บทที่ 4
ผลการวิจัย

ผลของการศึกษาแบบจำลองอนุภาคในบ่อศักย์แบบต่าง ๆ ซึ่งเขียนเป็นโปรแกรมภาษาจาวาแบ่งเป็นหัวข้อดังต่อไปนี้

4.1 ผลลัพธ์การจำลองการทำงานของโปรแกรมใน text mode

4.2 ผลลัพธ์การแสดงภาพกราฟิกของบ่อศักย์และฟังก์ชันคลื่นบนจอภาพ

4.3 โปรแกรมต้นฉบับ (source code)

4.1 ผลลัพธ์การจำลองการทำงานของโปรแกรมใน text mode

การจำลองอนุภาคในบ่อศักย์ สามารถกำหนดให้โปรแกรมแสดงผลเฉพาะตัวอักษร (text mode) เพื่อตรวจสอบความถูกต้องและความคลาดเคลื่อนในการคำนวณของโปรแกรม สามารถแสดงผลได้รวดเร็วขึ้น เพราะหน่วยประมวลผลไม่ต้องไปเสียเวลากับการแสดงภาพกราฟิก

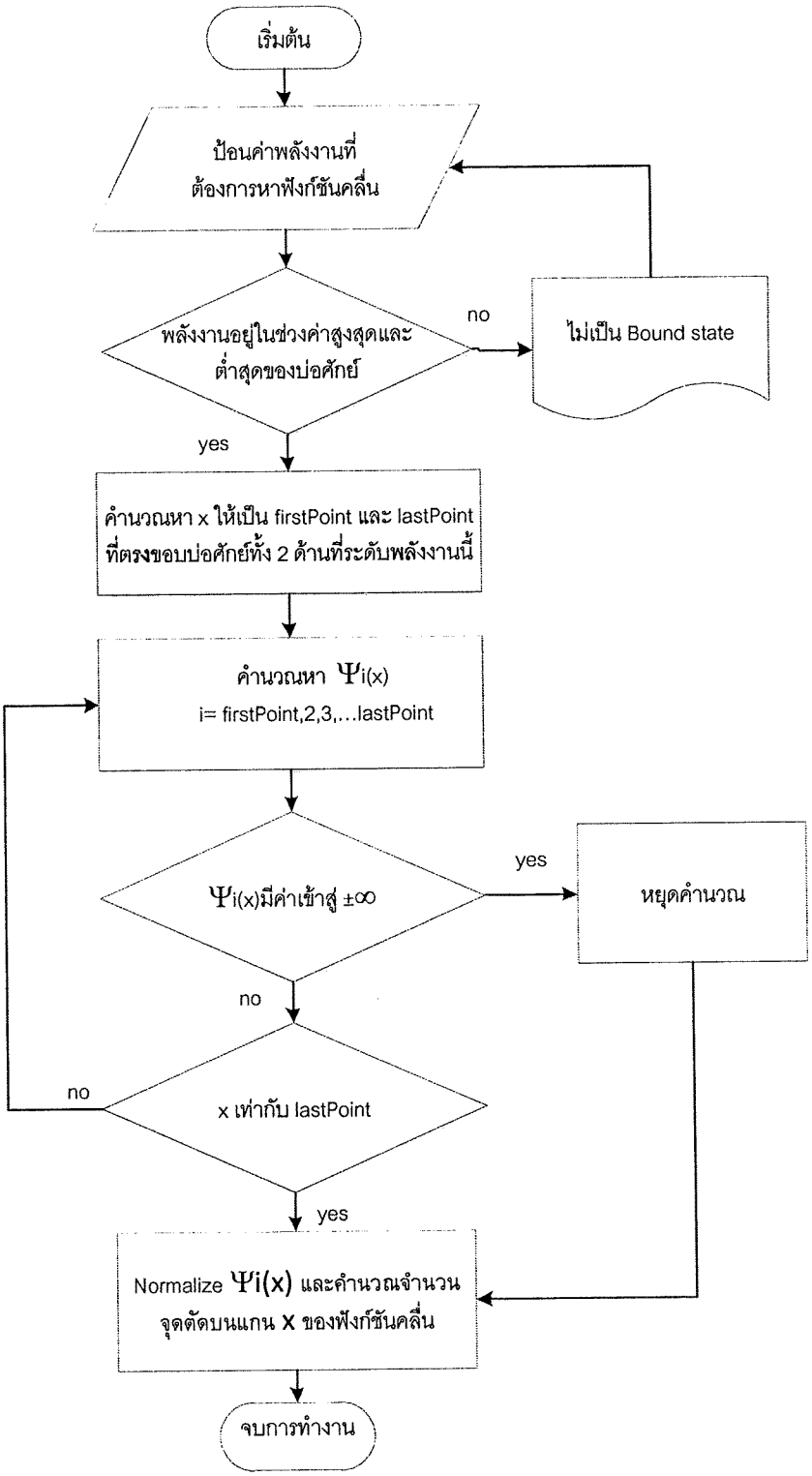
ไม่ว่าจะเป็นบ่อศักย์แบบใดสามารถใช้ระเบียบวิธี(algorithm) ต่อไปนี้คำนวณหาฟังก์ชันคลื่นและพลังงานของอนุภาคได้ทุกแบบ เพื่อให้ง่ายต่อความเข้าใจจึงแยกFlowchart เป็น 2 ส่วน ส่วนแรกเป็น Flowchart ของการหาฟังก์ชันคลื่น และ ส่วนที่ 2 เป็น Flowchart การหาพลังงานของอนุภาค ส่วนที่ 2 นี้ต้องใช้ขั้นตอนการหาฟังก์ชันคลื่นใน Flowchart ส่วนที่ 1 มาเกี่ยวข้องด้วย ในการจัดสร้างโปรแกรมได้รวมการหาฟังก์ชันคลื่น และการคำนวณพลังงานของอนุภาครวมไว้เป็นคลาสเดียวกัน ให้ชื่อคลาสว่า Schrodinger1D

การคำนวณหาฟังก์ชันคลื่นมีขั้นตอนดังต่อไปนี้

1. เลือกรูปแบบพลังงานศักย์ที่จะใช้ในการจำลองสถานการณ์
2. กำหนดขนาดของพลังงาน (E) ที่ต้องการหาฟังก์ชันคลื่น มีค่าไม่เกินค่าสูงสุดของบ่อศักย์ในช่วง x ที่กำหนดไว้
3. คำนวณหาผนังของบ่อศักย์ทั้งด้านซ้ายและด้านขวา ให้ x เป็นค่าแรก (first point) และค่าสุดท้าย (last point) ตรงจุดที่เป็นผนังของบ่อศักย์นี้
4. หาฟังก์ชันคลื่น $\psi(x_i)$ ทุก ๆ ค่าของ x_i i มีค่าตั้งแต่ first point จนถึง last point แต่ละค่า i ให้ตรวจสอบฟังก์ชันคลื่นว่าลู่ออกเกินกว่าค่าที่จำกัดไว้หรือไม่ ถ้าเกินให้หยุดการทำงาน
5. นับจำนวนจุดที่ฟังก์ชันคลื่นตัดกับแกน x
6. ทำการ normalize ฟังก์ชันคลื่นที่คำนวณได้

ขั้นตอนทั้งหมดสามารถเขียนเป็น Flow chart ได้ดังนี้

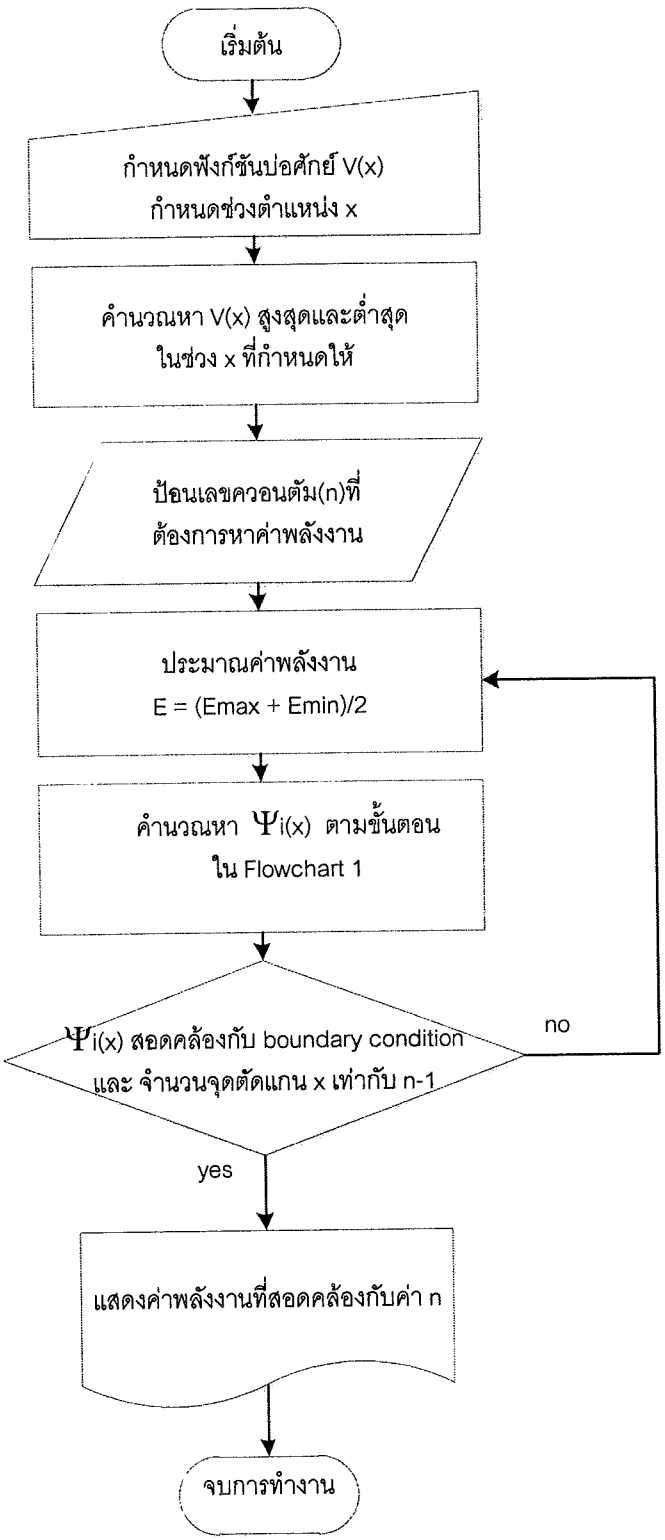
Flowchart 1: แสดงการหาฟังก์ชันคลื่นของอนุภาค
เมื่อกำหนดพลังงาน



การคำนวณหาพลังงานของอนุภาค (E) ซึ่งเป็นค่าเจาะจงของฟังก์ชันคลื่น มีขั้นตอนดังนี้

1. เลือกรูปแบบพลังงานศักย์ที่จะใช้ในการจำลองสถานการณ์
2. คำนวณพลังงานศักย์ค่าสูงสุดและต่ำสุดที่เป็นไปได้ในช่วง x ที่กำหนด
3. กำหนดระดับพลังงาน หรือเลขควอนตัม n ที่ต้องการหาพลังงาน
4. เริ่มต้นสุ่มค่าพลังงานที่ระดับพลังงาน n นี้ โดยกำหนดให้ $E_{expected} = \frac{E_{max} + E_{min}}{2}$
5. นำค่า $E_{expected}$ ไปหาฟังก์ชันคลื่น ถ้าไม่สอดคล้องกับเงื่อนไขขอบเขต และจำนวนจุดตัดแกน x ยังไม่เท่ากับ n – 1 ให้กลับไปทำข้อ 4 ใหม่ จนกว่าจะได้ฟังก์ชันคลื่น และพลังงานที่สอดคล้องกับเงื่อนไขขอบเขตและ จำนวนจุดตัดแกน x ของฟังก์ชันคลื่น = n – 1
6. ถ้าพบว่า $\psi(x)$ สอดคล้องกับเงื่อนไขขอบเขต และจำนวนจุดตัดแกน x ที่ได้ มีค่าเท่ากับ n – 1 แสดงว่า $\psi(x)$ เป็นฟังก์ชันเจาะจง และค่า E ที่ได้ในครั้งหลังสุดนี้เป็นค่าเจาะจงที่สอดคล้องกับฟังก์ชันคลื่นนี้เช่นกัน ให้จบการทำงานที่ขั้นตอนนี้

Flowchart 2: แสดงการคำนวณพลังงานของ
อนุภาคเมื่อกำหนดเลขควอนตัม n



นำขั้นตอนทั้งหมด เขียนเป็น source code ภาษาจาวา เพื่อหาคำตอบของสมการชเรอดิงเงอร์ แบบ 1 มิติ ดังนี้

โปรแกรม 4.1 Schrodinger1D.java

```
1:  /**
2:   * Title: Schrodinger1D
3:   * Description: compute energy, eigen state and Wave function
4:   *
5:   * @author: Wach R
6:   *@ version 0.1
7:   * Date : April 23, 2011
8:   * Last updated: May 2, 2011
9:   *============================
10:  * Schrodinger Eq :  d^2psi/dx^2 = 2*m/(hbar)^2(E-V)*psi
11:  *  define  k^2 = 2*m/(hbar)^2(E-V)
12:  * in atomic unit:  hbar = 1, m = 1
13:  */
14:  package classes.SE1D;
15:
16:  import static classes.MathTools.Common.CommonConstants.*;
17:  import classes.MathTools.Common.Function;
18:
19: public class  Schrodinger1D
20: {
21:
22:  public static final int NO_ERROR = 0;
23:  public static final int MAXIMUM_NODE=20;
24:  int errorCode = 0;
25:
26: double energy;
27: double potentialMin;
28: double potentialMax;
29: double[] psi;  // wave function of the particle.
30: double[] x;  // positions on x axis
31: double xmin, xmax;  // min and max values of x
32:
33: double tolerance = 1.0e-3;
34: //  double tolerance =DEFAULT_TOLERANCE;
35:  double maxAmplitude;  // maximum amplitude of psi
36:   int firstPoint, lastPoint;
37:   double dx;
38:   Function potential;
39:
40:   double psiState ;   // keep current psi
41:   double  slopeState;  // keep d psi / dx
42:   double  xState;    // keep x value
```

```
43:
44:     public Schrodinger1D (Function potential, double xmin,
                                double xmax, int numberOfPoints) {
45:     this.potential = potential;
46:     psi = new double[numberOfPoints];
47:     x= new double[numberOfPoints];
48:     this.xmin = xmin;
49:     this. xmax = xmax;
50:     dx = (xmax -xmin)/(numberOfPoints -1);
51:     firstPoint = 0;
52:     lastPoint = numberOfPoints;
53:     findMinAndMaxPotential();
54:     }
55:
56:     private void  findMinAndMaxPotential() {
57:                 double xValue = xmin;
58:                 double newPotential;
59:       potentialMin = potential.Of(xValue);
60:                 potentialMax = potentialMin;
61:       for(int i = 0; i <  psi.length; i++) {
62:         x[i] = xValue;
63:         newPotential = potential.Of(xValue);
64:         if(potentialMin > newPotential)
                                potentialMin = newPotential;
65:                   if(potentialMax < newPotential)
                                potentialMax = newPotential;
66:         xValue += dx;
67:       }
68:
69:     }
70:     /**
71:       * finds  the area where the wave function is non-zero.
72:       * and  bounded by the firstPoint and lastPoint.
73:       *
74:       * @param energy
75:       */
76:      protected void findFirstAndLastPoint(double energy) {
77:        double  current_x = xmin;
78:                 double startX=0, stopX=0;
79:
80:     if(( energy <= potentialMin) || (energy >= potentialMax))  {
81:                                 System.out.println( "It's  not a bound state,
                                    check your Potential and Energy again");
82:                                 return;
83:                 }
84:
```

```
85:          // find the first point where E>V
86:
87:          for(int i = 0; i< psi.length;  i++) {
88:                                   if((energy - potential.Of(current_x))>0)  {
89:             firstPoint = i;
90:                                  startX = current_x;
91:             break;
92:           }
93:          current_x += dx;
94:         }
95:         current_x = xmax;
96:
97:         // find the last point where E>V
98:         for(int i = psi.length; i>firstPoint; i--) {
99:                                   if((energy - potential.Of(current_x)) > 0) {
100:                                  lastPoint = i;
101:                                  stopX= current_x;
102:            break;
103:          }
104:          current_x -= dx;
105:         }
106:
107:       // Actually, wave function value is not zero immediately
108:       // it still decay at the both sides
109:       // Left side :
110:          double decay = 1;
111:          current_x = xmin+firstPoint*dx;
112:          while(decay > tolerance && firstPoint > 0) {
113:            firstPoint--;
114:           current_x -= dx;
115:                   // this region E < V  the result shoud be minus sign
116:         double k = Math.sqrt(2*Math.abs(energy-potential.Of(current_x)));
117:           decay *= Math.exp(-k*dx);
118:       //System.out.print(" left side: k =  " + k + " decay = " +decay);
119:
120:          }
121:                  // Right side:
122:         decay = 1;
123:         current_x = xmin+lastPoint*dx;
124:          // assume exponential decay on right hand side
125:          while(decay > tolerance && lastPoint < psi.length) {
126:            lastPoint++;
127:            current_x += dx;
128: double k = Math.sqrt(2*Math.abs(energy-potential.Of(current_x)));
129:
130:            decay *= Math.exp(-k*dx);
```

```
131:
132:          }
133:        }
134:
135:        /**
136:         * Solves the Schroedinger Equation with the given energy.
137:         * @param energy double the energy
138:         * @return int node  ( number of zero crossing of wave function
139:         */
140:        public int solveWaveFunction(double energy) {
141:          int node = 0;  // count the number of zero crossings.
142:          boolean slopeChanged = false;
143:          maxAmplitude = 0;
144:          double integral = 0;
145:                  double lastPsiState, lastSlopeState,lastXState;
146:                  double k2;   //  k^2 = 2*(E-V)
147:                  double  k2LastState, k2State;
148:                  double slope;
149:
150:                  this.energy = energy;
151:          findFirstAndLastPoint(energy);
152:          if(lastPoint - firstPoint <3) {
153:        //if number of point in range < 3  cannot use with Numerov method
154:            return 0;
155:          }
156:          // set wave function to zero outside the bound region
157:          for(int i = 0;i<firstPoint;i++) {
158:            psi[i] = 0;
159:            x[i] = xmin+i*dx;
160:          }
161:                  // state of the firstPoint
162:                  lastPsiState=0;
163:                  lastSlopeState = 0;
164:                  lastXState =xmin+firstPoint*dx;
165:                  psi[firstPoint]=lastPsiState;
166:                  x[firstPoint] = lastXState;
167:                  k2LastState = 2.0*(energy - potential.Of(lastXState));
168:                  // state of the firstPoint +1
169:                  psiState = dx;
170:          slopeState = 1.0;  // initial derivative of psi by dx
171:          xState =  xmin + (firstPoint+1)*dx;   // initial value of x
172:                  psi[firstPoint+1]=psiState;
173:                  x[firstPoint+1] = xState;
174:                  k2State = 2.0*(energy - potential.Of(xState));
175:                  slopeState = (psiState - lastPsiState)/dx;
176:        // find wave function (psi) with Numberov method
```

```
177:    // the differential eq must be in a form
178:    //                                   d^2 psi /dx^2  = -k^2 (psi)
179:    //                                    define  k^2 = 2*m/(hbar)^2(E-V)
180:    // ------------------------------------
181:    double  c =  dx*dx/12.0;  // coeficient for Numerov Algorithm
182:       for(int i = firstPoint+2; i<lastPoint; i++) {
183:                              x[i]= xmin+i*dx;
184:                              k2 = 2.0*(energy - potential.Of(x[i]));
185:    // System.out.print ( " i =  "  + i + "     k in loop =  " + k2);
186:
187:                              psi[i] = ((2.0 -10.0*c*k2State)*psiState -
                                 (1+ c*k2LastState)*lastPsiState)/(1+c*k2);
188:                              slope = (psi[i] - psiState)/dx;
189:
190:                              // keep previous values for next step
191:                  lastPsiState=psiState;
192:                  lastSlopeState = slopeState;
193:                  lastXState =xState;
194:                  k2LastState = k2State;
195:
196:     // Now store current values
197:                  psiState = psi[i];
198:                  k2State = k2;
199:                   xState = x[i] ;
200:                   slopeState = slope;
201:
202:          if(maxAmplitude<Math.abs(psiState)) {
203:            maxAmplitude = Math.abs(psiState);
204:          }
205:    // Normalize  wave function  with trapezoidal rule
206:    //  Integrate f(x) dx =  deltaX ( summation fi - f0/2 -fn/2)
207:
208:                    integral += psiState*psiState;
209:
210:                   if(slopeState*lastSlopeState < 0)
211:              slopeChanged = true;
212:
213:                     if(psiState*lastPsiState < 0)  {
214:              node++;
215:              slopeChanged = false;
216:            }
217:            if(Math.abs(psiState)>1.0e12) { //prevent diverging so much
218:              for(int m = i+1;m<lastPoint; m++)
219:                psi[m] = psiState;
220:                              x[m] = xmin + m*dx;
221:              }
```

```
222:              break;
223:            }
224:          } // end of ----->  for(int i = firstPoint+2
225:
226:          integral *= dx;
227:                  integral = integral -( psi[firstPoint]/2.0 +
                      psi[lastPoint-1]/2.0)*dx;
228:                  // find the magnitude of wave function
229:                  double sqrtIntegral = Math.sqrt(integral);
230:
231:                  // fill the rest point at the right side
232:          for(int i = lastPoint; i < psi.length; i++) {
233:            psi[i] = psi[lastPoint-1];
234:            x[i] = xmin+i*dx;
235:          }
236:        /*
237:          // check to see if last value is close enough to a crossing
238:if(slopeChanged&&(Math.abs(phi[phi.length-1])<=tolerance*maxAmp)) {
239:            crossing++;
240:          }
241:        */
242:
243:          normalize(sqrtIntegral);
244:
245:                  // check normalize it shoulld equals  1
246:        // double sumOfPsi2 =0;
247:        //          for(int i = firstPoint; i <lastPoint; i++) {
248:        //            sumOfPsi2  += psi[i]*psi[i];
249:        //          }
250:        //          System.out.println(" sum of  Psi ^2 = " + sumOfPsi2*dx );
251:          return node;
252:        }
253:        //  Normalize wave function
254:         private void normalize(double magnitude) {
255:          if(magnitude==0) {
256:            return;
257:          }
258:          for(int i = 0; i< psi.length; i++) {
259:            psi[i] /= magnitude;
260:            psiState /= magnitude;
261:           slopeState /= magnitude;
262:          }
263:          maxAmplitude /= magnitude;
264:        }
265:
266:        /**
```

```
267:      * Solves the eigen energy  with the given energy level
268:      * @param int quantumNumber
269:      * @return int node  ( number of zero crossing of wave function
270:      */
271:     public double[ ] solveEigenEnergy( int quantumNumber)
272:     {
273:     double eMin;
274:     double eMax;
275:     int nodeCount=0;
276:     int counter = 0;
277:
278:               if(quantumNumber > MAXIMUM_NODE) {
279:               System.out.println("Quantum  Number must not  greater
                  than  " + MAXIMUM_NODE);
280:               return new double[psi.length];
281:     }
282:               eMin = potentialMin;
283:               eMax = eMin +1;
284:     while (counter < 20 && nodeCount <= quantumNumber) {
285:               nodeCount = solveWaveFunction(eMax);
286:
287:     //Increase energy until the number of node is greater
          than quantum number
288:               eMax  = eMax +( eMax-eMin);
289:               counter++;
290:     }
291:     counter = 0;
292:
293:     do{
294:               double tempEnergy = (eMax + eMin)/2.0;
295:               int crossing = solveWaveFunction(tempEnergy);
296:                 System.out.println( counter + "  energy =  " +
                  tempEnergy + " and crossing zero =   " + crossing);
297:
298:               if((crossing == quantumNumber) &&
                  (Math.abs(psi[psi.length-1]) <= tolerance*maxAmplitude)) {
299:                         errorCode = 0;
300:     //System.out.println("in IF #1:  psi[psi.lenth-1] = " +
                  psi[psi.length-1] + "     tolerance*maxAmp = "
                  + tolerance*maxAmplitude );
301:                         return psi;
302:               }
303:     if((crossing > quantumNumber) || ((crossing == quantumNumber)
          &&(checkEvenNumber(crossing)*psi[psi.length-1] > 0)) ) {
304:                         eMax = tempEnergy;
305:
```

```
306:
307:                    }
308:                    else {
309:                                    eMin = tempEnergy;
310:
311:
312:                    }
313:                    counter ++;
314:                    System.out.println();
315:          } while (counter < 32 && (eMax-eMin )> ZERO_APPROACH);
316:          errorCode = 1;
317:          return new double[psi.length];
318:          }
319:            /**
320:             * Gets the x coordinates
321:             */
322:            public double[] getXCoordinates() {
323:              return x;
324:            }
325:            public double[] getWaveFunction() {
326:              return psi;
327:            }
328:           public double getEnergy() {
329:                    return energy;
330:           }
331:            /**
332:             * Gets the error code from computation.
333:             * @return int
334:             */
335:            public int getError() {
336:              return errorCode;
337:            }
338:            private int checkEvenNumber(int number) {
339:                        if(number%2==0)
340:                                    return 1;  // Even
341:                        else
342:                                    return -1; //Odd
343:             }
344:           }
345:
```

โปรแกรม 4.1 Schrodinger1D.java มีส่วนที่เป็นการฟังก์ชันคลื่น (ใน flowchart ที่ 1) และส่วนที่หาขนาดพลังงานที่เลขควอนตัมใดๆ (ใน flowchart ที่ 2)

บรรทัดที่ 140 ถึง 264 เป็น method solveWaveFunction มีรูปแบบการใช้งานดังนี้

```
public int solveWaveFunction(double energy)
```

เมธอดนี้ใช้หาฟังก์ชันคลื่นของอนุภาคที่มีพลังงาน เท่ากับค่า energy ลำดับการทำงานของเมธอดนี้เป็นไปตามที่เขียนไว้ใน Flowchart ที่ 1 มีรายละเอียดปลีกย่อยได้แก่ บรรทัดที่ 180 – 224 เป็นการหาค่า ψ_i โดยใช้วิธีหาผลเฉลยของสมการอนุพันธ์อันดับสองของ นูเมอรอฟ (Numerov) บรรทัดที่ 213-215 เป็นการตรวจสอบว่า ψ_i ตัดแกน x หรือไม่ ถ้ามีการตัดแกน x จะนับจำนวนครั้งที่ตัดแกน x ไว้ในตัวแปรที่ชื่อว่า node บรรทัดที่ 217 -223 เป็นตรวจสอบการลู่ออกของฟังก์ชันคลื่น ψ_i ถ้าค่า ψ_i ที่คำนวณได้มีค่ามากกว่า 10^{12} การวนรอบหาฟังก์ชันคลื่นจะหยุดคำนวณทันที บรรทัดที่ 205 -208 เป็นการปริพันธ์ของฟังก์ชันคลื่น เพื่อเตรียมไว้สำหรับ normalize โดยอาศัยวิธีรวมพื้นที่สี่เหลี่ยมคางหมูเล็ก ๆ ทั้งหมด จากนั้นนำผลลัพธ์ที่ได้จากการหาปริพันธ์ไปทำการ normalize โดยใช้เมธอด normalize ในบรรทัดที่ 254 -264

เมธอด solveWaveFunction ส่งค่าคืนกลับเป็นจำนวนจุดตัดของฟังก์ชันคลื่นที่ผ่านแกน x

บรรทัดที่ 271 ถึง 318 เป็น method solveEigenEnergy มีรูปแบบการใช้งานดังนี้

```
public double[ ] solveEigenEnergy( int quantumNumber)
```

เมธอดนี้ใช้หาขนาดของพลังงานที่ระดับพลังงานหรือเลขควอนตัม n ใด ๆ ลำดับการทำงานเป็นไปตาม Flowchart 2 มีการเรียกใช้ method solveWaveFunction ในบรรทัดที่ 295 การวนรอบหาฟังก์ชันคลื่นจะดำเนินไปเรื่อย ๆ จนกว่าจะได้ฟังก์ชันคลื่นที่สอดคล้องเงื่อนไขขอบเขตและสอดคล้องกับเลขควอนตัม n ขนาดของพลังงานที่ได้จะเป็นค่าเจาะจง เมธอดนี้ส่งค่าคืนกลับเป็นเซตของฟังก์ชันคลื่นที่สอดคล้องกับเงื่อนไขขอบเขตและเป็นฟังก์ชันเจาะจง

ในการทดสอบการทำงานของคลาส Schrodinger1D ทำได้โดยสร้างโปรแกรมเรียกใช้งานคลาสดังกล่าว ตัวอย่างต่อไปนี้เป็นการสร้าง Application ชื่อ NSEApp.java (Appllication for Schrodinger equation with quantum number n) ใช้พลังงานศักย์เป็น Harmonic oscillator เป็นตัวอย่าง รายละเอียดของโปรแกรมมีดังนี้

โปรแกรม 4.2 NSEApp.java

```
1:  /**
2:  * Title: NSEApp.java  (Schrodinger Application)
3:  * Description : test for Schrodinger App
4:  * @author: Wach R
5:  *@ version 0.1
```

```
6:      * Date : April 28, 2011
7:      *============================
8:       */
9:      package  JApp.SE1D;
10:
11:     import java.util.*;
12:     import java.io.*;
13:     import classes.SE1D.*;
14:
15:     public class NSEApp {
16:
17:       /**
18:        * Starts the Java application.
19:        * @param args[]  the input parameters
20:        */
21:       public static void main(String[] args) throws Exception {
22:         int numberOfPoints = 300;
23:         double xmin = -5, xmax = +5;
24:                 int n; // quantum number
25:                 double en;
26:                 String fileName = "NPsi";
27:                 int crossing=0;
28:
29:     System.out.println("\nThis program finding eigen energy of
        a particle\n in harmonic oscillator");
30:     System.out.print(" You must input n (quantum no. (from 1 to 30))
        ---------->");
31:                             Scanner  console = new Scanner(System.in);
32:                                     n = console.nextInt();
33:
34:         Schrodinger1D  se = new Schrodinger1D(new Quadratic(),
                xmin, xmax, numberOfPoints);
35:     //  Schrodinger1D  se = new Schrodinger1D(new SquareWell(2.0,
                -10.0), xmin, xmax, numberOfPoints);
36:
37:                 double[] psi = se.solveEigenEnergy(n);
38:         double[] x = se.getXCoordinates();
39:
40:                 if (se.getError()==se.NO_ERROR) {
41:           System.out.println("energy="+se.getEnergy());
42:                             PrintWriter w = new PrintWriter (new
                                    FileOutputStream(fileName+n), true);
43:                                         for(int i = 0; i < x.length ; i++)
44:                                             w.println( x[i]+"\t"+psi[i]);
45:
46:           } else {
```

```
47:             System.out.println ("There must be some errors");
48:         }
49:     }
50:     }
```

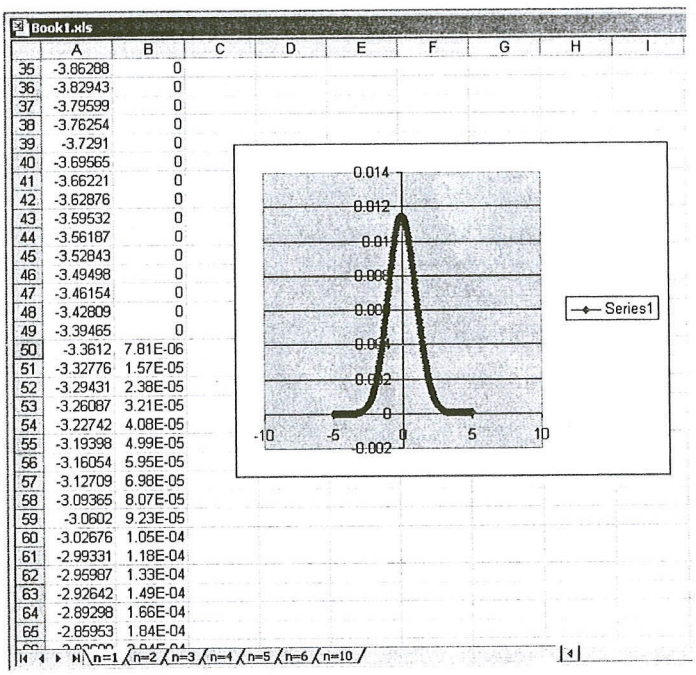
โปรแกรม 4.2 นี้ แสดงตัวอย่างการจำลองอนุภาคในบ่อศักย์กรณีที่เป็น Harmonic oscillator การทำงานของโปรแกรม NSEApp.java เริ่มต้นด้วยการป้อนค่า n ที่ต้องการทราบพลังงาน

บรรทัดที่ 34 คือบรรทัดที่กำหนดแบบของบ่อศักย์ V(x) ถ้าต้องการทดสอบกับบ่อศักย์ชนิดอื่น ๆ ให้เปลี่ยน ฟังก์ชันของบ่อศักย์ที่บรรทัดนี้

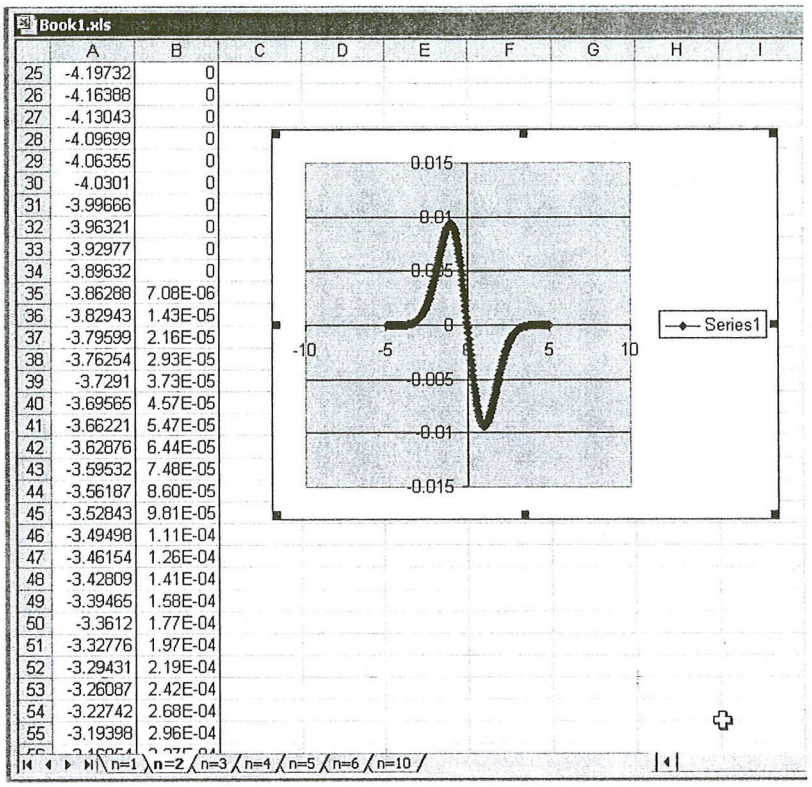
บรรทัดที่ 37 โปรแกรมจะนำค่า n ไปคำนวณพลังงาน โดยเรียกใช้ฟังก์ชัน se.solveEigenEnergy(n) แสดงผลพลังงานที่คำนวณได้ในบรรทัดที่ 41

บรรทัดที่ 42 – 45 เป็นการบันทึกค่าฟังก์ชันคลื่นที่คำนวณได้ลงในแฟ้มข้อมูล ชื่อ NPsi1, NPsi2, ... ชื่อแฟ้มข้อมูลจะเปลี่ยนไปตามเลขควอนตัมที่ป้อนทางอินพุต การบันทึกเก็บไว้ในไฟล์จะเป็นประโยชน์สำหรับการนำไปตรวจสอบฟังก์ชันคลื่นว่ามีคุณสมบัติสอดคล้องกับเงื่อนไขขอบเขตหรือไม่ สามารถนำไปพล็อตกราฟ โดยใช้โปรแกรมตารางคำนวณ Excel เพื่อศึกษารูปกราฟว่าเป็นไปตามทฤษฎีหรือไม่

จากรูปที่ 4.1 และ 4.2 เป็นการนำฟังก์ชันคลื่นที่คำนวณได้ กรณี n = 1 และ n = 2 ไปพล็อตกราฟในโปรแกรม Excel จะเห็นว่าจะได้เส้นกราฟสอดคล้องกับทฤษฎี กล่าวคือ ฟังก์ชันคลื่นจะหายไปตรงผนังของบ่อ จำนวนจุดตัดแกน x จะมีค่าเท่ากับ n-1

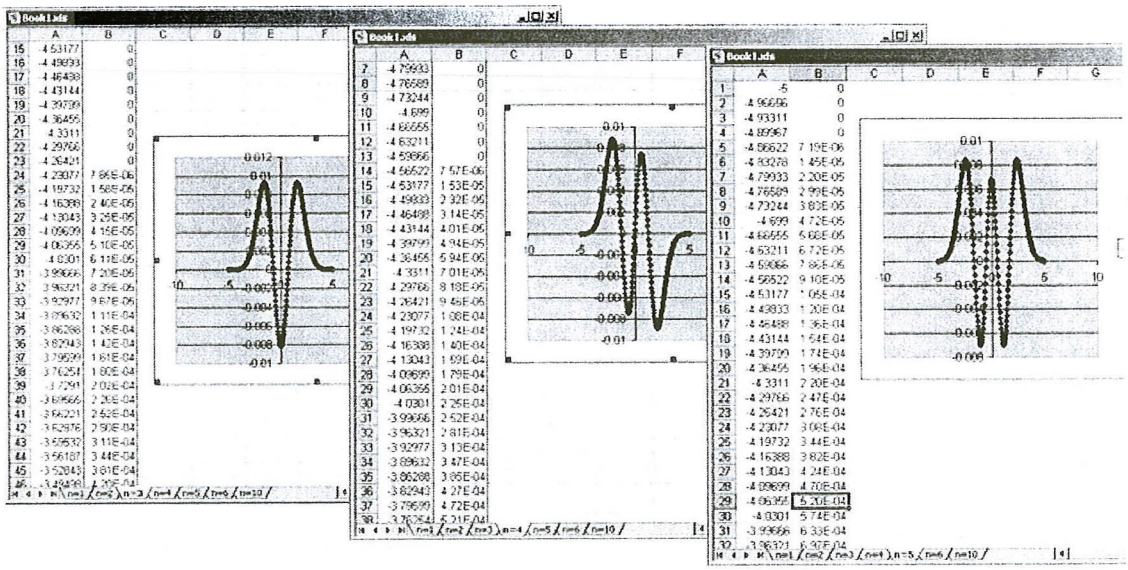


รูปที่ 4.1 แสดงการนำฟังก์ชันคลื่นกรณี n = 1 พล็อตกราฟในโปรแกรม Excel



	A	B
25	-4.19732	0
26	-4.16388	0
27	-4.13043	0
28	-4.09699	0
29	-4.06355	0
30	-4.0301	0
31	-3.99666	0
32	-3.96321	0
33	-3.92977	0
34	-3.89632	0
35	-3.86288	7.08E-06
36	-3.82943	1.43E-05
37	-3.79599	2.16E-05
38	-3.76254	2.93E-05
39	-3.7291	3.73E-05
40	-3.69565	4.57E-05
41	-3.66221	5.47E-05
42	-3.62876	6.44E-05
43	-3.59532	7.48E-05
44	-3.56187	8.60E-05
45	-3.52843	9.81E-05
46	-3.49498	1.11E-04
47	-3.46154	1.26E-04
48	-3.42809	1.41E-04
49	-3.39465	1.58E-04
50	-3.3612	1.77E-04
51	-3.32776	1.97E-04
52	-3.29431	2.19E-04
53	-3.26087	2.42E-04
54	-3.22742	2.68E-04
55	-3.19398	2.96E-04

รูปที่ 4.2 แสดงการนำฟังก์ชันคลื่นกรณี n = 2 พล็อตกราฟในโปรแกรม Excel



รูปที่ 4.3 กราฟของฟังก์ชันคลื่น กรณี n =3, 4 และ 5

ผลลัพธ์จากการจำลองอนุภาคในบ่อศักย์แบบสี่เหลี่ยม (1 dimensional finite rectangular well)

อนุภาคมวล m เคลื่อนที่ในบ่อศักย์ที่มีความกว้างเท่ากับ 2a ความลึกของบ่อศักย์ V ดังนี้

$$V = \begin{cases} -V_0 & if \; -a < x < a \\ 0 & if \quad |x| \geq a \end{cases}$$

นำไปเขียนเป็นโปรแกรม เพื่อเรียกใช้ฟังก์ชันของบ่อศักย์ดังนี้

โปรแกรม 4.3 SquareWell.java

```
1:      /**
2:      * Title: SquareWell.java
3:      * Description Square well potential.
4:      *
5:      * @author: Wach R
6:      *@ version 0.1
7:      * Date : April 24, 2011
8:      *============================
9:       */
10:      package classes.SE1D;
11:
12:      import classes.MathTools.Common.Function;
13:
14:     public class SquareWell  extends Function {
15:     double halfWidth;  // halfWidth of potential well( -a <= x <= a )
16:                     double dept;//the depth of well  must be minus sign
17:     public          SquareWell( double width, double dept) {
18:                             halfWidth = width;
19:                             this.dept = dept;
20:                     }
21:
22:       public double Of(double x) {
23:                       if(  Math.abs(x) >= halfWidth)
24:                                 return 0;
25:                       else
26:                                           return  dept;
27:       }
28:
29:     }
30:
```

การคำนวณพลังงาน (E) ของอนุภาคในบ่อศักย์ที่ระดับพลังงาน n ใด ๆ โดยวิธีวิเคราะห์สามารถหาได้จาก สมการ (2.32)

$$2a\sqrt{\frac{2m}{\hbar^2}(V_0 - E)} = (n-1)\pi + 2\tan^{-1}\sqrt{\frac{E}{V_0 - E}}$$

ในหน่วยอะตอม สมการจะถูกลดรูปเหลือเพียง

$$2a\sqrt{2(V_0 - E)} = (n-1)\pi + 2\tan^{-1}\sqrt{\frac{E}{V_0 - E}} \qquad (4.1)$$

ในที่นี้จะใช้สมการ (4.1) ในการคำนวณหาพลังงาน E เมื่อกำหนดค่า n =1, 2, 3 ..., 6 ให้ความกว้างของบ่อ เท่ากับ 4 หน่วย (a = 2) และความลึกของบ่อศักย์เท่ากับ 10 หน่วย เนื่องจากสมการ (4.1) เป็นสมการประเภท transcendental function ไม่สามารถใช้วิธีวิเคราะห์ใด ๆ คำนวณหารากสมการ (ในที่นี้คือ E) ได้ จะต้องใช้วิธีคำนวณเชิงตัวเลขเข้ามาช่วย ซอฟต์แวร์ที่จะใช้คำนวณในงานวิจัยนี้คือ MathematicA รุ่น 5.1 และใช้การหารากสมการโดยวิธีแบ่งครึ่งช่วง (Bisection method) โดยเขียนเป็นโปรแกรมคอมพิวเตอร์แล้วป้อนข้อมูลที่กำหนดให้ เพื่อใช้ในการคำนวณ (ดูรายละเอียดการคำนวณของโปรแกรม MathematicA และโปรแกรมต้นฉบับ (source code) ที่ใช้วิธีแบ่งครึ่งช่วงหารากสมการ ในภาคผนวก 3) ผลที่ได้ปรากฏดังในตารางที่ 4.1 ดังนี้

n (ระดับพลังงาน)	E จาก MathematicA	E จากวิธีแบ่งครึ่งช่วง	E จากแบบจำลอง ฯ
1	-9.7507	-9.750697	-9.752279
2	-9.00542	-9.005423	-9.011693
3	-7.77297	-7.772970	-7.786852
4	-6.07186	-6.071860	-6.095848
5	-3.94146	-3.941455	-3.977063
6	-1.49125	-1.491250	-1.536339

ตาราง 4.1 เปรียบเทียบพลังงานของอนุภาค (E) ที่ n ค่าต่างๆ กัน ของบ่อศักย์แบบสี่เหลี่ยม

ผลลัพธ์ที่ได้จากการจำลองอนุภาคในบ่อศักย์แบบสี่เหลี่ยมที่มีความลึก -10 หน่วย และความกว้าง 4 หน่วย จะได้พลังงานที่เป็นค่าเจาะจง ที่ n = 1, 2, ..., 5, 6 มีค่าใกล้เคียงกับค่าที่ได้จากการคำนวณโดยใช้โปรแกรมสำเร็จรูป MathematicA และวิธีคำนวณเชิงตัวเลขโดยวิธีแบ่งครึ่งช่วง ถูกต้องที่ทศนิยมตำแหน่งที่ 1

ผลลัพธ์จากการจำลองอนุภาคในบ่อศักย์เป็นฮาร์มอนิก ออสซิลเลเตอร์

(1 dimensional harmonic oscillator)

อนุภาคเคลื่อนที่ในบ่อพลังงานศักย์แบบฮาร์มอนิก ออสซิลเลเตอร์ ซึ่งมีรูปสมการดังนี้

$$V(x)=\frac{1}{2}m\omega^2x^2$$

ω คือความเร็วเชิงมุมของการสั่นของอนุภาคในหน่วย radian/sec

นำไปเขียนเป็นโปรแกรม เพื่อเรียกใช้ฟังก์ชันของบ่อศักย์ดังนี้

โปรแกรม 4.4 Quadratic.java

```
1:  /**
2:   * Title: Quadratic.java
3:   * Description Simple harmonic oscillator potential.
4:   *
5:   * @author: Wach R
6:   *@ version 0.1
7:   * Date : April 23, 2011
8:   *==========================
9:   */
10:  package classes.SE1D;
11:
12:  import classes.MathTools.Common.Function;
13:
14:  public class Quadratic  extends Function {
15:    public double Of(double x) {
16:    return (x*x)/2;
17:    }
18:    public  double DerivativeOf(double x) {  return 0.0; }
19:
20:
21:  }
22:
```

พลังงานของอนุภาคในบ่อศักย์ มีค่าไม่ต่อเนื่อง เมื่อ n = 1, 2, 3, ... หาได้จาก

$$E_{n-1}=\bar{h}\omega(n-\frac{1}{2}) \qquad (4.2)$$

ผลลัพธ์ที่ได้จากการจำลองอนุภาคในบ่อศักย์แบบฮาร์มอนิก ออสซิลเลเตอร์ ที่กำหนดให้ $\bar{h}=\omega=1$ จะได้พลังงานที่เป็นค่าเจาะจง ที่ n = 1, 2, ..., 5, 6 เปรียบเทียบกับค่าที่ได้จากการคำนวณตามสมการ (4.2) ดังตารางต่อไปนี้ พบว่าผลลัพธ์ที่ได้จากแบบจำลองมีค่าเท่ากับผลที่ได้จากการคำนวณโดยวิธีวิเคราะห์ทุกประการ

n (ระดับพลังงาน)	E จากวิธีวิเคราะห์	E จากแบบจำลอง ฯ
1	0.50	0.50
2	1.50	1.50
3	2.50	2.50
4	3.50	3.50
5	4.50	4.50
6	5.50	5.50

ตาราง 4.2 พลังงานของอนุภาค (E) ที่ n ค่าต่าง ๆ กันของบ่อศักย์แบบฮาร์มอนิก ออสซิลเลเตอร์

ผลลัพธ์จากการจำลองอนุภาคในบ่อศักย์แบบอื่น ๆ

ผลการคำนวณพลังงานของอนุภาคที่สถานะ n (ตั้งแต่ 1 ถึง 10) ของบ่อศักย์แต่ละแบบ โดยกำหนดค่าคงที่ของพลังงานศักย์ เพื่อให้การแสดงผลทางกราฟิกอยู่ในขอบเขตของจอภาพ ดังแสดงไว้ในตาราง ผลลัพธ์ในการคำนวณที่ได้จากโปรแกรม สามารถเขียนเป็นตารางได้ดังนี้

สถานะ n	Square well	Parabolic	Triangular	Square tangent	Square hyperbolic cosine
	$V=0 \quad \{\|x\|>2$ $V=-10 \quad \{-2\le x\le 2$	$V=\frac{1}{2}x^2$	$V=2.0\|x\|$	$V=\frac{1}{560}\tan^2 x$ $-\frac{\pi}{2}\le x\le\frac{\pi}{2}$	$V=\frac{-10}{\cosh^2 x}$
1	-9.752 280	+0.500 001	+1.283 745	+0.397 526	-7.999 999
2	-9.011 696	+1.500 001	+2.945 831	+1.578 424	-4.999 999
3	-7.786 852	+2.500 001	+4.092 519	+3.507 880	-2.000 000
4	-6.095 848	+3.500 002	+5.150 50	+6.107 141	-0.498 165
5	-3.977 063	+4.500 014	+6.073 163	+9.112 893	+0.136 219
6	-1.536 339	+5.500 102	+6.957 181	+10.363 892	+0.488 270
7	+0.369 385	+6.500 613	+7.775 388	+10.466 563	+0.997 425
8	+0.567 383	+7.502 947	+8.593 503	+11.360 108	+1.639 515
9	+1.154 297	+8.511 492	+9.418 333	+11.729 493	+2.402 942
10	+2.072 021	+9.536 639	+10.318 601	+12.688 721	+3.281 116

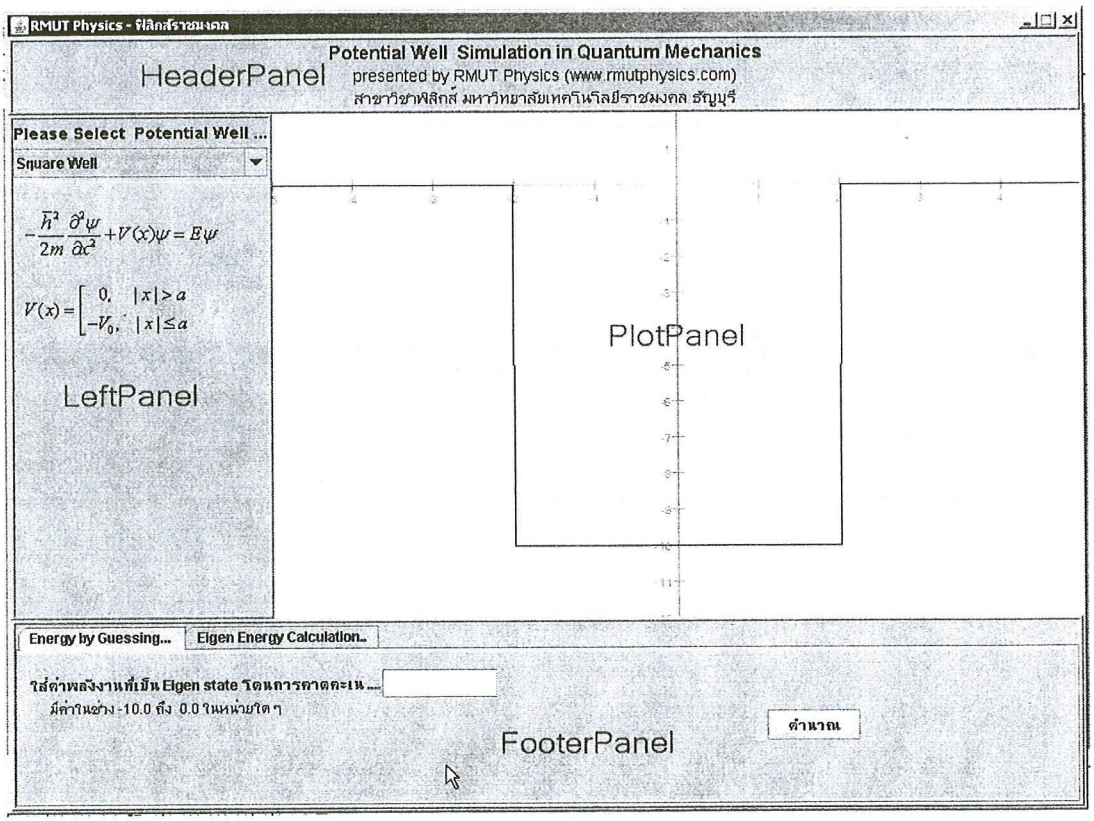
พลังงานที่ n ที่พิมพ์ไว้ด้วยตัวหนานั้น โปรแกรมจะแจ้งเตือนว่าระดับพลังงานที่ n ดังกล่าวไม่เป็น bound state

4.2 ผลลัพธ์การแสดงภาพกราฟิกของบ่อศักย์และฟังก์ชันคลื่นบนจอภาพ

การแสดงภาพกราฟิกในงานวิจัยนี้ ได้ใช้ Java Swing ซึ่งเป็นคลาสที่ติดตั้งพร้อมกับ JDK สามารถเรียกใช้งานคลาสต่าง ๆ เกี่ยวกับกราฟิก ได้ทันทีโดยไม่ต้องเขียนเมธอดขึ้นมาใหม่ แต่อาจต้องมีการ override เมธอดบางเมธอด เพื่อให้ได้ผลลัพธ์ตามจุดมุ่งหมายของงานวิจัย

Panel ต่าง ๆ ที่ใช้แสดงผลทั้งหมดสืบทอดมาจากคลาส JPanel โดยมี BasePanel เป็น panel หลักรองรับ panel อื่น ๆ ทั้งหมด การซ้อนทับของ panel มีลักษณะเป็นชั้น ๆ ประกอบด้วย HeaderPanel จะอยู่ด้านบนสุดใช้แสดงชื่อเรื่องและบ่อศักย์ที่กำลังเลือกศึกษา LeftPanel จะอยู่ด้านซ้ายมือของผู้ใช้ ส่วนนี้จะใช้แสดงตัวเลือกฟังก์ชันของบ่อศักย์ FooterPanel จะอยู่ด้านล่างสุดของจอภาพ เป็นส่วนที่แสดงการรับค่าจากคีย์บอร์ด และใช้เลือกสถานะ n ของพลังงาน PlotPanel จะเป็น panel ที่ใช้แสดงรูปกราฟของบ่อศักย์ และแสดงฟังก์ชันคลื่นที่สอดคล้องกับพลังงานที่ผู้ใช้เลือก

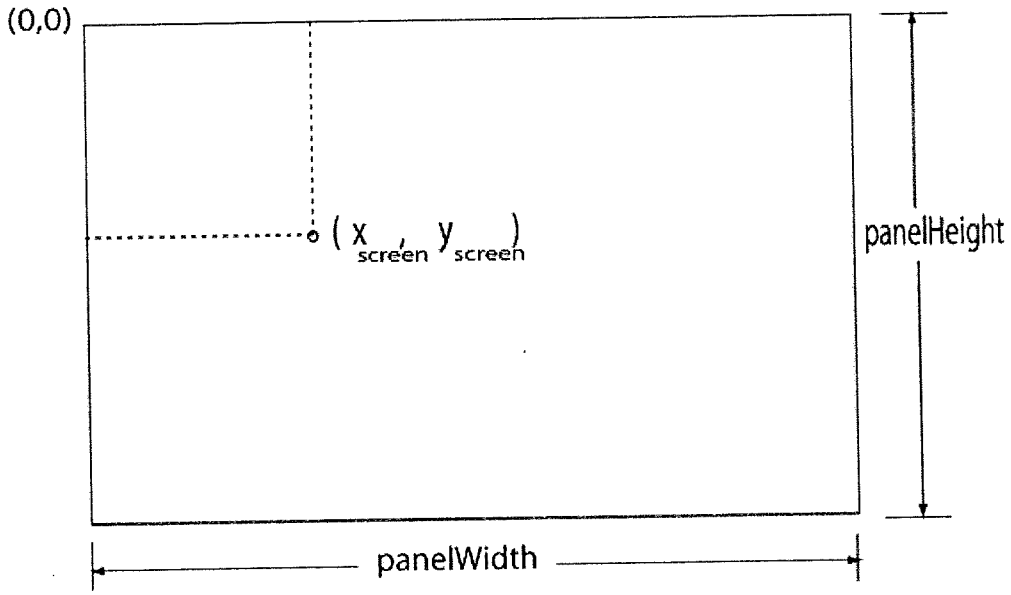
เมื่อแสดงผลบนจอภาพ จะเป็นดังรูปที่ 4.4



รูปที่ 4.4 เมื่อโปรแกรมทำงานจะได้ panel แต่ละ panel วางซ้อนบน BasePanel

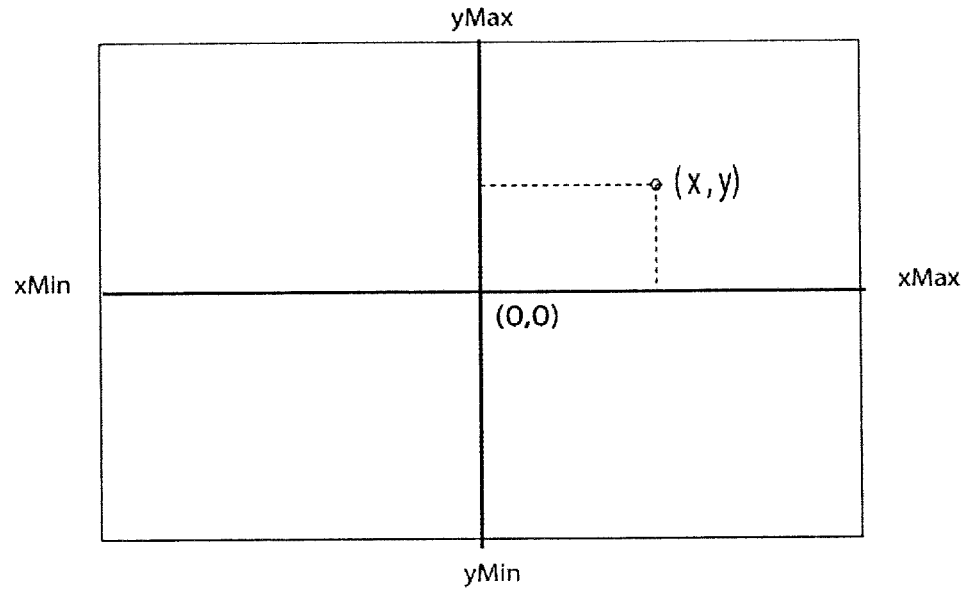
การแสดงภาพกราฟิกของบ่อศักย์และฟังก์ชันคลื่น จะแสดงอยู่บน Plot Panel ขนาดของพื้นที่ของPlotPanel จะเปลี่ยนแปลงไปตามขนาดของจอภาพที่ผู้ใช้ใช้งาน จุดโคออร์ดิเนตของระบบพิกัดฉากที่ใช้ในการแสดงผลบนจอภาพจะแตกต่างจากจุดโคออร์ดิเนตของระบบพิกัดฉากที่ใช้ในการพล็อตกราฟทางคณิตศาสตร์ ในการวาดรูปกราฟิกบน PlotPanel จึงต้องมีการแปลงจุดโคออร์ดิเนตจากผลการคำนวณทางคณิตศาสตร์ให้เป็นจุดโคออร์ดิเนตบนจอภาพเสียก่อนที่จะวาดภาพลงไป

จุดโคออร์ดิเนตของ PlotPanel บนจอภาพจะเริ่มต้น จุด (0,0) ที่มุมบนซ้าย x มีค่าเป็นบวกโดยนับไปทางด้านขวามือ y มีค่าเป็นบวกเมื่อนับลงไปทางด้านล่าง ดังรูป



รูปที่ 4.5 แสดงตำแหน่ง x, y บนจอภาพ

ในการพล็อตกราฟทางคณิตศาสตร์ จะตั้งแกน x และ y ตรงกึ่งกลางจอภาพ ค่า x มีค่าเป็นลบเมื่อนับมาทางด้านซ้าย และมีค่าเป็นบวก เมื่อนับไปทางด้านขวา ส่วนค่า y มีค่าเป็นบวกถ้านับในทิศทางพุ่งขึ้น และมีค่าเป็นลบเมื่อนับในทิศทางพุ่งลง การนำจุดโคออร์ดิเนตที่ได้จากการคำนวณมาพล็อตลงบนพื้นที่ PlotPanel จึงมิสามารถกระทำได้โดยตรง ต้องอาศัยการแปลงดังสมการต่อไปนี้



รูปที่ 4.6 แสดงจุดโคออร์ดิเนตทางคณิตศาสตร์ที่พล็อตลงในกระดาษกราฟ

เมื่อทราบตำแหน่ง (x, y) ใด ๆ สามารถแปลงเป็น ตำแหน่ง (x, y) บนพื้นที่ PlotPanel ของจอภาพ (ต่อไปจะเรียกเป็น x_{screen} และ y_{screen}) ดังนี้

$$x_{screen} = (x + xMin) \times \frac{panelWidth}{(xMax - xMin)}$$

$$y_{screen} = (yMax - y) \times \frac{panelHeight}{(yMax - yMin)}$$

ตำแหน่งที่จะวาดแกน x (xAxisRow) และ แกน y (yAxisCol) หาได้จาก

$$xAxisRow = round(\frac{yMax}{yMax - yMin}) \times panelHeight$$

$$yAxisCol = round(\frac{-xMin}{xMax - xMin}) \times panelWidth$$

การวาดแกน x และแกน y ใช้คำสั่งต่อไปนี้

แกน x: bg.drawLine(0, xAxisRow, panelWidth, xAxisRow);

แกน y: bg.drawLine(yAxisCol,0, yAxisCol, panelHeight);

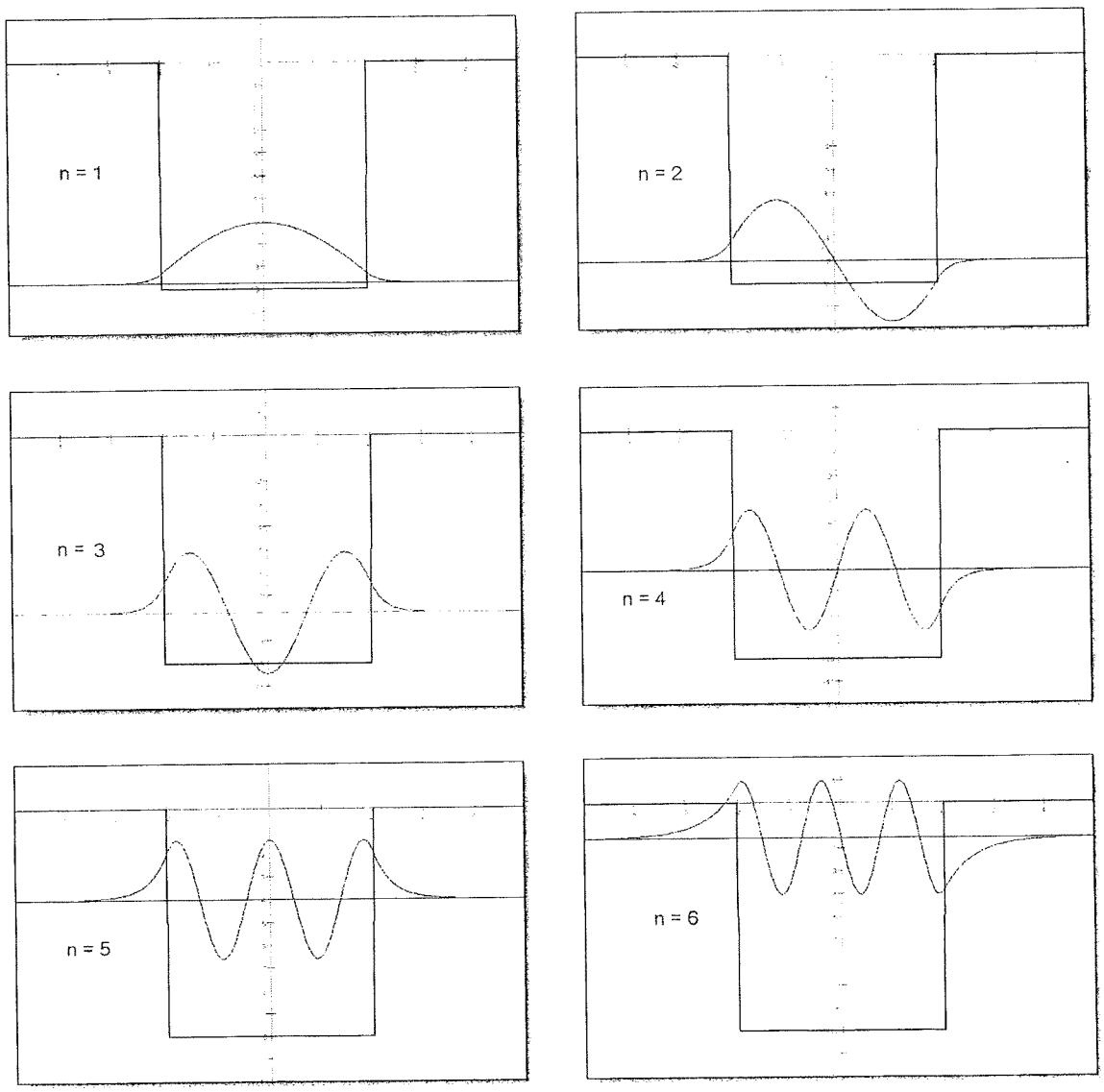
ในกรณีที่จะวาดฟังก์ชันคลื่น ให้แกน x สามารถเลื่อนขึ้นเลื่อนลงได้ตามระดับของพลังงาน สามารถทำได้โดยกำหนดค่า offset ของแกน y ดังนี้

$$y_{offset} = round(\frac{(yMax - y)}{yMax - yMin} \times panelHeight) - xAxisRow$$

รูปกราฟของบ่อศักย์และฟังก์ชันคลื่นของอนุภาคที่สถานะ n ต่าง ๆ โดยเลือกแสดงเพียงบางค่าดังนี้

บ่อศักย์แบบสี่เหลี่ยม มีความกว้าง 4 หน่วย มีความลึก –10 หน่วย เขียนเป็นสมการได้ดังนี้

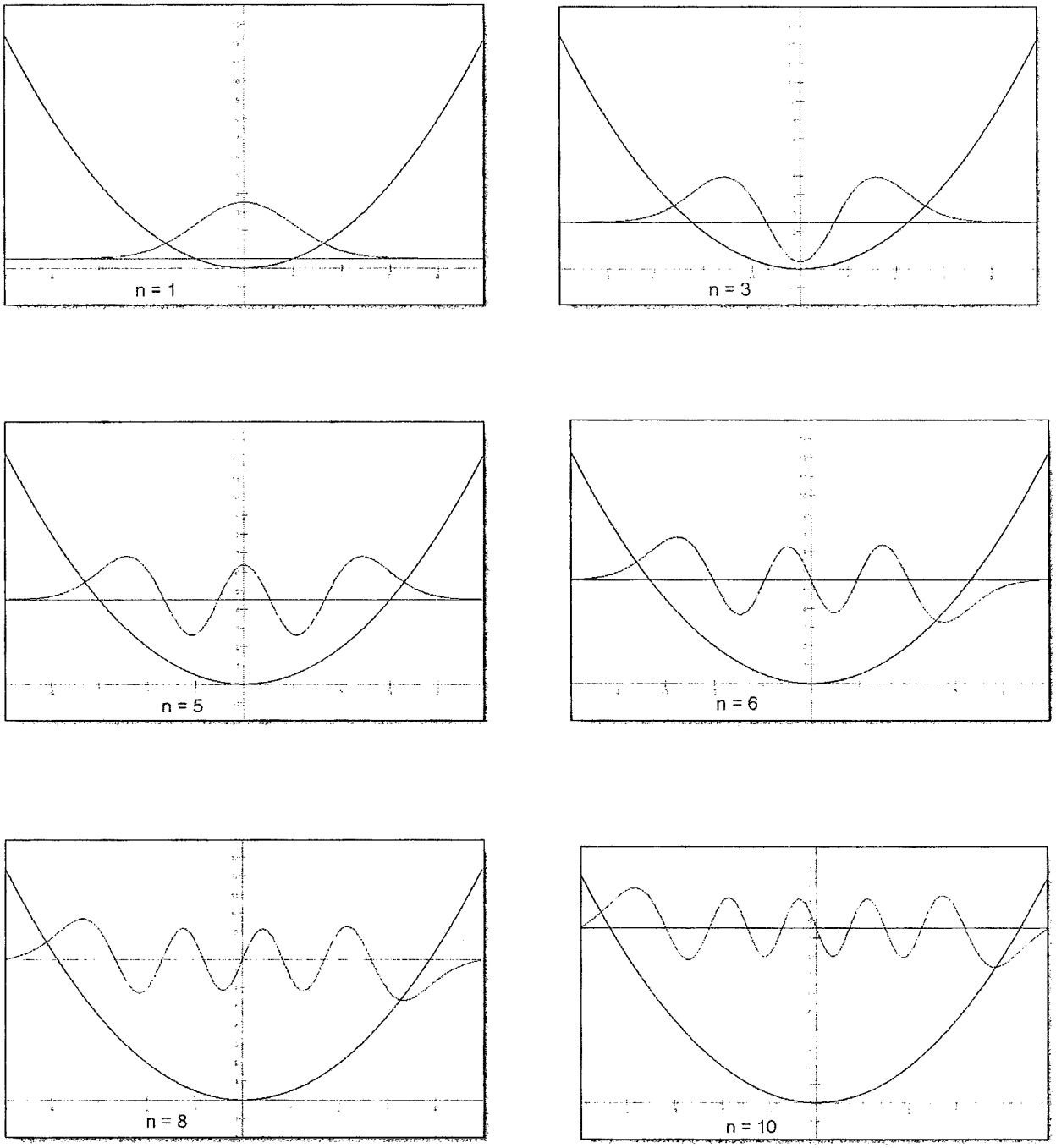
$$V = 0 \quad \{|x| > 2$$
$$V = -10 \quad \{-2 \le x \le 2$$



รูปที่ 4.7 ฟังก์ชันคลื่นในบ่อศักย์แบบสี่เหลี่ยมที่มีความลึกจำกัด

บ่อศักย์แบบพาราโบลา หรือ ฮาร์มอนิก ออสซิลเลเตอร์ มีรูปสมการดังนี้

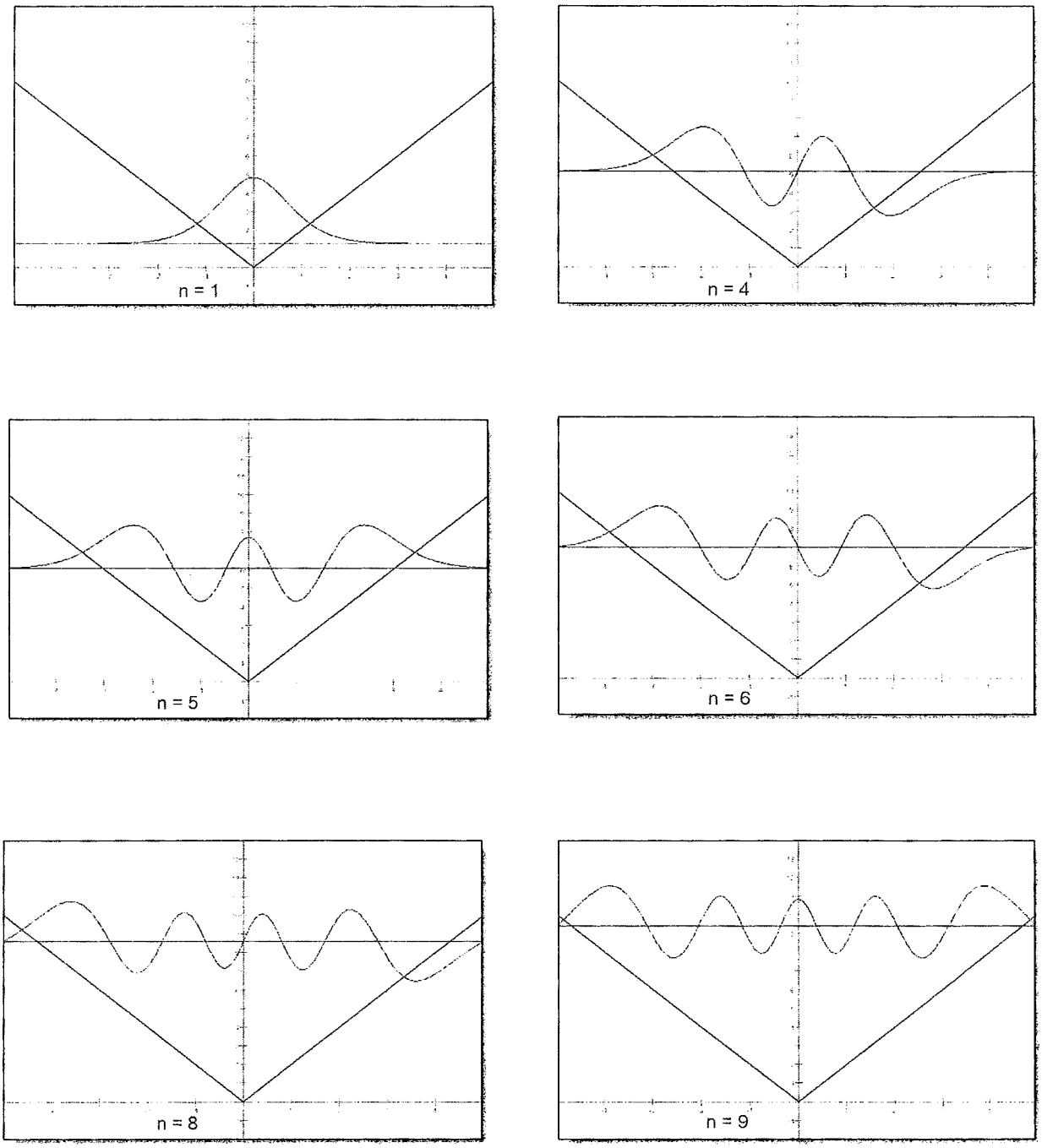
$$V = \frac{1}{2}x^2$$



รูปที่ 4.8 ฟังก์ชันคลื่นในบ่อศักย์แบบฮาร์มอนิก ออสซิลเลเตอร์ 1 มิติ

บ่อศักย์แบบสามเหลี่ยม Triangular well สมการของพลังงานศักย์เป็นดังนี้

$$V = 2.0|x|$$

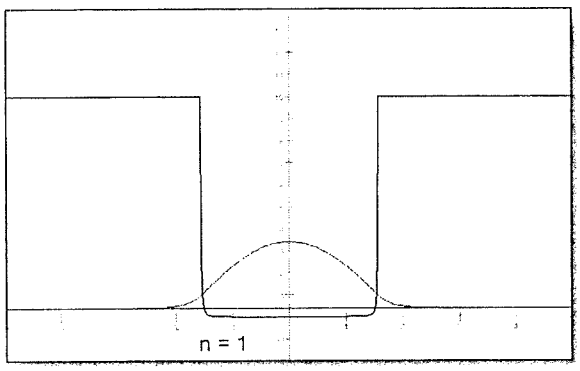


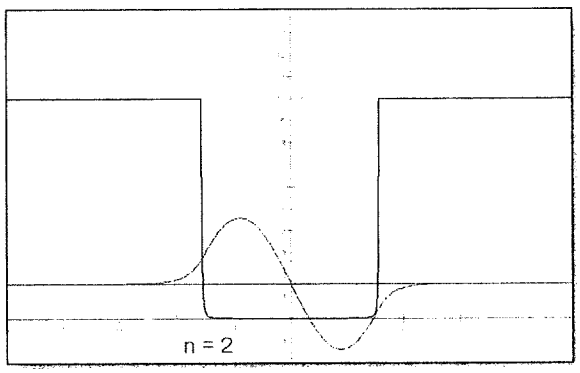
รูปที่ 4.9 ฟังก์ชันคลื่นในบ่อศักย์แบบสามเหลี่ยม

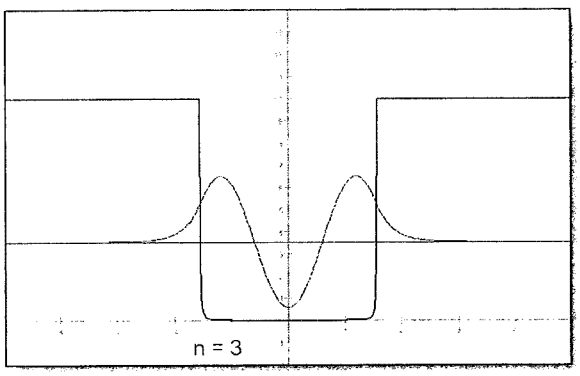
บ่อศักย์แบบ Square tangent well

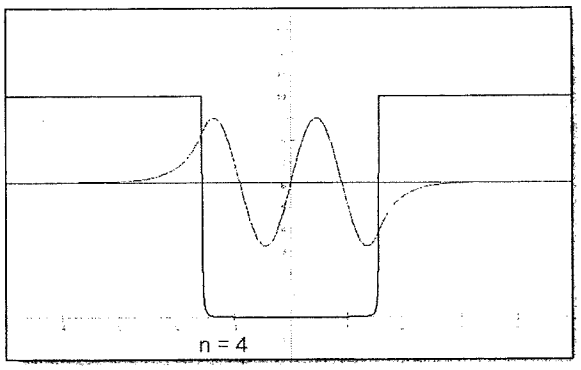
$$V = \frac{1}{560}\tan^2 x$$

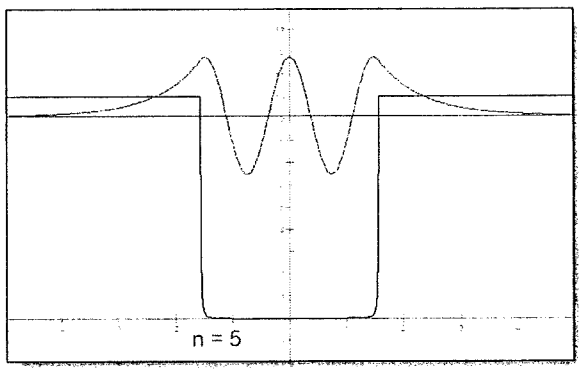
$$-\frac{\pi}{2} \le x \le \frac{\pi}{2}$$







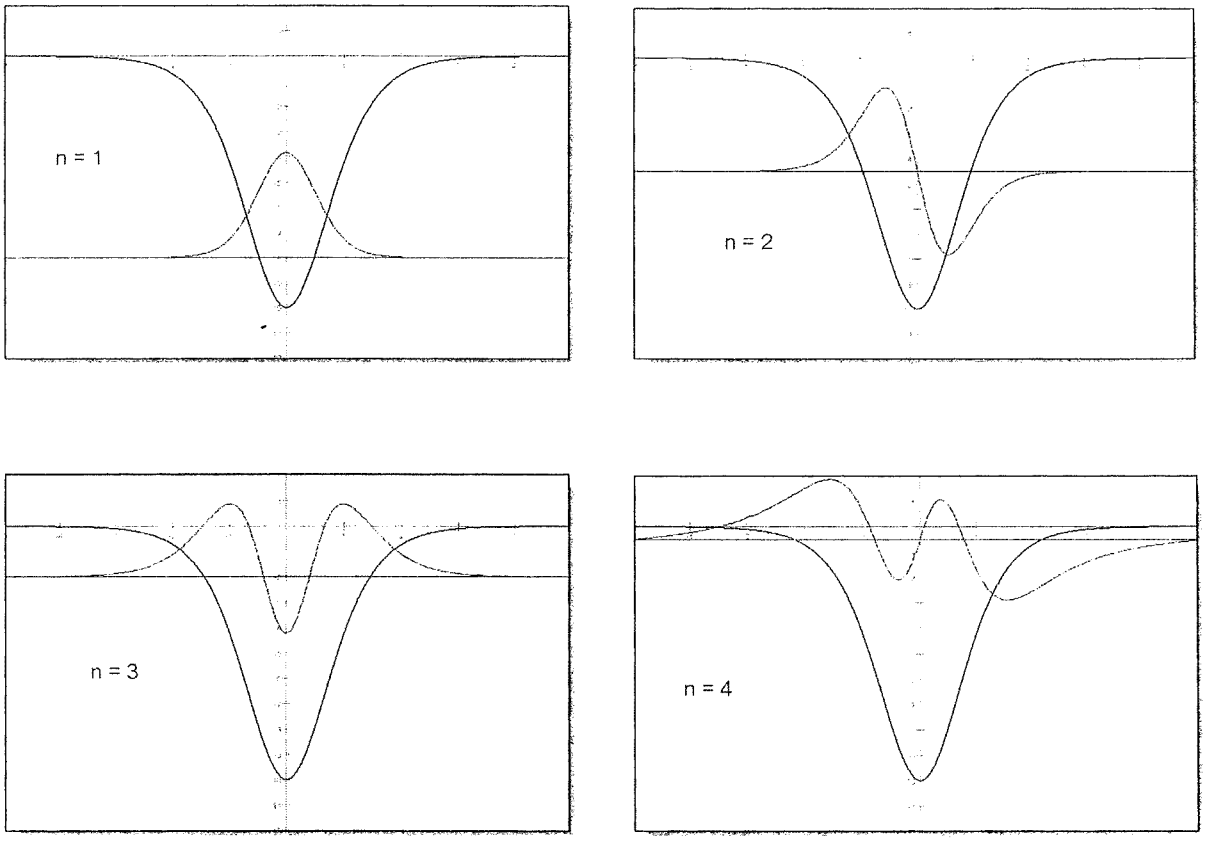




รูปที่ 4.10 ฟังก์ชันคลื่นในบ่อศักย์แบบ Square tangent

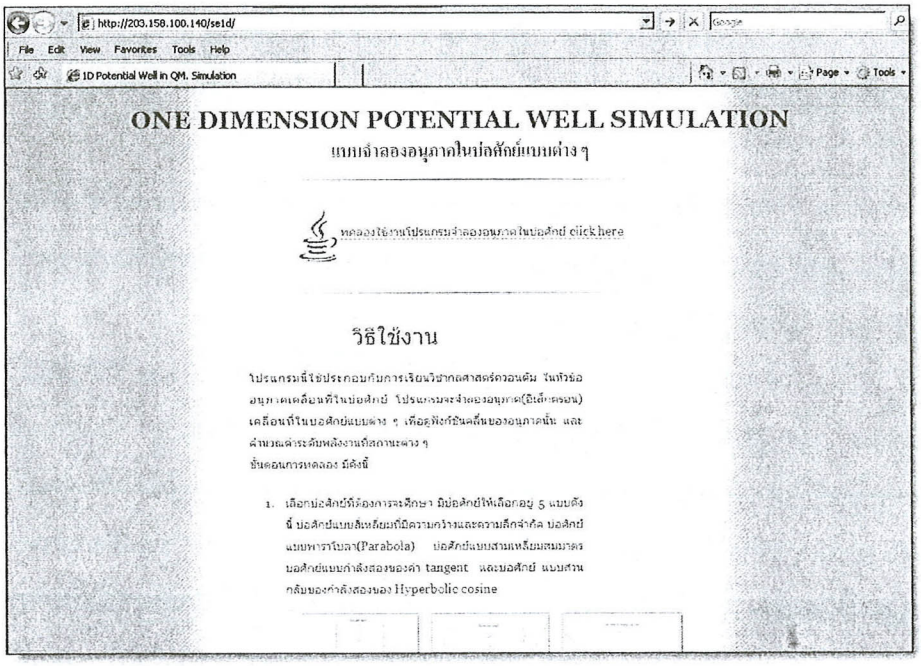
บ่อศักย์แบบ inverse square hyperbolic cosine

$$V = \frac{-10}{\cosh^2 x}$$



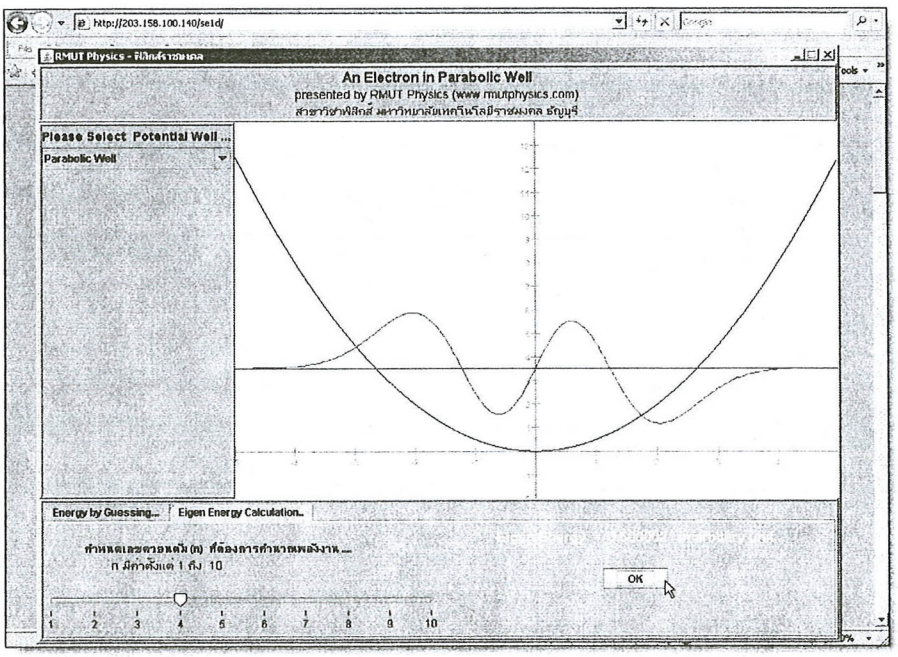
รูปที่ 4.11 ฟังก์ชันคลื่นในบ่อศักย์แบบสามเหลี่ยม

หลังจากทดสอบ และตรวจสอบความถูกต้องของโปรแกรมแล้ว ได้นำโปรแกรมไปติดตั้งในแม่ข่ายอินเตอร์เน็ต โดยติดตั้งไว้ที่ www.electron.rmutphysics.com/SE1D เมื่อเชื่อมต่ออินเตอร์เน็ตจะได้เว็บเพจ ดังภาพ



รูปที่ 4.12 เว็บเพจ ที่ใช้ติดตั้งโปรแกรมจำลองอนุภาคในบ่อศักย์แบบต่าง ๆ

เมื่อคลิกที่ข้อความให้โปรแกรมจำลองอนุภาคในบ่อศักย์แบบต่าง ๆ ทำงานจะได้หน้าตาของโปรแกรมดังรูปที่ 4.13



รูปที่ 4.13 เมื่อให้โปรแกรมทำงานผ่านเครือข่ายอินเตอร์เน็ต

4.3 โปรแกรมต้นฉบับ (Source code)

เมื่อนำการจำลองอนุภาคในบ่อศักย์แบบต่าง ๆ แสดงผลเป็นกราฟิก และสามารถทำงานผ่านเครือข่ายอินเตอร์เน็ต จะได้โปรแกรมต้นฉบับ(Source code) จำนวน 17 ไฟล์ แยกเป็นประเภทดังนี้

1. ส่วนที่ใช้ในการคำนวณ สมการชเรอดิงเงอร์ แบบ 1 มิติ ได้แก่ Schrodinger1D.java สามารถหาค่าพลังงานที่เป็นค่าจำเพาะเจาะจง และฟังก์ชันคลื่นของอนุภาค จัดเป็นหัวใจของโปรแกรมในงานวิจัยครั้งนี้

2. ส่วนที่เป็นฟังก์ชันของบ่อศักย์ ได้แก่ SquareWell.java (บ่อศักย์แบบสี่เหลี่ยมความลึกจำกัด) Parabolic.java (บ่อศักย์ของฮาร์มอนิก ออสซิลเลเตอร์) Triangular.java (บ่อศักย์แบบสามเหลี่ยมสมมาตร) TanSquare.java (บ่อศักย์แบบแทนเจนต์กำลังสอง) และ InveseCosh.java (บ่อศักย์แบบไฮเปอบอลิก คอซกำลังสอง) เมื่อผู้ใช้เลือกศึกษาบ่อศักย์แบบใด ฟังก์ชันของบ่อศักย์เหล่านั้นจะถูกเรียกใช้โดย Schrodinger1D.java และนำฟังก์ชันทางคณิตศาสตร์ไปพล็อตบนจอภาพ

3. ส่วนที่เป็นcontainer หลักของโปรแกรม โดยใช้ SWING GUI และส่วนที่เป็น Input ของโปรแกรม ได้แก่ BasePanel.java เป็นกรอบหน้าต่างหลักที่บรรจุหน้าต่างอื่น ๆ ไว้ทั้งหมด LeftPanel.java ใช้ในการแสดงผล ComboBox สำหรับเลือกบ่อศักย์ InputPanel.java EnergyPanel.java QuantumStatePanel.java ใช้สำหรับป้อนข้อมูลเกี่ยวกับพลังงานและ สถานะควอนตัม ทั้งสามโปรแกรมนี้ถูกรวมไว้ใน FooterPanel.java และ HeaderPanel.java ใช้แสดงหัวเรื่องซึ่งจะเปลี่ยนไปตามชนิดบ่อศักย์ที่เลือก

4. ส่วนที่แสดงผลกราฟิก มี 2 ไฟล์ คือ PlotPanel.java จะเป็นหน้าต่างหลักในการแสดงภาพกราฟิกทั้งหมด ตั้งแต่เขียนรูปบ่อศักย์เป็นฉากหลัง เขียนแกน x และ y รวมทั้งมาตราส่วนของกราฟ เขียนรูปฟังก์ชันคลื่นเมื่อค่าพลังงานเปลี่ยนไป รวมทั้งการ update หน้าจอ เมื่อมีการเปลี่ยนแปลงชนิดบ่อศักย์ โดยมี SEPanel.java เป็นตัวกำหนดพารามิเตอร์ต่าง ๆ

5. ส่วนที่เป็นทางเข้าของโปรแกรม หรือเป็น main program คือ mainApp.java โปรแกรมจะเริ่มต้นทำงานที่โปรแกรมนี้

รายละเอียดของโปรแกรมทั้ง 17 ไฟล์ได้รวบรวมไว้ที่เดียวกันในภาคผนวก 4 โปรแกรมอาจมีการแก้ไข ปรับปรุงเล็ก ๆ น้อย ๆ จนกระทั่งนาทีที่ปิดเล่มรายงานการวิจัยฉบับนี้ ต้องการต้นฉบับโปรแกรมที่มีการ update ล่าสุด สามารถ download ได้ที่ www.electron.rmutphysics.com/se1d