Installation Instructions.

The compressed files `DSDP5.X.tar.gz` and `DSDP5.X.zip` contain an implementation of the dual-scaling algorithm for conic programming optimization problems.

Create the `DSDP5.X` directory structure and enter it. For example,

```
gunzip DSDP5.X.tar.gz
tar -xvf DSDP5.X.tar
```

or

```
unzip DSDP5.X.zip
```

Several executables may have been provided. If not, it will have to be compiled. DSDP is written in the C programming language. It has been tested using several different compilers and architectures. In particular, it has been tested using gcc-2.96 and gcc-3.2 (C and C++), Intel-6.0, Intel-7.1, Intel-8.0, and Microsoft Visual Studio 2003. It has been compiled on 32 bit and 64 bit architectures.

Compiling DSDP

Using Make

DSDP was developed using Make -- which is available on Linux and most Unix systems. To compile DSDP using Make

- Go to the DSDP directory and edit the file `make.include`
  - Go to the DSDP directory

    ```
    > cd DSDP5.7
    ```

  - Define the `DSDPROOT` variable as the full directory name. For example,

    ```
    DSDPROOT        = /home/benson/DSDP5.7
    ```

    For those using the GCC compilers, the default values for the remaining variables may suffice.
  - Edit the compiler flags `CC` and `OPTFLAGS`. For example,

    ```
    CC       = gcc
    ```

  - Edit the compiler optimization flag

    ```
    OPTFLAGS = -O3
    ```

  - Add timing flag. By default, timing routines are not implemented due to portability issues among architectures. However, DSDP can provide time profiles for several important operations by defining one of the following variables.

    ```
    DSDPTIMER   = NONE
    ```

    Users of the Microsoft compiler will probably need the flags

    ```
    DSDPTIMER   = DSDP_MS_TIME
    ```

Users of the GCC and many other compilers should consider defining

```
DSDPTIMER   = DSDP_TIME
```

The routines that call these two timing utilities can be found in the file **`dsdptime.c`** and edited.

  - Add additional compiler flags. See the next section for more information about specific compiler flags.
  - If the Matlab interface is required, also check and edit the `MEX` flag. For example,

```
MEX            = mex -O
```

  - In the same file, edit the location of the BLAS and LAPACK libraries and include any other libraries required to link to them such as `-lg2c` or `-lm`. For example,

```
LAPACKBLAS  = -llapack -lblas -lg2c -lm
```

Linking to a fast implementations of these libraries, such as ATLAS or Intel, will significantly improve performance.

- Compile the source code to create the DSDP library and drivers.

```
> make all
```

Alternatively, to compile just the DSDP Matlab routine:

```
> make dsdpmatlab
```

Alternatively, to compile just the DSDP library and C examples:

```
> make dsdpapi
```

If the DSDP Matlab mex function does not link with the library, try the next method of compilation. If problems persist, please send a copy of the compilation log to the developers.

Without Make

DSDP can also be compiled by copying all of the source and header files into another directory, compiled, and linked with a driver routine (such as one of the files in `DSDPROOT/examples/`). This process is demonstrated in the `dsdpagain` target in `DSDPROOT/examples/Makefile`. This method neglects the directory structure of the source code and the Makefile system, but it works fine.

Compiler Flags

BLAS and LAPACK

DSDP uses BLAS and LAPACK for many of the underlying operations and it must be linked to these libraries. Two of the eigenvalue routines require LAPACK version 3 or later. The routines in BLAS and LAPACK are called from the C programming language under the assumption that the routine names are lower case and end with an underscore. The most common linking problem occurs when these assumptions are not true. Several compiler flags can be defined to change these assumptions. Define

- Define `NOUNDERBLAS` if no underscore is appended to the end of routine names. This flag was used to link DSDP to the reference BLAS available in Matlab using the Microsoft Compiler.
- Define `CAPSBLAS` if the names of BLAS and LAPACK routine names use all capital letters.
- Define `__DSDP_NONAMEMANGLING` if a C++ compiler is being used and the BLAS and LAPACK routine names should not be changed.

See the makefiles in the distribution for examples of use of these terms. Those compiling in the Microsoft Windows Operating System usually need to define the `NOUNDERBLAS` flag.

DSDP also assumes that the `integer` type in Fortran is the same size as a `long int` in C, and a `double precision` variable in Fortran is the same size as a `double` in C. If problems persist, the macros and type definitions in the file **`dsdplapack.h`** will have to be edited.

Matlab

If the Matlab interface is be generated, one other flag may be of interest.

- `DSDPMATLAB` enables the use of the memory allocation and additional print statements available from Matlab.

The standard memory allocation and print statements will be used if this flag is not defined.

Testing

- Run the executables by switching to directory `DSDPROOT/exec` and test the examples.

  ```
  > dsdp5  truss1.dat-s
  > maxcut  graph1
  > theta  graph1
  ```

  Compare the output with the files `output.truss1` , `output.maxcut` , and `output.theta` . If the output from any of the tests differs significantly from the files, please report it to the developers.

- DSDP can be called from Matlab version 6.0 and higher. Run the sample problems by starting Matlab in the `DSDPROOT/matlab` directory and test the solver

  ```
  > check;
  ```

  Compare the output with the output in `check.out` . For help using the package, type

  ```
  > help dsdp;
  ```

  Several example Matlab files have been provided in `DSDPROOT/matlab` that create example problems, read data files, and verify solutions.

Help

For help with installation, please send a copy of the compilation log to the developers.

Generated on Tue Sep 6 09:54:58 2005 for DSDP by doxygen *1.4.2*