iSWIFT - Documentation

Abhishek Pandala, Yanran Ding, Hae-Won Park

March 22, 2019

1 Introduction

iSWIFT is a small-scale quadratic programming solver written in ANSI C. It is programmed to solve QP's of the following form

$$\text{minimize } \frac{1}{2}\boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \tag{1a}$$

$$\text{subject to } \boldsymbol{A}\cdot\boldsymbol{x} = \boldsymbol{b} \tag{1b}$$

$$\boldsymbol{G}\cdot\boldsymbol{x} \leq \boldsymbol{h} \tag{1c}$$

It is assumed that the matrix P is symmetric positive definite and matrix A has full row rank.

2 Code Organization

iSWIFT is organised into the following source files listed in $...\src$ folder

- Prime.c - Contains three major functions needed by the QP solver (outlined in 3.1)
- Auxilary.c - Contains all the auxilary functions needed by the three major functions in Prime.c
- timer.c - Contains functions needed to time events in the QP solver; similar to tic,toc functions in matlab
- RUNQP.c - Contains a sample implementation of QP solver

The following header files can be found in $...\include$ folder

- GlobalOptions.h - Contains all the settings and exit flags of the QP solver
- Prime.h - Contains declarations of all the structures and functions used in iSWIFT
- timer.h - Contains declarations of functions in timer.c
- Matrices.h - Contains sample input data to run QP solver

Apart from these files, there is also an LDL package in $...\ldl$ which performs the Cholesky factorization and solves the linear system of equations. In the $...\matlab$ folder, we have two simulink files Swift_sfunc_e.c and Swift_sfunc.c and one matlab cmex file Swift_cmex.c. To build each of these functions, call the function Swift_mex.m with the filename as argument. For example, to build Swift_sfunc.c use the following command Swift_mex('Swift_sfunc').

3 Usage

3.1 C/C++

iSWIFT has the following three main functions listed in Prime.c file.

3.1.1 QP_SETUP

This function takes as input the following arguments

- n = no. of decision variables
- m = no. of inequality constraints

- p = no. of equality constraints
- Pjc, Pir, Ppr correspond to P matrix in CCS format
- Ajc, Air, Apr correspond to A matrix in CCS format
- Gjc, Gir, Gpr correspond to G matrix in CCS format
- c = column vector in cost function
- h = right hand side of inequality constraint matrix
- b = right hand side of equality constraint matrix
- $sigma\_d$ = centering parameter (chosen as 0)
- $Permut$ = permutation vector

With these arguments, the function creates a structure QP and also determines the initial starting points x_0, y_0, z_0 and s_0 for the algorithm. In the absence of equality constraints, one can set either p to 0 or any of the pointers corresponding to Ajc, Air, Apr, b to NULL. The time taken by this function can be found via setup_time.

3.1.2 QP_SOLVE

This function takes as argument the structure QP created by QP_SETUP and performs the following tasks

- Check the termination criterion
- Perform Mehrotra Predictor Corrector steps
- Update the primal and dual variables

The algorithm terminates when either the residuals and duality gap are within the tolerance values or when the maximum number of iterations are reached. The time taken by this function can be found via solve_time. The total time taken by the solver is the sum of setup_time and solve_time.

3.1.3 QP_CLEAN

This function cleans all the memory allocated by QP_SETUP function. Make sure all the information (including solution and stats) is extracted before calling this function.

A sample implementation is outlined in RUNQP.c file.

3.2 Simulink

iSWIFT has interface with Simulink real-time through gateway S-function. iSWIFT two S-function files Swift_sfunc_e.c and Swift_sfunc.c. The first S-function file is used when equality constraints are present and the second file is used when equality constraints are not present. Each of these functions takes the following input arguments

- P - Matrix in dense format
- c - column vector in cost function
- A - Matrix corresponding to equality constraints (for use in Swift_sfunc_e.c file only)
- b - Matrix corresponding to right hand side of equality constraint matrix (for use in Swift_sfunc_e.c file only)
- G - Matrix corresponding to inequality constraints
- h - Matrix corresponding to right hand side of inequality constraint matrix
- $sigma\_d$ - Centering parameter (chosen as 0)

Before building these simulink files, two things are to be considered. First, specify the length of the output vector in the macro "$\#define \quad NV$". This corresponds to the length of solution vector you want to retrieve from the QP solver. Second, the corresponding permutation vector has to be changed depending on the problem instance before building it using Swift_mex.m. Upon successful completion, the solver gives the solution vector, iteration count, setup time and solve time respectively.

4 Appendix

4.1 Compressed Column Storage format

iSWIFT operates on sparse matrices stored in Compressed Column Storage format. In this format, an $m \times n$ sparse matrix A that can contain nnz non-zero entries is stored as an integer array of Ajc of length $n+1$, an integer array Air of length nnz and a real array Apc of length nnz.

- The real array Apr holds all the nonzero entries of A in column major format
- The integer array Air holds the rows indices of the corresponding elements in Apr
- The integer array Ajc is defined as
  - $Ajc[0] = 0$
  - $Ajc[i] = Ajc[i-1]$ + number of non-zeros in i^{th} column of A

For the following sample matrix A,

$$A = \begin{bmatrix} 4.5 & 0 & 3.2 & 0 \\ 3.1 & 2.9 & 0 & 2.9 \\ 0 & 1.7 & 3.0 & 0 \\ 3.5 & 0.4 & 0 & 1.0 \end{bmatrix} \tag{2}$$

we have the following CCS representation

$$\begin{aligned} \text{int Ajc} &= \{0, 3, 6, 8, 10\} \\ \text{int Air} &= \{0, 1, 3, 1, 2, 3, 0, 2, 1, 3\} \\ \text{double Apr} &= \{4.5, 3.1, 3.5, 2.9, 1.7, 0.4, 3.2, 3.0, 2.9, 1.0\} \end{aligned} \tag{3}$$

4.2 Permutation vector

Performing LDL^T factorization on a sparse linear system of equations typically results in fill-in. A fill-in is a non-zero entry in L but not in A. To minimize fill-in, permutation matrices are used and the new system

$$PAP^T \tag{4}$$

is factorized. Obtaining a perfect elimination ordering (permutation matrix with least fill-in) is an NP-hard problem. Hence, heuristics are used to compute permutation matrices. Some of the popular ones are nested dissection and minimum degree ordering methods. iSWIFT uses the Approximate minimum degree ordering to compute permutation matrix. The user can opt to use other ordering methods as well. The following code snippet helps to compute permutation matrix in Matlab.

```
n = size(P,1);
m = size(G,1);
p = size(A,1);
Phi = [P A' G'; A zeros(p,m + p);G zeros(m,p) -eye(m,m)];
Permut = amd(Phi) - 1;
```