop

    // write heatmap matrix: rows k=1..N, columns time snapshots
    for(int k=1;k<=N;k++){
        for(int c=0;c<write_idx;c++){
            fheat << heat[k][c];
            if(c+1<write_idx) fheat << " ";
        }
        fheat << "\n";
    }

    fmodes.close(); f123.close(); fheat.close(); fR.close();
    cout << "Done. Outputs: energies_123.txt modes_time.txt
       heatmap_norm.txt R_vs_t.txt\n";
    return 0;
}
```

Python Plotting Script (plot_fput.py)

```
# plot_fput.py
# Usage: python3 plot_fput.py
import numpy as np
import matplotlib.pyplot as plt

# Load E1,E2,E3
data123 = np.loadtxt("energies_123.txt", comments="#")
t = data123[:,0]; E1 = data123[:,1]; E2 = data123[:,2]; E3 = data123
    [:,3]

plt.figure(figsize=(8,4))
plt.plot(t, E1, label='E1')
plt.plot(t, E2, label='E2')
plt.plot(t, E3, label='E3')
plt.xlabel('Time'); plt.ylabel('Modal Energy')
plt.legend(); plt.title('E1, E2, E3 vs time')
plt.grid(True); plt.tight_layout()
plt.savefig('E123_vs_time.png', dpi=200)

# Heatmap
heat = np.loadtxt("heatmap_norm.txt")
plt.figure(figsize=(8,5))
extent = [t[0], t[-1], 1, heat.shape[0]]
plt.imshow(heat, aspect='auto', origin='lower', extent=extent)
plt.colorbar(label='Normalized Ek(t)')
plt.ylabel('Mode index k'); plt.xlabel('Time')
plt.title('Heatmap of normalized modal energy')
plt.tight_layout()
```

```
plt.savefig('heatmap_modes.png', dpi=200)

# R(t)
R = np.loadtxt("R_vs_t.txt", comments="#")
plt.figure(figsize=(7,3.5))
plt.plot(R[:,0], R[:,1])
plt.xlabel('Time'); plt.ylabel('R(t)')
plt.title('Recurrence measure')
plt.grid(True); plt.tight_layout()
plt.savefig('R_vs_t.png', dpi=200)
```

Section 3: Outputs and Discussion

Expected Outputs

- **Energies:** $E_1(t), E_2(t), E_3(t)$ vs time show energy transfer and recurrence.
- **Heatmap:** normalized $E_k(t)$ across modes reveals spreading and refocusing of energy.
- **Recurrence measure:** $R(t)$ shows peaks near 1 indicating recurrence times T_R.

Answers to Questions

1. **Dependence of T_R on α:** Smaller α (weaker nonlinearity) $\Rightarrow$ longer recurrence time. Typically $T_R \propto 1/\alpha$ (order of magnitude).
2. **Energy dissipation:** For weak α, energy does not irreversibly dissipate to all modes. It oscillates among low modes and returns.
3. **Paradoxical behavior:** Statistical mechanics predicts equipartition, but instead the system shows quasi-periodic recurrences and lack of ergodicity.

KAM Theorem

KAM theorem states that for nearly integrable Hamiltonians, most invariant tori survive under weak perturbation. Hence the dynamics remains quasi-periodic, preventing full equipartition. This resolves the paradox.

What I Learned

Imagine a line of $N = 32$ small balls (oscillators) connected by springs in a row. These springs are **not purely linear**; they have a small **nonlinear component**. The ends of the chain are fixed, so the first and last balls cannot move.

Classical intuition suggests that if we give energy to one simple mode of motion, the nonlinearity should allow the energy to spread among all modes, eventually reaching **equipartition** (equal distribution).

Surprisingly, when Fermi, Pasta, Ulam, and Tsingou did their computer experiment in the 1950s, they observed something very counterintuitive:

- Energy initially given to the **first mode** did not spread evenly among all modes.

- Instead, energy **cycled back** to the first mode after some time — a phenomenon called **recurrence**.
- This repeated multiple times: the system seemed to "remember" its initial state.

—

Mathematical Model

Each oscillator j has displacement $q_j(t)$. The equations of motion for the α-FPUT model are:

$$\ddot{q}_j = (q_{j+1} - 2q_j + q_{j-1}) + \alpha \left[(q_{j+1} - q_j)^2 - (q_j - q_{j-1})^2 \right], \quad j = 1, 2, \ldots, N$$

with **fixed ends**:

$$q_0 = q_{N+1} = 0$$

Here:

- The first term is the linear spring force.
- The second term is the nonlinear correction, controlled by the parameter α.

—

Intuitive Example

Think of a **guitar string**:

1. Pluck the string at one end: only the first wave (mode) moves.
2. If the string were fully nonlinear, the energy should scatter into many vibration patterns.
3. In the FPUT system, most of the energy **returns** to the original mode after a while, as if the string "remembers" your pluck.

Why it Matters

The FPUT problem is important because:

- It was one of the first **numerical experiments** in physics.
- It showed that **nonlinear systems do not always behave chaotically**.
- It inspired research on **solitons**, **KAM theory**, and modern nonlinear dynamics.
- It demonstrates a **paradox in statistical mechanics**: energy does not thermalize as expected.