2011 BIOSTAT 615/815 Homework #6

Due is Thursday April 7th, 08:30AM.

Problem 1. Evaluation of Simplex Method

Consider the vector norm function:

$$f(\mathbf{x}) = |\mathbf{x}| = \sqrt{\sum_{i=1}^{k} x_i^2} \tag{1}$$

The function has minimum 0 when $x = 0$.

We will use this function to examine the efficacy of the Nelder-Mead algorithm for function minimization. For $k = 1$ to 20 dimensions, generate random starting vectors of k random elements, where each element varies between value between -100 and 100. With these vectors as starting points, use the Nelder-Mead algorithm to find the minimum of the vector norm function above (use 10^{-7}) as the relative accuracy threshold). Then graph:

1. The total number of function evaluations carried out for each k.
2. The norm evaluated at the minimum point identified by the Nelder-Mead routine.
3. Repeat part 1. using the vector 0 (in k dimensions) as the starting point for the minimization routine.

When generating starting point, use current timestamp to randomize your outcomes.

Print the hard copy of your source code and the results, and E-mail your source code to the instructor too.

You may (but don't have to) start from the source code given in the lecture. The source code is also available in the class web page

```
#ifndef __SIMPLEX_615_H
#define __SIMPLEX_615_H

#include <vector>
#include <cmath>
#include <iostream>

#define ZEPS 1e-10

class optFunc {
 public:
  virtual double operator() (std::vector<double>& x) = 0;
};

// Simplex contains (dim+1)*dim points
class simplex615 {
 protected:
  std::vector<std::vector<double> > X;
  std::vector<double> Y;
  std::vector<double> midPoint;
  std::vector<double> thruLine;

  int dim, idxLo, idxHi, idxNextHi;

  void evaluateFunction(optFunc& foo);
  void evaluateExtremes();
  void prepareUpdate();
  bool updateSimplex(optFunc& foo, double scale);
  void contractSimplex(optFunc& foo);
  static int check_tol(double fmax, double fmin, double ftol);
 public:
  simplex615(double* p, int d);
  void amoeba(optFunc& foo, double tol);
  std::vector<double>& xmin();
  double ymin();
};

simplex615::simplex615(double* p, int d) : dim(d) {
  X.resize(dim+1);
  Y.resize(dim+1);
  midPoint.resize(dim);
  thruLine.resize(dim);
  for(int i=0; i < dim+1; ++i) {
    X[i].resize(dim);
  }

  // set every point the same
  for(int i=0; i < dim+1; ++i) {
    for(int j=0; j < dim; ++j) {
      X[i][j] = p[j];
    }
  }

  // then increase each dimension by one except for the last point
  for(int i=0; i < dim; ++i) {
    X[i][i] += 1.;
  }
}

void simplex615::evaluateFunction(optFunc& foo) {
  for(int i=0; i < dim+1; ++i) {
    Y[i] = foo(X[i]);
  }
}

void simplex615::evaluateExtremes() {
  if ( Y[0] > Y[1] ) {
    idxHi = 0;
    idxLo = idxNextHi = 1;
  }
  else {
    idxHi = 1;
    idxLo = idxNextHi = 0;
  }

  for(int i=2; i < dim+1; ++i) {
    if ( Y[i] <= Y[idxLo] ) {
      idxLo = i;
    }
    else if ( Y[i] > Y[idxHi] ) {
      idxNextHi = idxHi;
      idxHi = i;
    }
    else if ( Y[i] > Y[idxNextHi] ) {
      idxNextHi = i;
    }
  }
}

void simplex615::prepareUpdate() {
  for(int j=0; j < dim; ++j) {
    midPoint[j] = 0;
  }
  for(int i=0; i < dim+1; ++i) {
    if ( i != idxHi ) {
      for(int j=0; j < dim; ++j) {
   midPoint[j] += X[i][j];
      }
    }
  }
  for(int j=0; j < dim; ++j) {
    midPoint[j] /= dim;
    thruLine[j] = X[idxHi][j] - midPoint[j];
  }
}

bool simplex615::updateSimplex(optFunc& foo, double scale) {
  std::vector<double> nextPoint;
  nextPoint.resize(dim);
  for(int i=0; i < dim; ++i) {
    nextPoint[i] = midPoint[i] + scale * thruLine[i];
  }
  double fNext = foo(nextPoint);
  if ( fNext < Y[idxHi] ) { // exchange with maximum
    for(int i=0; i < dim; ++i) {
      X[idxHi][i] = nextPoint[i];
    }
    Y[idxHi] = fNext;
    return true;
  }
  else {
    return false;
  }
}

void simplex615::contractSimplex(optFunc& foo) {
  for(int i=0; i < dim+1; ++i) {
    if ( i != idxLo ) {
      for(int j=0; j < dim; ++j) {
   X[i][j] = 0.5*( X[idxLo][j] + X[i][j] );
   Y[i] = foo(X[i]);
      }
    }
  }
}

void simplex615::amoeba(optFunc& foo, double tol) {
  evaluateFunction(foo);
  while(true) {
    evaluateExtremes();
    prepareUpdate();

    if ( check_tol(Y[idxHi],Y[idxLo],tol) ) break;

    updateSimplex(foo, -1.0); // reflection

    if ( Y[idxHi] < Y[idxLo] ) {
      updateSimplex(foo, -2.0); // expansion
    }
    else if ( Y[idxHi] >= Y[idxNextHi] ) {
      if ( !updateSimplex(foo, 0.5) ) {
   contractSimplex(foo);
      }
    }
  }
}

std::vector<double>& simplex615::xmin() {
  return X[idxLo];
}

double simplex615::ymin() {
  return Y[idxLo];
}

int simplex615::check_tol(double fmax, double fmin, double ftol) {
  double delta = fabs(fmax - fmin); double accuracy = (fabs(fmax) + fabs(fmin)) * ftol;
  return (delta < (accuracy + ZEPS));
}

#endif // __SIMPLEX_615_H
```