```cpp
#include <iostream>
#include <cmath>
#include "ddouble.hpp"
#include "conjgrad.hpp"

using namespace std;

ddouble Minproblem::E(){
    ddouble E1=0;
    for(int i=0;i<nz;i++)
        E1 += (z[i]-i)*(z[i]-i) ;// +exp(z[i]);
    return E1;
}

void Minproblem::dE(){
    for(int i=0;i<nz;i++){
        z[i].val[1]=1;
        grad[i] = E().val[1];
        z[i].val[1]=0;
    }

}

int main () {

    Minproblem pb(10);
    for(int i=0;i<pb.nz;i++) pb.z[i]=0;
    pb.descent(10);
    for(int i=0;i<pb.nz;i++) cout<<pb.z[i]<<endl;
    return 0;
}
```

```
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <fstream>

#include "ddouble.hpp"

using namespace std;

 class Minproblem
  {
    public:
           int nz;      // nb of unknowns
           ddouble* z;   // array of unknowns
           double* grad;// array of partial derivatives
           double* h;   // direction of descent
           double rho;  // step size


    ddouble E();             // cost function
    void   dE();        // fills partial derivatives in grad
      Minproblem ( const int nz1 ) : nz(nz1){
             grad = new double[nz];  h = new double[nz]; z = new ddouble
                 [nz];
      }
    void   descent(int pmax, double rho1=1, double eps=1.e-5);
    double e(double rho, double E0, double* h);
    double armijorule(double rho,double dE0, double alpha, double E0,
        double* h);
  };
////////////////////// implementation  ////////////////////////////////

double Minproblem::armijorule(double rho,double dE0, double alpha, double
    E0, double* h)
{
  int i = 0;
   if(e(rho, E0, h) < alpha * rho * dE0)
  {
    do
         rho = 2*rho;
    while( e(rho, E0, h) < alpha * rho * dE0 && i++ < 100);
    return rho / 2;
  } else {
    do
         rho = rho / 2;
    while( e(rho, E0, h) > alpha * rho * dE0 && i++ < 100);
    return rho;
  }
}


double Minproblem::e(double rho, double E0, double* h)
{
    for(int i = 0; i < nz; i++)
       z[i] += rho*h[i];
    double aux = E().val[0] - E0;
    for(int i = 0; i < nz; i++)
```

```
        z[i] -= rho*h[i];
    return aux;
}

void Minproblem::descent(int pmax, double rho1, double eps)
{
    rho=rho1;
    double normg2old = 1e60;
  for(int i =0; i<nz; i++) h[i]=0;

  for(int p = 0; p < pmax; p++)
   {
      double dE2=0, normg2=0, E0 = E().val[0];
      dE();

     for(int i =0; i<nz; i++) normg2 += grad[i]*grad[i];

     double gam = normg2/normg2old;

     for(int i =0; i<nz; i++) h[i] = -grad[i] + gam*h[i];

     normg2old = normg2;

     for(int i =0; i<nz; i++)  dE2 += grad[i]*h[i];

     double rhom =  armijorule(rho,dE2,0.45,E0,h);

     if(rhom<0) {cout<<"rho<0:\n"; rhom = -rho / 1000;}
     else if(rhom<1e-20) {cout<< "rho<<1:\n"; rhom=1e-20;}
     else if(rhom>1e10) cout<< "rho>>1:\n";

     for(int i =0; i<nz; i++)z[i] += rhom*h[i];

     cout<<p<<'\t'<<E0<<' '<<normg2<<'\t'<<rhom<<'\t'<<gam<<endl;
     double E1 = E().val[0];
     if(E1>E0) cout<<E1<<" cost grows, wrong gradient"<<endl;
     if(normg2<eps*eps) cout<< "optimization done\n";
     if(normg2<eps*eps) break;
     if(E1>E0 || normg2<eps*eps) break;
   }
}
```

```
// file ddouble.hpp, for automatic differentiation (single variable)
// adapted from M. Grundmann's MIPAD

#ifndef _DDOUBLE__H_
#define _DDOUBLE__H_

#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <fstream>
using namespace std;

static const double sqrtpi = sqrt(4*atan(1.0));

class ddouble
{
 public:  double val[2]; // val[0] is value and val[1] is the derivative
 ddouble() { val[0] = 0; val[1] = 0;}
 ddouble(const ddouble& a){ val[0] = a.val[0]; val[1] = a.val[1]; }
 ddouble(double a, double b=0){ val[0]=a; val[1]=b;} // specifies var for
     diff

 ddouble& operator=(double a)
    { val[0] = a; val[1] = 0.0; return *this;}
 ddouble& operator=(const ddouble& a)
    { val[0] = a.val[0]; val[1] = a.val[1]; return *this;}

 double& operator[](const int ii){ return this->val[ii];}
 double operator[](const int ii) const { return this->val[ii];}

  ddouble& operator  + (){return *this;};
  ddouble& operator += (double);
  ddouble& operator += (const ddouble&);
  ddouble& operator -= (double );
  ddouble& operator -= (const ddouble&);
  ddouble& operator *= (double);
  ddouble& operator *= (const ddouble&);
  ddouble& operator /= (double) ;
  ddouble& operator /= (const ddouble&) ;

  ddouble operator++(int);
  ddouble operator--(int);
  ddouble&  operator++();
  ddouble&  operator--();

  friend ostream& operator << (ostream&, const ddouble&);
  friend ddouble& operator << (ddouble&,double);
  friend ddouble parameter(double);

  friend int operator != (const ddouble&,const ddouble&);
  friend int operator != (double,const ddouble&);
  friend int operator != (const ddouble&,double);
  friend int operator == (const ddouble&,const ddouble&);
  friend int operator == (double,const ddouble&);
  friend int operator == (const ddouble&,double);
  friend int operator >= (const ddouble&,const ddouble&);
  friend int operator >= (double,const ddouble&);
```

```
  friend int operator >= (const ddouble&,double);
  friend int operator <= (const ddouble&,const ddouble&);
  friend int operator <= (double,const ddouble&);
  friend int operator <= (const ddouble&,double);
  friend int operator > (const ddouble&,const ddouble&);
  friend int operator > (double,const ddouble&);
  friend int operator > (const ddouble&,double);
  friend int operator < (const ddouble&,const ddouble&);
  friend int operator < (double,const ddouble&);
  friend int operator < (const ddouble&,double);


  friend ddouble operator + (const ddouble& x); //{return x + 0.0 ;} ;
  friend ddouble operator + (const ddouble&,const ddouble&);
  friend ddouble operator + (double, const ddouble&);
  friend ddouble operator + (const ddouble&, double);
  friend ddouble operator - (const ddouble& x ,double y); //{return (-y)+
      x;};
  friend ddouble operator - (const ddouble&,const ddouble&);
  friend ddouble operator - (double, const ddouble&);
  friend ddouble operator - ( const ddouble& );
  friend ddouble operator * (const ddouble&,const ddouble&);
  friend ddouble operator * (double, const ddouble& );
  friend ddouble operator * (const ddouble& x, double y);
  friend ddouble operator / (const ddouble& x, double y);
  friend ddouble operator / (const ddouble&,const ddouble&);
  friend ddouble operator / (double,const ddouble&) ;
  friend ddouble exp (const ddouble&);
  friend ddouble log (const ddouble&) ;
  friend ddouble sqrt (const ddouble&) ;
  friend ddouble sin (const ddouble&);
  friend ddouble cos (const ddouble&);
  friend ddouble tan (const ddouble&);
  friend ddouble pow (const ddouble&,double);
  friend ddouble abs (const ddouble&) ;
  friend ddouble erfc (const ddouble&) ;
};

inline double sign(const ddouble& x)
{ return ( x < 0.0 ? -1.0 : 1.0); }

inline double sign(const ddouble& x, double y)
{ return ( x < 0.0 ? -fabs(y) : fabs(y)); }

// f2c functions
inline ddouble d_abs(ddouble * x){ return abs(*x);  }
inline ddouble d_cos(ddouble * x){ return cos(*x);  }
inline ddouble d_sin(ddouble * x){ return sin(*x);  }
inline ddouble d_tan(ddouble * x){ return tan(*x);  }
inline ddouble d_exp(ddouble * x){ return exp(*x);  }
inline ddouble d_log(ddouble * x){ return log(*x);  }
inline ddouble d_sign(ddouble * x){ return sign(*x);    }
inline ddouble d_sign(ddouble * x,double*y){ return sign(*x,*y);    }
inline ddouble d_sqrt(ddouble * x){ return sqrt(*x);    }

////////////////// Implementation Part //////////////////////////
const double eps = 1.0e-50; // to avoid NaN when y=0 in sqrt(y)
```

```
ostream& operator<<(ostream& f, const ddouble& a)
{ f << "[" << a[0] << ','<< a[1] << "]"; return f;}

ddouble ddouble::operator++(int)
{   ddouble r=(*this);  r[0]++; return r;}

ddouble ddouble::operator--(int)
{   ddouble r=(*this);  r[0]--; return r;}

ddouble& ddouble::operator++()
{  (*this)[0]++; return *this;}

ddouble& ddouble::operator--()
{  (*this)[0]--; return *this;}

ddouble& ddouble::operator += (double y)
{  (*this)[0] += y; return *this; }


ddouble operator - (const ddouble& a)
{ ddouble r; r[0] = -a[0]; r[1] = -a[1];  return r;}

ddouble& ddouble::operator -= (double y)
{  (*this)[0]-=y; return *this;}

ddouble& ddouble::operator += (const ddouble& y)
{   (*this)[0]+=y[0];(*this)[1]+=y[1];  return *this; }

ddouble& ddouble::operator -= (const ddouble& y)
{   (*this)[0]-=y[0];(*this)[1]-=y[1];  return *this; }

ddouble& ddouble::operator *= (double y)
{ (*this)[0] *=y; (*this)[1] *=y;  return *this;}

ddouble& ddouble::operator *= (const ddouble& y)
{ return *this = *this * y;}

ddouble& ddouble::operator /= (const ddouble& y)
{  return *this = *this / y;}

ddouble& ddouble::operator /= (double y)
{ const double inv = 1.0 / y;
    (*this)[1]  *= inv; (*this)[1]  *= inv;
return *this;
}

int operator != (const ddouble& u,const ddouble& v)
{  return u[0] != v[0];}

int operator != (double u,const ddouble& v)
{  return u != v[0];}

int operator != (const ddouble& v,double u)
{  return v[0] != u;}

int operator == (const ddouble& u,const ddouble& v)
```

```
{  return u[0] == v[0];}

int operator == (double u,const ddouble& v)
{  return u == v[0];}

int operator == (const ddouble& v,double u)
{  return v[0] == u;}

int operator <= (const ddouble& u,const ddouble& v)
{  return u[0] <= v[0];}

int operator <= (double u,const ddouble& v)
{  return u <= v[0];}

int operator <= (const ddouble& v,double u)
{  return v[0] <= u;}

int operator >= (const ddouble& u,const ddouble& v)
{  return u[0] >= v[0];}

int operator >= (double u,const ddouble& v)
{  return u >= v[0];}

int operator >= (const ddouble& v,double u)
{  return v[0] >= u;}


int operator > (const ddouble& u,const ddouble& v)
{  return u[0] > v[0];}

int operator > (double u,const ddouble& v)
{  return u > v[0];}

int operator > (const ddouble& v,double u)
{  return v[0] > u;}

int operator < (const ddouble& u,const ddouble& v)
{  return u[0] < v[0];}

int operator < (double u,const ddouble& v)
{  return u < v[0];}

int operator < (const ddouble& v,double u)
{  return v[0] < u;}

ddouble operator + (const ddouble& x, const ddouble& y){
  ddouble r;
    r[0]    = x[0] + y[0];r[1]  = x[1] + y[1];
  return r;
}

ddouble operator + (double x, const ddouble& y)
{   ddouble r(y);  r[0] += x;  return r;}

ddouble operator + (const ddouble& y, double x)
{   ddouble r(y);  r[0] += x;  return r;}
```

```
ddouble operator - (const ddouble& x, const ddouble& y)
{   ddouble r;  r[0] = x[0] - y[0];r[1] = x[1] - y[1]; return r;}

ddouble operator - (double x, const ddouble& y)
{  ddouble r; r[1]  = - y[1];  r[0] = x - y[0];   return r;}

ddouble operator - (const ddouble& x, double y)
{  ddouble  r(x);  r[0] -= y;  return r;        }

ddouble operator * (const ddouble& x, const ddouble& y)
{ ddouble r; r[0] = x[0]*y[0]; r[1]=x[0]*y[1]+x[1]*y[0];return r;}


ddouble operator * (double x, const ddouble& y)
{ddouble r; r[0]    = x * y[0]; r[1]    = x * y[1]; return r;}

ddouble operator * ( const ddouble& y, double x)
{return x * y;}

ddouble operator / (const ddouble& x, const ddouble& y)
{  ddouble r; r[0] = x[0]/y[0]; r[1]=(x[1]-x[0]*y[1]/y[0])/y[0];return r;
    }

ddouble operator / (double x, const ddouble& y)
{ ddouble r; r[0] = x/y[0]; r[1]=-x*y[1]/y[0]/y[0];return r;}

ddouble operator/(const ddouble& x, double y)
{ddouble r; r[0] = x[0]/y; r[1]=x[1]/y;return r;}

ddouble exp (const ddouble& x)
{  ddouble r;r[0] = exp(x[0]); r[1] = x[1]*r[0];return r;}

ddouble log (const ddouble& x)
{ ddouble r;r[0] = log(x[0]);r[1]     = x[1]/x[0];return r;}

ddouble sqrt (const ddouble& x)
{ ddouble r;r[0] = sqrt(x[0]); r[1] = 0.5*x[1]/(eps+r[0]);return r;}

ddouble sin (const ddouble& x)
{   ddouble r; r[0]=sin(x[0]); r[1]=x[1]*cos(x[0]);  return r;}

ddouble cos (const ddouble& x)
{   ddouble r; r[0]=cos(x[0]); r[1]=-x[1]*sin(x[0]);  return r;}

ddouble tan (const ddouble& x) { return (sin(x) / cos(x));}

ddouble pow (const ddouble& x,double y) {return exp(log(x) * y);}

ddouble abs (const ddouble& x)
{ ddouble y;if(x[0] >= 0) y=x; else y = -x; return y;}

ddouble erfc (const ddouble& x)
{ ddouble y;y[1] = -2*x[1]*exp(-x[0]*x[0])/sqrtpi; y[0] = erfc(x[0]);
    return y;}

#endif
```