3.3: Modular Programming

Procedural vs. Object Oriented Programming

**Procedural programming.** [verb-oriented]
- Tell the computer to do this.
- Tell the computer to do that.

**Alan Kay’s philosophy.** Software is a *simulation* of the real world.
- We know (approximately) how the real world works.
- Design software to model the real world.

**Objected oriented programming (OOP).** [noun-oriented]
- Programming paradigm based on data types.
- Identify *things* that are part of the problem domain or solution.
- Things in the world know *things*: instance variables.
- Things in the world do *things*: methods.

**Data type:** set of values and operations on those values.

A *Java* class allows us to define a data type by:
- Specifying a set of variables.
- Defining operations on those values.

**Modular programming:** break up a larger program into smaller independent pieces.
- Class = program that defines a data type.
- Client = program that uses a data type.

**Alan Kay.** [Xerox PARC 1970s]
- Inventor of Smalltalk programming language.
- Conceiver of Dynabook portable computer.
- Ideas led to: laptop, modern GUI, OOP.

"The computer revolution hasn’t started yet."

"The best way to predict the future is to invent it."

"If you don’t fail at least 90 percent of the time, you’re not aiming high enough."
Ferromagnetism

Ferromagnetism. Phenomenon by which a material exhibits spontaneous magnetization.
- Explains most of magnetization that arises in everyday life.
- Quantum mechanical cause: spin and Pauli exclusion principle.

Phase transition.
- Curie temperature: ferromagnetism occurs if temperature < $T_C$; does not occur if temperature > $T_C$.
- Ex: for iron $T_C = 1043° K$; for nickel $T_C = 627° K$.
- Study of phase transitions showed defect in mean field theory.

via computational approaches (Ising spin model)

Ising Spin Model

- N-by-N lattice $S$ of cells.
- Cell $(i, j)$ has spin $s_{ij} = \pm 1$.
- Interactions occur only between a cell and its 4 nearest neighbors.

Physical quantities.
- Magnetization: $M(S) = \sum_{i} \sum_{j} s_{ij}$
- Cell energy: $e_{ij} = -J (s_{ij} + s_{i+1,j} + s_{i,j+1} + s_{i-1,j})$

Energy: $E(S) = \sum_{ij} e_{ij}$

we double-counted

Doing physics by tossing dice.
Lattice

N-by-N lattice.
- Typical lab system has \( N = 1 \times 10^9 \).
- Approximate with smaller systems. Why?

Periodic boundary condition.
- Mitigates finite size effect.
- Cell \((i, 0)\) and \((i, N-1)\) are neighbors.
- Cell \((0, j)\) and \((N-1, j)\) are neighbors.
- Topologically equivalent to a torus.

State.java

```java
public class State {   private int N;   private Cell[][] lattice;   public State(int N, double p) {   ... = 0; j < N; j++) {            StdDraw.moveTo(i + 0.5, j + 0.5);            lattice[i][j].draw();         }      }   }
```

Cell.java

```java
public class Cell {   private boolean spin;   public Cell(double p) {   ... spin = (Math.random() < p);   }   public void flip() { spin = !spin; }   public double magnetization() {   if (spin) return 1.0;   else return -1.0;   }   public void draw() {   if (spin) StdDraw.setColor(StdDraw.WHITE);   else StdDraw.setColor(StdDraw.BLUE);   StdDraw.dot(i, j);   }   }
```

State.java (cont)

```java
public double magnetization() {   double M = 0.0;   for (int i = 0; i < N; i++)   for (int j = 0; j < N; j++)   M += lattice[i][j].magnetization();   return M; 
}
```
Boltzmann Energy Distribution

Boltzmann distribution. The probability of finding system in state $S$ is:

$$Bz(S) = \frac{e^{-E(S)/kT}}{Z}$$

where $Z$ is the normalization constant chosen so that $\sum Bz(S) = 1$.

Mean magnetization. $<M> = \sum_{S} M(S) Bz(S)$

Metropolis Algorithm

Goal. Dynamic process to create states with desired probability.

Single-flip dynamics.
- Cell $(i, j)$ randomly flips spin.
- Physical interpretation: result of vibrations in lattice.

Metropolis algorithm. [Metropolis-Rosenbruth-Rosenbruth-Teller-Teller 1953]
- Current state $S$.
- Randomly flip spin of one cell to get state $S'$.
- Let $\Delta E = E(S') - E(S)$ be change in energy.
- If $\Delta E < 0$, update state to $S'$.
- Else, update state to $S'$ with probability $e^{-\Delta E/kT}$.

Theorem. Long run fraction of time process is in state $S = Bz(S)$. 
Spin Over Time

**Low kT:** strong tendency for all spins to align.
**High kT:** strong tendency for spins to cancel out.

State.java

```java
public void metropolis(int i, int j, double kT) {
    double deltaE = -2 * energy(i, j);
    if (deltaE < 0) {
        if (Math.random() <= Math.exp(-deltaE / kT)) {
            cell[i][j].flip();
        }
    } else {
        metropolis(i, j, kT);
    }
}
```

Metropolis Algorithm: Properties

**Estimate physical quantities.** Let $S_t$ be Metropolis state after $t$ phases.

- **Estimate mean magnetization by:**
  \[
  \langle M \rangle = \frac{1}{t} \sum_{i=1}^{t} M(S_i)
  \]

- **Estimate mean energy by:**
  \[
  \langle E \rangle = \frac{1}{t} \sum_{i=1}^{t} E(S_i)
  \]

- **Estimate mean squared energy by:**
  \[
  \langle E^2 \rangle = \frac{1}{t} \sum_{i=1}^{t} E(S_i)^2
  \]

- **Estimate heat capacity by:**
  \[
  C = \frac{1}{kT} \langle E^2 \rangle - \langle E \rangle^2
  \]

**Consequence.** Easy to estimate relevant physical quantities.

Metropolis Visualization

Animate Metropolis algorithm by plotting results after each phase.

```java
public class Metropolis {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int i = (int) Math.random() * N;
        int j = (int) Math.random() * N;
        metropolis(i, j, kT);
    }
}
```
Mean Magnetization

**Approach.** Perform phases; record average value of desired quantity.

```java
// mean magnetization per cell in N-by-N grid, at
// temperature kT, after given number of phases
public static double meanM(int N, double kT, int PHASES) {
    State state = new State(N, 0.5);
    double sumM = 0.0;
    for (int t = 0; t < PHASES; t++) {
        state.phase(kT);
        sumM += state.magnetization() / (N*N);
    }
    return sumM / PHASES;
}
```

### Implementation details.
- How to choose N?
- How many phases is enough?
- Burn-in: run for 8 phases before accumulating statistics.

### Goal
Run a computational experiment and visualize results.
- Run for various values of temperature.
- Plot mean magnetization vs. temperature.

---

**Plotting**

**Diversion.** Plot an array.

```java
public static void main(String[] args) {
    StdDraw.create(512, 512);
    int N = Integer.parseInt(args[0]);
    double[] a = new double[N];
    for (int i = 0; i < N; i++) {
        double t = 2 * Math.PI * i / N;
        a[i] = Math.sin(t) + 0.5 * Math.sin(5*t);
    }
    Plot.lines(a);
    StdDraw.show();
}
```

---

**Scientific Experiment**

**Diversion.** Plot an array.

```java
public static void main(String[] args) {
    StdDraw.create(512, 512);
    int N = Integer.parseInt(args[0]);
    double[] a = new double[N];
    for (int i = 0; i < N; i++) {
        double t = 2 * Math.PI * i / N;
        a[i] = Math.sin(t) + 0.5 * Math.sin(5*t);
    }
    Plot.lines(a);
    StdDraw.show();
}
```

---

\[a(t) = \sin(t) + \frac{1}{2}\sin(5t), \ t = 0..2\pi\]
Plot library. Simple library for plotting points and lines.

```java
public class Plot {
    public static void lines(double[] a) {
        double ymin = ProbStat.min(a);
        double ymax = ProbStat.max(a);
        StdDraw.setScale(0, ymin, a.length - 1, ymax);
        for (int i = 0; i < a.length; i++) {
            StdDraw.moveTo(i-1, a[i-1]);
            StdDraw.lineTo(i, a[i]);
        }
    }

    public static void spots(double[] a) {
        for (int i = 0; i < a.length; i++)
            StdDraw.moveTo(i, a[i]);
            StdDraw.spot();
    }
}
```

Layers of Abstraction

Relationships among data types.

Computational Experiment

Approach. Perform phases; record average value of desired quantity.

```java
public class Experiment {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int PHASES = Integer.parseInt(args[1]);
        double[] m = new double[N];
        for (int i = 0; i < N; i++) {
            kT = 4.0 * i / N;
            m[i] = meanM(N, kT, PHASES);
            m[i] /= (N - N);
        }
        Plot.lines(m);
    }
}
```

plot mean magnetization per spin vs. temperature.
Computational Experiment vs. Theory

**Hypothesis.** Phase transition near $kT = 2.2$, above which material loses its magnetization.

**Conclusion.** Computational experiments agree closely with theory, (and both predict phase transition observed in physical experiments)
Current design. Does not extend to other lattice structures.

OOP design.

- Each Cell knows its spin and its neighbors.
- Each Cell calculates its energy by communicating with its neighbors.
- A State is a list of Cell objects (in any lattice structure).
  
State data type implementation depends on 2D lattice

Bottom line. OOP design makes programs easier to extend.

Other Applications of Ising Model

Applications.

- Spin glasses.
- Protein folds.
- Flocking birds.
- Social behavior.
- Neural networks.
- Beating heart cells.
- Phase separation in a binary alloy.

Variants.

- External magnetic field.
- Different lattice topologies.
- Potts model: spin is one of k discrete values.

Bottom line. With modular design, can reuse code to solve new problems.

Summary

Modular programming.

- Break a large program into smaller independent modules.
- Ex: Cell, State, Experiment, Plot, ProbStat, StdDraw.

Debug and test each piece independently. [unit testing]

- Each class can have its own main.
- Spend less overall time debugging.

Divide work for multiple programmers.

- Software architect specifies data types.
- Each programmer writes, debugs, and tests one.

Reuse code.

- Ex: reuse StdDraw and ProbStat in all kinds of programs.
- Ex: reuse Experiment or Plot with any scientific experiment.
- Ex: reuse Cell for other lattice structures.