Embedded Conic Solver (ECOS)

ECOS is a numerical software for solving convex second-order cone programs (SOCPs) of type

```
min  c'*x
s.t. A*x = b
     G*x <=_K h
```

where the last inequality is generalized, i.e. `h - G*x` belongs to the cone `K`. ECOS supports the positive orthant `R_+` and second-order cones `Q_n` defined as

```
Q_n = { (t,x) | t >= || x ||_2 }
```

In the definition above, t is a scalar and `x` is in `R_{n-1}`. The cone `K` is therefore a direct product of the positive orthant and second-order cones:

```
K = R_+ x Q_n1 x ... x Q_nN
```

Features of ECOS

- *ECOS runs on embedded platforms*. Written in ANSI C (except for the timing code), it can be compiled for any platform for which a C compiler is available. Excluding the problem setup part, no memory manager is needed for solving problem instances of same structure.
- *ECOS is efficient*. Using sparse linear algebra routines, it computes only what is really necessary. The interior point algorithm it implements is one of the fastest converging methods that are currently in use for solving convex conic problems.
- *ECOS has a tiny footprint*. The ECOS solver consists of 750 lines of C code (excluding the problem setup code).
- *ECOS is numerically robust*. Using regularization and iterative refinement coupled with a carefully chosen sparse representation of scaling matrices, millions of problem instances are solved reliably.
- *ECOS comes with a MATLAB and CVX 2.1 interface*, and *it is supported by YALMIP*. With CVX and YALMIP you can prototype, simulate and verify the performance of ECOS before you implement the very same code on your embedded hardware.
- *ECOS comes with a Python interface*. This interface is built on top of NUMPY and SCIPY and uses its sparse data structures.
- *There is a Julia interface for ECOS*. Julia is a high-level, high-performance language for technical and scientific computing. You can pull the Julia interface here.
- *ECOS is library-free*. No need to link any external library to ECOS, apart from `AMD` and `sparseLDL`, both from Timothy A. Davis, which are included in this project.

Credits

The solver is essentially based on Lieven Vandenberghe's CVXOPT ConeLP solver, although it differs in the particular way the linear systems are treated.

The following people have been, and are, involved in the development and maintenance of ECOS:

- Alexander Domahidi (principal developer)
- Eric Chu (Python interface, unit tests)
- Stephen Boyd (methods and maths)
- Michael Grant (CVX interface)
- Johan Löfberg (YALMIP interface)
- Karanveer Mohan (Julia interface)

The main technical idea behind ECOS is described in a short paper. More details are given in Alexander Domahidi's PhD Thesis in Chapter 9.

If you find ECOS useful, you can cite it using the following BibTex entry:

```
@INPROCEEDINGS{bib:Domahidi2013ecos,
author={Domahidi, A. and Chu, E. and Boyd, S.},
booktitle={European Control Conference (ECC)},
title={{ECOS}: {A}n {SOCP} solver for embedded systems},
year={2013},
pages={3071-3076}
}
```

Using ECOS with CVX

The simplest way to use ECOS is to install a CVX 2.0 shim. For this to work, you must have the latest version of CVX installed in MATLAB. Once CVX is installed, add your ECOS directory to your MATLAB install path and run `cvx_setup`.

```
addpath <ecos-directory>/matlab
cvx_setup
```

This will automatically detect the ECOS shim and add it to CVX. If you want to ensure you have the latest binary for ECOS, instead run

```
cd <ecos-directory>/matlab
makemex
addpath <ecos-directory>/matlab
cvx_setup
```

This will build ECOS and install the CVX shim. Please report any error messages to us. The old method also works:

```
cd <ecos-directory>/matlab
cvx_install_ecos
```

This is maintained for compatibility issues (i.e., for users who have not upgraded to CVX 2.0).

Once the ECOS shim is installed, the CVX solver can be switched using the `cvx_solver` command. For instance,

```
cvx_begin
     cvx_solver ecos     % without this line, CVX will use its default solve
     variable x(n)

     minimize sum_square(A*x - b)
     subject to
          x >= 0
cvx_end
```

IMPORTANT: Not all of CVX's atoms are SOCP-representable. Some of the atoms implemented in CVX require the use of SDP cones. Some atoms that could be implemented with a second-order cone are instead implemented as SDPs, but these are automatically converted to SOC cones. See Issue #8 for more information.

Using ECOS with CVXPY

CVXPY is a powerful Python modeling framework for convex optimization, similar to the MATLAB counterpart CVX. ECOS is one of the default solvers in CVXPY, so there is nothing special you have to do in order to use ECOS with CVXPY, besides specifying it as a solver. Here is a small example from the CVXPY tutorial:

```
# Solving a problem with different solvers.
x = Variable(2)
```

```
obj = Minimize(norm(x, 2) + norm(x, 1))
constraints = [x >= 2]
prob = Problem(obj, constraints)

# Solve with ECOS.
prob.solve(solver=ECOS)
print "optimal value with ECOS:", prob.value
```

Using ECOS with YALMIP

As of release R20130628, YALMIP supports ECOS as a solver - simply use the command

```
sdpsettings('solver','ecos');
```

to select ECOS as the solver for your problem. Below is a toy example:

```
% Solve 1000 SOCPs
x = sdpvar(3,1);
Ufo= [norm(x) <= 2, norm(x+1) <= 2];
plot(Ufo,x,'y',1000,sdpsettings('solver','ecos'))
```

Using ECOS in MATLAB

Compiling ECOS for MATLAB

ECOS comes with a makefile which resides in the `matlab` subdirectory of the code. To build ECOS for MATLAB:

```
cd <ecos-directory>/matlab
makemex
```

You should now have a binary file `ecos.[ending]`, with a platform-specific ending. This is the solver binary. Add the directory `<ecos-directory>/matlab` to your path to be able to call ECOS from any place. The command

```
makemex clean
```

deletes unecessary files that were produced during compilation.

Calling ECOS from MATLAB

You can directly call ECOS from Matlab using its native interface:

```
[x,y,info,s,z] = ecos(c,G,h,dims,A,b)
```

It takes the problem data `c,G,h,A,b` and some dimension information that is given in the struct `dims`. Note that `A` and `G` have to be given in sparse format. The equality constraints defined by `A` and `b` are optional and can be omitted. The `dims` structure has the following fields:

```
dims.l - scalar, dimension of positive orthant (LP-cone) R_+
dims.q - vector with dimensions of second order cones
```

The length of `dims.q` determines the number of second order cones. If you do not have a cone in your problem, use the empty matrix `[ ]` instead, for example `dims.q = [ ]` if you do not have second-order cones. After a solve, ECOS returns the following variables

```
  x: primal variables
```

```
  y: dual variables for equality constraints
  s: slacks for Gx + s <= h, s \in K
  z: dual variables for inequality constraints s \in K
```

In addition, the struct `info` is returned which contains the following fields:

```
    exitflag: 0=OPTIMAL, 1=PRIMAL INFEASIBLE, etc. (see exitcodes section in
  infostring: gives information about the status of solution
       pcost: value of primal objective
       dcost: value of dual objective
        pres: primal residual on inequalities and equalities
        dres: dual residual
        pinf: primal infeasibility measure
        dinf: dual infeasibility measure
     pinfres: NaN
     dinfres: 3.9666e+15
         gap: duality gap
      relgap: relative duality gap
          r0: ???
      numerr: numerical error?
        iter: number of iterations
      timing: struct with timing information
```

Example: L1 minimization (Linear Programming)

In the following, we show how to solve a L1 minimization problem, which arises for example in sparse signal reconstruction problems (compressed sensing):

```
    minimize  ||x||_1         (L1)
  subject to  Ax = b
```

where `x` is in `R^n`, `A` in `R^{m x n}` with `m <= n`. We use the epigraph reformulation to express the L1-norm of `x`,

```
     x <= u
    -x <= u
```

where `u` is in `R^n`, and we minimize `sum(u)`. Hence the optimization variables are stacked as follows:

```
    z = [x; u]
```

With this reformulation, (L1) can be written as linear program (LP),

```
  minimize   c'*z
  subject to Atilde*z = b;    (LP)
             Gx <= h
```

where the inequality is w.r.t. the positive orthant. The following MATLAB code generates a random instance of this problem and calls ECOS to solve the problem:

```
% set dimensions and sparsity of A
n = 1000;
m = 10;
density = 0.01;

% linear term
c = [zeros(n,1); ones(n,1)];

% equality constraints
A = sprandn(m,n,density);
```

```
Atilde = [A, zeros(m,n)];
b = randn(m,1);

% linear inequality constraints
I = speye(n);
G = [  I -I;
      -I -I];
h = zeros(2*n,1);

% cone dimensions (LP cone only)
dims.l = 2*n;
dims.q = [];

% call solver
fprintf('Calling solver...');
z = ecos(c,G,h,dims,Atilde,b);
x = z(1:n);
u = z(n+1:2*n);
nnzx = sum(abs(x) > 1e-8);

% print sparsity info
fprintf('Optimal x has %d/%d (%4.2f%%) non-zero (>1e-8 in abs. value) entries
```

Example: Quadratic Programming

In this example, we consider problems of form

```
  minimize 0.5*x'*H*x + f'*x
subject to A*x <= b                (QP)
           Aeq*x = beq
           lb <= x <= ub
```

where we assume that `H` is positive definite. This is the standard formulation that also MATLAB's built-in solver `quadprog` uses. To deal with the quadratic objective, you have to reformulate it into a second-order cone constraint to directly call ECOS. We do provide a MATLAB interface called `ecosqp` that automatically does this transformation for you, and has the exact same interface as `quadprog`. Hence you can just use

```
[x,fval,exitflag,output,lambda,t] = ecosqp(H,f,A,b,Aeq,beq,lb,ub)
```

to solve (QP). See `help ecosqp` for more details. The last output argument, `t`, gives the solution time.

Using ECOS in Python

Compiling ECOS for Python

To create the Python interface, you need Numpy and Scipy. For installation instructions, see their respective pages. Once those are installed, the following lines of code should work:

```
cd <ecos-directory>/python
python setup.py install
```

You may need `sudo` privileges for a global installation.

Windows installation

Windows users may experience some extreme pain when installing ECOS for Python 2.7. We suggest switching to Linux or Mac OSX.

If you must use (or insist on using) Windows, we suggest using the Miniconda distribution to minimize this pain.

If during the installation process, you see the error message `Unable to find vcvarsall.bat`, you will need to install Microsoft Visual Studio Express 2008, since *Python 2.7* is built against the 2008 compiler.

If using a newer version of Python, you can use a newer version of Visual Studio. For instance, Python 3.3 is built against Visual Studio 2010.

Calling ECOS from Python

After installing the ECOS interface, you must import the module with

```
import ecos
```

This module provides a single function `ecos` with one of the following calling sequences:

```
solution = ecos.solve(c,G,h,dims)
solution = ecos.solve(c,G,h,dims,A,b,**kwargs)
```

The arguments `c`, `h`, and `b` are Numpy arrays (i.e., matrices with a single column). The arguments `G` and `A` are Scipy *sparse* matrices in CSR format; if they are not of the proper format, ECOS will attempt to convert them. The argument `dims` is a dictionary with two fields, `dims['l']` and `dims['q']`. These are the same fields as in the Matlab case. If the fields are omitted or empty, they default to 0. The argument `kwargs` can include the keywords `feastol`, `abstol`, `reltol`, `feastol_inacc`, `abstol_innac`, and `reltol_inacc` for tolerance values, `max_iters` for the maximum number of iterations, and the Boolean `verbose`. The arguments `A`, `b`, and `kwargs` are optional.

The returned object is a dictionary containing the fields `solution['x']`, `solution['y']`, `solution['s']`, `solution['z']`, and `solution['info']`. The first four are Numpy arrays containing the relevant solution. The last field contains a dictionary with the same fields as the `info` struct in the MATLAB interface.

Using ECOS in C

ECOS exports 3 functions, see ecos.h. You need to call these in the following sequence:

1. Setup

Setup allocates memory for ECOS, computes the fill-in reducing ordering and provides other initialization necessary before solve can start. Use the following function to initialize ECOS:

```
pwork* ECOS_setup(idxint n, idxint m, idxint p, idxint l, idxint ncones, idxi
                  pfloat* Gpr, idxint* Gjc, idxint* Gir,
                  pfloat* Apr, idxint* Ajc, idxint* Air,
                  pfloat* c, pfloat* h, pfloat* b);
```

where you have to pass the following arguments: *`n` is the number of variables,* `m` is the number of inequality constraints (dimension 1 of the matrix `G` and the length of the vector `h`), *`p` is the number of equality constraints (can be 0)* `l` is the dimension of the positive orthant, i.e. in `Gx+s=h, s in K`, the first `l` elements of `s` are `>=0` *`ncones` is the number of second-order cones present in `K`* `q` is an array of integers of length `ncones`, where each element defines the dimension of the cone *`Gpr`, `Gjc`, `Gir` are the the data, the column index, and the row index arrays, respectively, for the matrix `G` represented in column compressed storage (CCS) format (Google it if you need more information on this format, it is one of the standard sparse matrix representations)* `Apr`, `Ajc`, `Air` is the CCS representation of the matrix `A` (can be all `NULL` if no equalities are present) *`c` is an array of type `pfloat` of size `n`* `h` is an array of type `pfloat` of size `m` * `b` is an array of type `pfloat` of size `p` (can be `NULL` if no equalities are present) The setup function returns a struct of type `pwork`, which you need to define first.

2. Solve

After the initialization is done, you can call the solve function, which contains the actual interior point method, by `idxint ECOS_solve(pwork* w);` The return value is an integer, see below.

3. Cleanup

Call `void ECOS_cleanup(pwork* w, idxint keepvars);` to free all allocated memory.

Exitcodes

ECOS defines a number of exitcodes that indicate the quality of the returned solution. In general, positive values indicate that the solver has converged within the given tolerance. More specifically, + 0: optimal + 1: primal infeasible + 2: dual infeasible

An exact definition of when these flags are returned can be found in Alexander Domahidi's PhD Thesis in Chapter 9.4.2. (pp. 163).

Negative numbers indicate that the problem could not be solved to the required accuracy (the returned iterate might still be satisfactory - please do check the duality gap etc.) + -1: maximum number of iterations reached + -2: numerical problems (unreliable search direction) + -3: numerical problems (slacks or multipliers became exterior) + -7: unknown problem in solver

It is in general good practice to check the exitcode, in particular when solving optimization problems in an unsupervised, automated fashion (in a batch job, for example). Please report optimization problems for which ECOS struggles to converge to one of the authors.