Übungen zur Strukturoptimierung mit Programm MAXIMA

1 Definitions for lbfgs

Function: **lbfgs** *(FOM, X, X0, epsilon, iprint)*

Finds an approximate solution of the unconstrained minimization of the figure of merit *FOM* over the list of variables *X*, starting from initial estimates *X0*, such that *norm grad FOM* < *epsilon max(1, norm X)*.

The algorithm applied is a limited-memory quasi-Newton (BFGS) algorithm. It is called a limited-memory method because a low-rank approximation of the Hessian matrix inverse is stored instead of the entire Hessian inverse. Each iteration of the algorithm is a line search, that is, a search along a ray in the variables *X*, with the search direction computed from the approximate Hessian inverse. The FOM is always decreased by a successful line search. Usually (but not always) the norm of the gradient of FOM also decreases.

iprint controls progress messages printed by `lbfgs`.

`iprint``[1]` controls the frequency of progress messages.

`iprint[1] < 0` No progress messages.
`iprint[1] = 0` Messages at the first and last iterations.
`iprint[1] > 0` Print a message every *`iprint`*`[1]` iterations.

`iprint``[2]` controls the verbosity of progress messages.

`iprint[2] = 0` Print out iteration count, number of evaluations of *FOM*, value of *FOM*, norm of the gradient of *FOM*, and step length.
`iprint[2] = 1` Same as *`iprint`*`[2] = 0`, plus *X0* and the gradient of *FOM* evaluated at *X0*.
`iprint[2] = 2` Same as *`iprint`*`[2] = 1`, plus values of *X* at each iteration.
`iprint[2] = 3` Same as *`iprint`*`[2] = 2`, plus the gradient of *FOM* at each iteration.

The columns printed by `lbfgs` are the following.

`I`	Number of iterations. It is incremented for each line search.
`NFN`	Number of evaluations of the figure of merit.
`FUNC`	Value of the figure of merit at the end of the most recent line search.
`GNORM`	Norm of the gradient of the figure of merit at the end of the most recent line search.
`STEPLENGTH`	An internal parameter of the search algorithm.

P0:

```
load(lbfgs);
```

```
FOM:(1-x)^2;
lbfgs(FOM,[x],[3],1.1E-2,[1,0]);
```

P1:

```
load(lbfgs);
FOM:(1-x)^2+(y-1)^2;
lbfgs(FOM,[x,y],[3,4],1.E-2,[1,0]);
```

ROSENBROCK:

```
load(lbfgs);
FOM:(1-x)^2+100*(y-x^2)^2;
lbfgs(FOM,[x,y],[3,4],1.1E-5,[1,0]);
```

2 Maxima Function for constrained Minimization by using augmented_lagrangian _method

Function: **augmented_lagrangian_method** *(FOM, xx, C, yy, optional_args)*

Returns an approximate minimum of the expression *FOM* with respect to the variables *xx*, holding the constraints *C* equal to zero. *yy* is a list of initial guesses for *xx*. The method employed is the augmented Lagrangian method.

`optional_args` represents additional arguments, specified as *`symbol = value`*. The optional arguments recognized are:

`niter`	Number of iterations of the augmented Lagrangian algorithm
`lbfgs_tolerance`	Tolerance supplied to LBFGS
`iprint`	IPRINT parameter (a list of two integers which controls verbosity) supplied to LBFGS
`%lambda`	Initial value of `%lambda` to be used for calculating the augmented

`load(augmented_lagrangian)` loads this function.

AUGMENTED1

```
load(lbfgs);
load(augmented_lagrangian);
FOM:x^2+2*y^2;
xx:[x,y];
yy:[1,1];
C:[x+y-1];
augmented_lagrangian_method(FOM,xx,C,yy,iprint=[-1,0]);
```

AUGMENTED2

```
load(lbfgs);
load(augmented_lagrangian);
FOM:x^2+2*y^2;
xx:[x,y];
yy:[1,1];
C:[x+y-1,x+2*y];
augmented_lagrangian_method(FOM,xx,C,yy,iprint=[-1,0]);
```

AUGMENTED2

```
load(lbfgs);
load(augmented_lagrangian);
FOM:x^2+2*y^2;
xx:[x,y];
yy:[1,1];
C:[x-1,x+2*y];
augmented_lagrangian_method(FOM,xx,C,yy,iprint=[-1,0]);
```

3 Definitions for simplex

Function: **linear_program** *(A, b, c)*

> `linear_program` is an implementation of the simplex algorithm. `linear_program(A, b, c)` computes a vector *x* for which `c.x` is minimum possible among vectors for which `A.x = b` and `x >= 0`. Argument *A* is a matrix and arguments *b* and *c* are lists.
>
> `linear_program` returns a list which contains the minimizing vector *x* and the minimum value `c.x`. If the problem is not bounded, it returns Problem not bounded! and if the problem is not feasible, it returns Problem not feasible! .

To use this function first load the `simplex` package with `load(simplex);`.

SIMPLEX1:

```
A:matrix([1,1,-1,0],[2,-3,0,-1],[4,5,0,0]);
b:[1,1,6];
c:[1,-2,0,0];
load(simplex);
linear_program(A,b,c);
```

SIMPLEX2

```
load(simplex);
A:matrix([3,4],[2,5]);
b:[4.5,5];
c:[24,40];
linear_program(A,b,c);:
```

SIMPLEX2LAGR

```
load(lbfgs);
load(augmented_lagrangian);
FOM:4.5*x+5*y;
xx:[x,y];
yy:[1,1];
C:[3*x+2*y-24,4*x+5*y-40];
augmented_lagrangian_method(FOM,xx,C,yy,iprint=[-1,0]);
```

SIMPLEX2LAGR1:

```
load(lbfgs);
load(augmented_lagrangian);
FOM:4.5*x+5*y;
```

```
xx:[x,y,t1,t2];
yy:[1,1,1,1];
C:[t1^2-3*x-2*y+24,t2^2-4*x-5*y+40];
augmented_lagrangian_method(FOM,xx,C,yy,iprint=[-1,0]);
```

4 Definitions for COBYLA

COBYLA minimizes an objective function F(X) subject to M inequality constraints of the form g(X) >= 0 on X, where X is a vector of variables that has N components.

Equality constraints g(X)=0 can often be implemented by a pair of inequality constraints g(X)>=0 and -g(X)>= 0. Maxima's interface to COBYLA allows equality constraints and internally converts the equality constraints to a pair of inequality constraints.

The algorithm employs linear approximations to the objective and constraint functions, the approximations being formed by linear interpolation at N+1 points in the space of the variables. The interpolation points are regarded as vertices of a simplex. The parameter RHO controls the size of the simplex and it is reduced automatically from RHOBEG to RHOEND. For each RHO the subroutine tries to achieve a good vector of variables for the current size, and then RHO is reduced until the value RHOEND is reached. Therefore RHOBEG and RHOEND should be set to reasonable initial changes to and the required accuracy in the variables respectively, but this accuracy should be viewed as a subject for experimentation because it is not guaranteed. The routine treats each constraint individually when calculating a change to the variables, rather than lumping the constraints together into a single penalty function. The name of the subroutine is derived from the phrase Constrained Optimization BY Linear Approximations.

Function: **fmin_cobyla** *(F, X, Y)*
Function: **fmin_cobyla** *(F, X, Y, optional_args)*

Returns an approximate minimum of the expression *F* with respect to the variables *X*, subject to an optional set of constraints. *Y* is a list of initial guesses for *X*.

F must be an ordinary expressions, not names of functions or lambda expressions.

`optional_args` represents additional arguments, specified as *`symbol = value`*. The optional arguments recognized are:

`constraints` List of inequality and equality constraints that must be satisfied by *X*. The inequality constraints must be actual inequalities of the form `g(X) >= h(X)` or `g(X) <= h(X)`. The equality constraints must be of the form `g(X) = h(X)`.

`rhobeg` Initial value of the internal RHO variable which controls the size of simplex. (Defaults to 1.0)
`rhoend` The desired final value rho parameter. It is approximately the accuracy in the variables. (Defaults to 1d-6.)

`iprint` Verbose output level. (Defaults to 0)

- 0 - No output
- 1 - Summary at the end of the calculation

- 2 - Each new value of RHO and SIGMA is printed, including the vector of variables, some function information when RHO is reduced.
- 3 - Like 2, but information is printed when F(X) is computed.

`Maxfun` The maximum number of function evaluations. (Defaults to 1000).

On return, a vector is given:

1. The value of the variables giving the minimum. This is a list of elements of the form *`var = value`* for each of the variables listed in *X*.
2. The minimized function value
3. The number of function evaluations.
4. Return code with the following meanings
    1. 0 - No errors.
    2. 1 - Limit on maximum number of function evaluations reached.
    3. 2 - Rounding errors inhibiting progress.

`load(fmin_cobyla)` loads this function.

Function: **bf_fmin_cobyla** *(F, X, Y)*
Function: **bf_fmin_cobyla** *(F, X, Y, optional_args)*

This function is identical to `fmin_cobyla`, except that bigfloat operations are used, and the default value for *rhoend* is `10^(fpprec/2)`.

COBYLA1

```
/*
 * 2D unit circle
 * f = x1*x2
 * 1-x1^2-x2^2 >= 0
 * True solution: x1=1/sqrt(2), x2=-1/sqrt(2)
 */
fmin_cobyla(x1*x2, [x1, x2], [1,1], constraints = [1>=x1^2+x2^2], iprint=1);
```

COBYLA2

```
/*
 * Test problem 8 (Rosen-Suzuki)
 * f = x1^2 + x2^2 + 2*x3^2 + x4^2 - 5*x1 - 5*x2 - 21*x3 + 7*x4
 * 8-x1^2-x2^2-x3^2-x4^2-x1+x2-x3+x4 >= 0
 * 10-x1^2-2*x2^2-x3^2-2*x4^2+x1+x4 >= 0
 * 5-2*x1^2-x2^2-x3^2-2*x1+x2+x4
 * True solution: x1=0, x2 = 1, x3 = 2, x4 = -1
 */
fmin_cobyla(x1^2 + x2^2 + 2*x3^2 + x4^2 - 5*x1 - 5*x2 - 21*x3 + 7*x4, [x1, x2, x3,
x4], [1,1,1,1], constraints = [8-x1^2-x2^2-x3^2-x4^2-x1+x2-x3+x4 >= 0,
 10-x1^2-2*x2^2-x3^2-2*x4^2+x1+x4 >= 0,
 5-2*x1^2-x2^2-x3^2-2*x1+x2+x4 >= 0], iprint = 1);
```

Reference: Maxima 5.24.0 Manual,
http://maxima.sourceforge.net/docs/manual/en/maxima.html#SEC_Top

V. Kobelev, Uni Siegen