```
// A program to solve the problem of a person sitting
// on a bench as described in the text.

import java.lang.*;
public class Bench {
  final static int n = 99, m = 2;
  public static void main(String argv[]) {
  double d[] = new double[n];
  double b[] = new double[n];
  double c[] = new double[n];
  double l = 3, l2 = l/2, h = l/(n+1), h2 = h*h;
  double x0 = 0.25, x2 = x0*x0, e0 = 1/Math.E;
  double rho = 3, g = 9.8, f0 = 200;
  double y = 1e9*Math.pow(0.03,3)*0.2/3;

 // Evaluate the coefficient matrix elements
    for (int i=0; i<n; ++i) {
      d[i] = -2;
      c[i] =  1;
      b[i] = -rho*g;
      double x = h*(i+1)-l2;
      if (Math.abs(x) < x0)
        b[i] -= f0*(Math.exp(-x*x/x2)-e0);
      b[i] *= h2/y;
    }

 // Obtain the solution of the curverture of the bench
    double u[] = tridiagonalLinearEq(d, c, c, b);

 // Output the result in every m time steps
    double x = h;
    double mh = m*h;
    for (int i=0; i<n; i+=m) {
      System.out.println(x + " " + 100*u[i]);
      x += mh;
    }
  }

// Method to solve the tridiagonal linear equation set.

  public static double[] tridiagonalLinearEq(double d[],
    double e[], double c[], double b[]) {...}
}
```

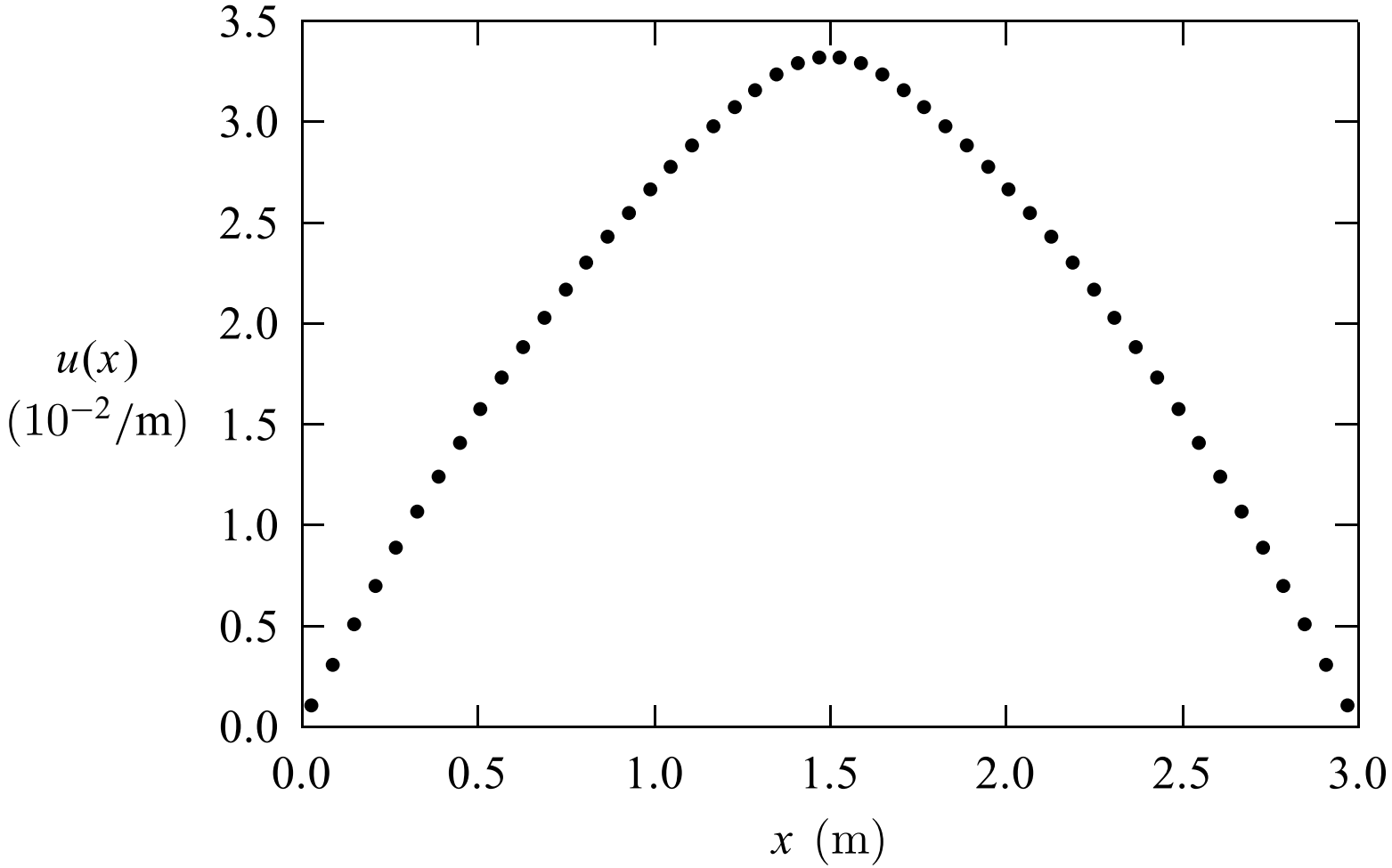
l=length of bench

Usual spatial step size

Bench parameters

Matrix elements for A in A*x=b formulation

Solve discretized/Matrix Form of Poisson Eqt.


3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
$u(x)$
$(10^{-2}/\mathrm{m})$
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x (m)

```
// A program to solve the problem of a person sitting
// on a bench with the relaxation scheme.

import java.lang.*;
public class Bench2 {
  final static int n = 100, m = 2;
  public static void main(String argv[]) {
  double u[] = new double[n+1];
  double d[] = new double[n+1];
  double s[] = new double[n+1];
  double l = 3, l2 = l/2, h = l/n, h2 = h*h;
  double x0 = 0.25, x2 = x0*x0, e0 = 1/Math.E;
  double x = 0, rho = 3, g = 9.8, f0 = 200;
  double y = 1e9*Math.pow(0.03,3)*0.2/3;
  double u0 = 0.032, p = 1.5, del =1e-3;
  int nmax = 100;

 // Evaluate the source in the equation
    for (int i=0; i<=n; ++i) {
      s[i] = rho*g;
      x = h*i-l2;
      if (Math.abs(x) < x0)
        s[i] += f0*(Math.exp(-x*x/x2)-e0);
      s[i] *= h2/y;
    }
    for (int i=1; i<n; ++i) {
      x = Math.PI*h*i/l;
      u[i] = u0*Math.sin(x);
      d[i] = 1;
    }
    d[0] = d[n] = 1;
    relax(u, d, s, p, del, nmax);

 // Output the result in every m time step
    x = 0;
    double mh = m*h;
    for (int i=0; i<n; i+=m) {
      System.out.println(x + " " + 100*u[i]);
      x += mh;
    }
```

del=global convergence tolerance

Relaxation mixing fraction p

Input Guess for Relaxation Method

Solve using Relaxation Method

```
// Method to complete one step of relaxation.
  public static void relax(double u[], double d[],
    double s[], double p, double del, int nmax) {
    int n = u.length-1;
    double q = 1-p, fi = 0;
    double du = 2*del;
    int k = 0;

    while ((du>del) && (k<nmax)) {
      du = 0;
      for (int i=1; i<n; ++i) {
        fi = u[i];
        u[i] = p*u[i]
              +q*((d[i+1]+d[i])*u[i+1]
              +(d[i]+d[i-1])*u[i-1]+2*s[i])/(4*d[i]);
        fi = u[i]-fi;
        du += fi*fi;
      }
      du = Math.sqrt(du/n);
      k++;
    }
    if (k==nmax) System.out.println("Convergence not" +
      " found after " + nmax + " iterations");
  }
}
```

Loop until global convergence tolerance met

Add old guess with weight p

Update with discretized version of Poisson equation with weight q=1-p

```
// An example of studying the 2-dimensional groundwater
// dynamics through the relaxation method.

import java.lang.*;
public class Groundwater {
  final static int nx = 100, ny = 50, ni = 5;
  public static void main(String argv[]) {
    double sigma0 = 1, a = -0.04, phi0 = 200, b = -20;
    double lx = 1000, hx = lx/nx, ly = 500, hy =ly/ny;
    double phi[][] = new double[nx+1][ny+1];
    double sigma[][] = new double[nx+1][ny+1];
    double f[][] = new double[nx+1][ny+1];
    double p = 0.5;

 // Set up boundary values and a trial solution
    for (int i=0; i<=nx; ++i) {
      double x = i*hx;
      for (int j=0; j<=ny; ++j) {
        double y = j*hy;
        sigma[i][j]  = sigma0+a*y;
        phi[i][j] = phi0+b*Math.cos(Math.PI*x/lx)*y/ly;
        f[i][j] = 0;
      }
    }
    for (int step=0; step<ni; ++step) {

   // Ensure boundary conditions by 4-point formula
      for (int j=0; j<ny; ++j) {
         phi[0][j] = (4*phi[1][j]-phi[2][j])/3;
         phi[nx][j] = (4*phi[nx-1][j]-phi[nx-2][j])/3;
      }
      relax2d(p, hx, hy, phi, sigma, f);
    }

 // Output the result
    for (int i=0; i<=nx; ++i) {
      double x = i*hx;
      for (int j=0; j<=ny; ++j) {
        double y = j*hy;
        System.out.println(x + " " + y + " "
          + phi[i][j]);
      }
    }
  }
```

Asymmetry-Set to identity matrix (ignore this)

Initial Guess

Non-trivial accommodation of boundaries

```
// Method to perform a relaxation step in 2D.

  public static void relax2d(double p, double hx,
   double hy, double u[][], double d[][],
   double s[][]) {
   double h2 = hx*hx, a = h2/(hy*hy),
     b = 1/(4*(1+a)), ab = a*b, q = 1-p;
   for (int i=1; i<nx; ++i) {
     for (int j = 1; j <ny; ++j) {
       double xp = b*(d[i+1][j]/d[i][j]+1);
       double xm = b*(d[i-1][j]/d[i][j]+1);
       double yp = ab*(d[i][j+1]/d[i][j]+1);
       double ym = ab*(d[i][j-1]/d[i][j]+1);
       u[i][j] = q*u[i][j]+p*(xp*u[i+1][j]
                 +xm*u[i-1][j]+yp*u[i][j+1]
                 +ym*u[i][j-1]+h2*s[i][j]);
     }
   }
  }
}
```

"a" is "alpha" for square of ratio of x-to-y grid spacing

Discretize in x and y

q terms adds old solution with weight q

p terms adds update with weight p