

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 occurs. Because of a numerical calculation, it is not exact zero value, but is set as near zero, threshold $1e - 2$

```
1  #ifndef __LinearCG__
2  #define __LinearCG__
3
4  #include "multivariate.h"
5
6  namespace numerical_optimization {
7
8  template<typename VectorTf>
9  class LinearCG : public Multivariate<VectorTf> {
10 public:
11     using Base = Multivariate<VectorTf>;
12     using Base::Base;
13     using Base::plot;
14     using Base::function;
```

Homework 5

```
15     using Base::gradient;
16     using function_t = typename Base::function_t;
17     using MatrixTf = Eigen::Matrix<typename VectorTf::Scalar,
    ↪ VectorTf::RowsAtCompileTime, VectorTf::RowsAtCompileTime>;
18
19     // constructor
20     LinearCG(function_t f, MatrixTf A, VectorTf b):Base(f),A(A),b(b){};
21
22     // compute
23     VectorTf eval(const VectorTf& init=VectorTf::Random(), float e=epsilon)
    ↪ override {
24         VectorTf xk = init;
25         VectorTf rk = A*xk - b;
26         VectorTf pk = -rk;
27
28         for(size_t i=0; i<this->iter; i++) {
29             if(rk.isZero(1e-2)) break;
30 #ifdef BUILD_WITH_PLOTTING
31             plot.emplace_back(std::make_pair(xk, function(xk)));
32 #endif
33             double inv = 1/(pk.transpose()*A*pk);
34             double alpha_k = (rk.transpose()*rk);
35             alpha_k *= inv;
36
37             xk = xk + alpha_k*pk;
38             VectorTf rk1 = rk + alpha_k*A*pk;
39             double invv = 1/(rk.transpose()*rk);
40             double beta_k1 = (rk1.transpose()*rk1);
41             beta_k1 *= inv;
42
43             pk = -rk1 + beta_k1*pk;
44             rk = rk1;
45         }
46         return pk;
47     };
48 private:
49     MatrixTf A;
50     VectorTf b;
51 };
52 //////////////////////////////////////
53 }/// the end of namespace numerical_optimization ///
54 //////////////////////////////////////
55 #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 $1e-4$

```
1  #ifndef __NonlinearCG__
2  #define __NonlinearCG__
3
4  #include "multivariate.h"
5
6  namespace numerical_optimization {
7  namespace nonlinear_cg {
8  enum Beta { CG_FR, CG_PR, CG_HS };
9  };
10
11 template<typename VectorTf, nonlinear_cg::Beta BetaMethod>
12 class NonlinearCG : public Multivariate<VectorTf> {
13 public:
14     using Base = Multivariate<VectorTf>;
15     using Base::Base;
16     using Base::plot;
17     using Base::function;
18     using Base::gradient;
19     using function_t = typename Base::function_t;
20
21     // constructors
22     NonlinearCG(function_t f):Base(f){};
23
24     // compute
25     VectorTf eval(const VectorTf& init=VectorTf::Random(), float e=epsilon)
26     ↪ override {
27         // 1. initialize
28         VectorTf xk = init;
29         double f0 = function(init);
30         VectorTf gk = gradient(init);
31         VectorTf pk = -gk;
32
33         // 2. loop
34         // k represents index
35         for(size_t i=0; i<this->iter; i++) {
36             if(gk.isZero(1e-4)) break;
37
38             #ifdef BUILD_WITH_PLOTTING
39                 plot.emplace_back(std::make_pair(xk, function(xk)));
40             #endif
41
42             double alpha = this->line_search_inexact(xk, pk, 0.99, 0.5, 3);
```

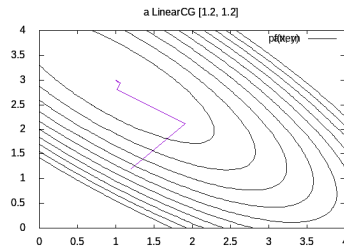
Homework 5

```
40
41     xk = xk + alpha * pk;
42     VectorTf gk1 = gradient(xk);
43
44     double beta_k = 0.0;
45     if constexpr (BetaMethod==nonlinear_cg::Beta::CG_FR) {
46         float inv = 1/gk1.dot(gk1);
47         beta_k = (gk1.dot(gk1))*inv;
48     } else if constexpr (BetaMethod==nonlinear_cg::Beta::CG_PR) {
49         float inv = 1/gk1.dot(gk1);
50         beta_k = (gk1.dot(gk1-gk))*inv;
51     } else if constexpr (BetaMethod==nonlinear_cg::Beta::CG_HS) {
52         float inv = 1/(gk1-gk).dot(pk);
53         beta_k = (gk1.dot(gk1-gk))*inv;
54     }
55
56     pk = -gk1 + beta_k * pk;
57     gk = gk1;
58 }
59 return pk;
60 };
61 };
62 //////////////////////////////////////
63 }/// the end of namespace numerical_optimization ///
64 //////////////////////////////////////
65 #endif //__NonlinearCG__
```

Homework 5

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 observed 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.

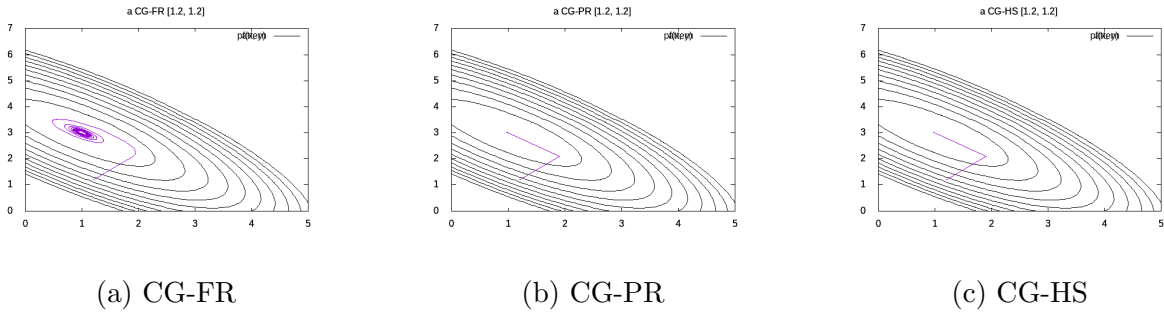


Figure 2: $f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$

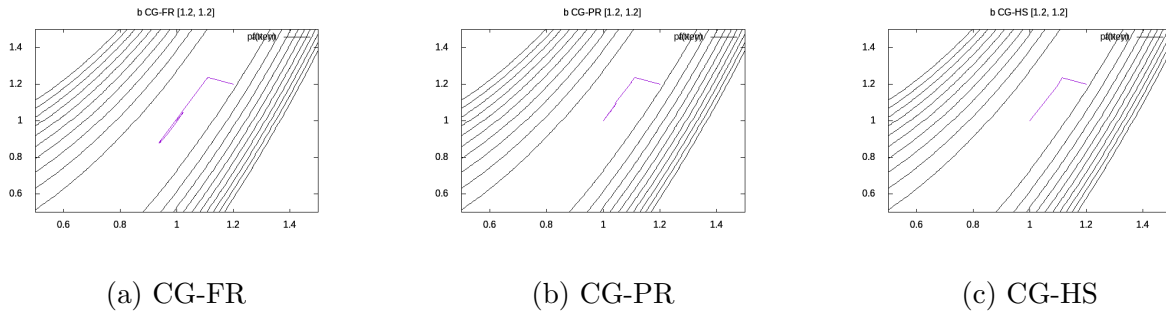


Figure 3: $f(x, y) = 40(y - x^2)^2 + (1 - x)^2$

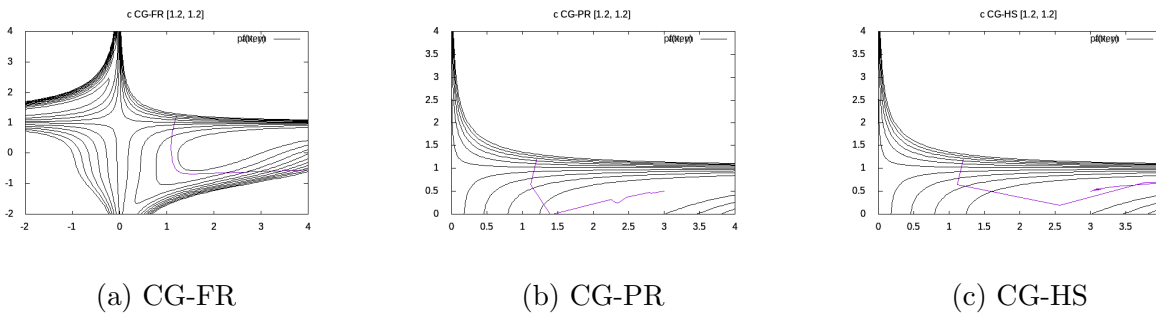


Figure 4: $f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$

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

Homework 5

indicated in *Figure 4*

- 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.