Homework 5

Problem

1. Implement linear Conjugate Gradient method for following function:
   (a) $f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$
2. Implement nonlinear Conjugate Gradient methods for the following functions:
   (a) $f(x,y) = 40(y-x^2)^2 + (1-x)^2$
   (b) $f(x,y) = (1.5-x+xy)^2 + (2.25-x+xy^2)^2 + (2.625-x+xy^3)^2$
3. Discuss their performances between nonlinear CGs:

Implementation

1. Implementation
   (a) **LinearCG**
      i. LinearCG is the method for linear function, which can be represented as $\frac{1}{2}x^T Ax - bx$
      ii. Therefore after deriving matrix A from the function, then we can apply this as input on a computing.
      iii. In the implementation, the constructor of LinearCG takes A and b as input argument
      iv. As termination condition, the residual r_k should be zero, when the convergence occrus. Because of a numerical calculation, it is not exact zero value, but is set as near zero, threshold $1e-2$

```
#ifndef __LinearCG__
#define __LinearCG__

#include "multivariate.h"

namespace numerical_optimization {

template<typename VectorTf>
class LinearCG : public Multivariate<VectorTf> {
public:
    using Base = Multivariate<VectorTf>;
    using Base::Base;
    using Base::plot;
    using Base::function;
```

```
    using Base::gradient;
    using function_t = typename Base::function_t;
    using MatrixTf = Eigen::Matrix<typename VectorTf::Scalar,
    ↪ VectorTf::RowsAtCompileTime, VectorTf::RowsAtCompileTime>;

    // constructor
    LinearCG(function_t f, MatrixTf A, VectorTf b):Base(f),A(A),b(b){};

    // compute
    VectorTf eval(const VectorTf& init=VectorTf::Random(), float e=epsilon)
    ↪ override {
        VectorTf xk = init;
        VectorTf rk = A*xk - b;
        VectorTf pk = -rk;

        for(size_t i=0; i<this->iter; i++) {
            if(rk.isZero(1e-2)) break;
#ifdef BUILD_WITH_PLOTTING
            plot.emplace_back(std::make_pair(xk, function(xk)));
#endif
            double inv = 1/(pk.transpose()*A*pk);
            double alpha_k = (rk.transpose()*rk);
            alpha_k *= inv;

            xk = xk + alpha_k*pk;
            VectorTf rk1 = rk + alpha_k*A*pk;
            double invv = 1/(rk.transpose()*rk);
            double beta_k1 = (rk1.transpose()*rk1);
            beta_k1 *= inv;

            pk = -rk1 + beta_k1*pk;
            rk = rk1;
        }
        return pk;
    };
private:
    MatrixTf A;
    VectorTf b;
};
//// ////////////////////////////////////////////////
}/// the end of namespace numerical_optimization ///
////////////////////////////////////////////////////
#endif //__LinearCG__
```

(a) **NonlinearCG**

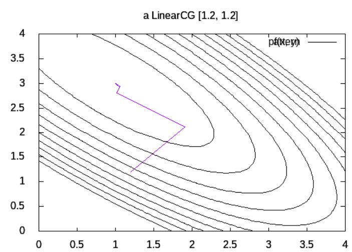
i. As similar as LinearCG, to terminate the criterion should be g_k is zero, but as same reason, currently set as $1e-4$

```
#ifndef __NonlinearCG__
#define __NonlinearCG__

#include "multivariate.h"

namespace numerical_optimization {
namespace nonlinear_cg {
enum Beta { CG_FR, CG_PR, CG_HS };
};

template<typename VectorTf, nonlinear_cg::Beta BetaMethod>
class NonlinearCG : public Multivariate<VectorTf> {
public:
    using Base = Multivariate<VectorTf>;
    using Base::Base;
    using Base::plot;
    using Base::function;
    using Base::gradient;
    using function_t = typename Base::function_t;

    // constructors
    NonlinearCG(function_t f):Base(f){};

    // compute
    VectorTf eval(const VectorTf& init=VectorTf::Random(), float e=epsilon) override {
        // 1. initialize
        VectorTf xk = init;
        double f0 = function(init);
        VectorTf gk = gradient(init);
        VectorTf pk = -gk;

        // 2. loop
        // k represents index
        for(size_t i=0; i<this->iter; i++) {
            if(gk.isZero(1e-4)) break;
#ifdef BUILD_WITH_PLOTTING
            plot.emplace_back(std::make_pair(xk, function(xk)));
#endif
            double alpha = this->line_search_inexact(xk, pk, 0.99, 0.5, 3);
```

```

            xk = xk + alpha * pk;
            VectorTf gk1 = gradient(xk);

            double beta_k = 0.0;
            if constexpr (BetaMethod==nonlinear_cg::Beta::CG_FR) {
                float inv = 1/gk1.dot(gk1);
                beta_k = (gk1.dot(gk1))*inv;
            } else if constexpr (BetaMethod==nonlinear_cg::Beta::CG_PR) {
                float inv = 1/gk1.dot(gk1);
                beta_k = (gk1.dot(gk1-gk))*inv;
            } else if constexpr (BetaMethod==nonlinear_cg::Beta::CG_HS) {
                float inv = 1/(gk1-gk).dot(pk);
                beta_k = (gk1.dot(gk1-gk))*inv;
            }

            pk = -gk1 + beta_k * pk;
            gk = gk1;
        }
        return pk;
    };
};
//// ///////////////////////////////////////////////////
}/// the end of namespace numerical_optimization ///
////////////////////////////////////////////////////////
#endif //__NonlinearCG__
```

Plotting

1. LinearCG behavior

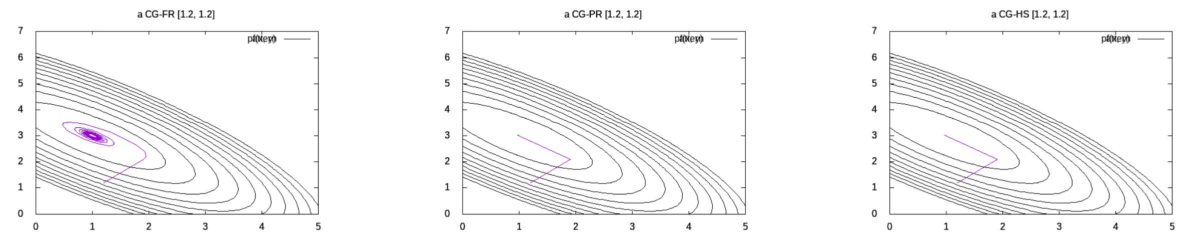


(a) LinearCG

Figure 1: $f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$

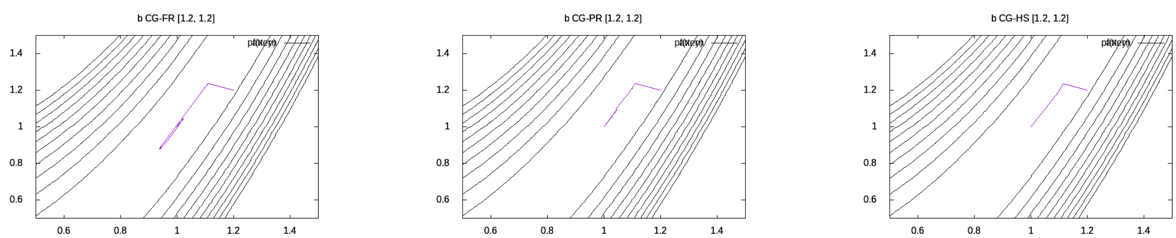
2. NonlinearCG behavior

   - According to Hager and Zhang [1], in Fletcher-Reeves scheme, a jamming can occur, when the search direction nearly orthogonal to the gradient
   - CG-PR is the method to escape this problem has the fastest convergence speed
     (a) $\beta_k^{PR} = \frac{y_k^T g_{k+1}}{\|g_k\|^2}$, where $y_k = g_{k+1} - g_k$. When jamming occurs $g_{k+1} \approx g_k, \beta_k^{PR} \approx 0$, and $d_{k+1} \approx -g_{k+1}$
     (b) It means when jamming occurs, the search direction is not orthogonal to the gradient
   - This situation is observied apparently in *Figure2*, in the CG-FR cases, the plot show the moving purple line following the contour. But, not in the CG-PR and CG-HS cases.
   - Also, *Figure3* show this type of result, which in the case of CG-FR makes convergence failed.



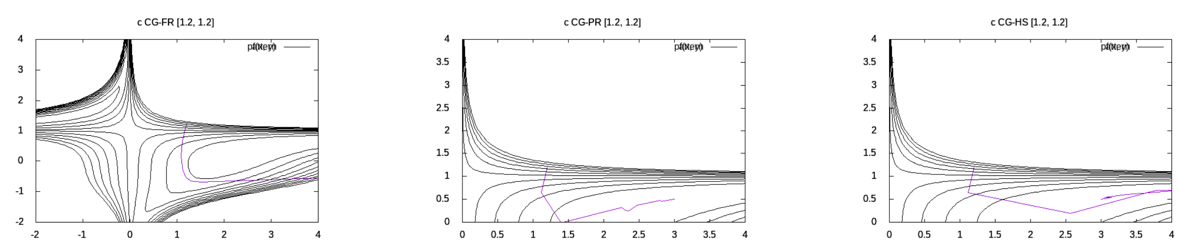
(a) CG-FR (b) CG-PR (c) CG-HS

Figure 2: $f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$



(a) CG-FR (b) CG-PR (c) CG-HS

Figure 3: $f(x,y) = 40(y-x^2)^2 + (1-x)^2$



(a) CG-FR (b) CG-PR (c) CG-HS

Figure 4: $f(x,y) = (1.5-x+xy)^2 + (2.25-x+xy^2)^2 + (2.625-x+xy^3)^2$

Performance

1. Convergence speed
   - As said before, CG-FR fails to converge, when the search direction is nearly orthogonal to gradient. And also, this affects Convergence speed
   - Especially, the function b case shows the failure case of convergence, which is

indicatied in *Figure4*

- Otherwise, CG-HS method has the fastest convergence speed

initial point	$f(x, y)$	Performance(x, y)		
		CG-FR	CG-PR	CG-HS
[1.2, 1.2]	(b)	62409445493 ns	320895182 ns	41624827 ns
	(c)	740809097 ns	349752676 ns	80612994 ns

References

[1] William W Hager and Hongchao Zhang. "Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent". In: *ACM Transactions on Mathematical Software (TOMS)* 32.1 (2006), pp. 113–137.