User's Guide for QDTOC Version 1.02 A C code for Solving Quadratic Discrete Optimal Control Problems

D. Ralph* Yi Xiao†

January 20, 2000

*Department of Mathematics and Statistics, The University of Melbourne
†Supported by

1 Introduction

QDTOC is a software package for solving the linear-quadratic Model Predictive Control problem. It is written in C and built on top of Meschach [4].

The linear-quadratic Model Predictive Control requires solution of structured QP problems repeatedly. This package intends to solve a Quadratic Discrete Time Optimal Control (Q-DTOC) problem as follows.

$$\begin{array}{rcll} \min J(\mathbf{x},\mathbf{u}) & = & \sum_{k=1}^{N} l_k(\mathbf{x}_k,\mathbf{u}_k) + l_{N+1}(\mathbf{x}_{N+1}) & (1a) \\ \text{subject to} & & \mathbf{x}_1 = \hat{\mathbf{x}}_1 & (1b) \\ & & \mathbf{x}_{k+1} = A_k\mathbf{x}_k + B_k\mathbf{u}_k + \mathbf{b}_k \quad (k=1,\cdots,N) & (1c) \\ & & C_k\mathbf{x}_k + D_k\mathbf{u}_k \le \mathbf{d}_k \quad (k=1,\cdots,N) & (1d) \\ & & C_k\mathbf{x}_{N+1} \le \mathbf{d}_{N+1} & (1e) \end{array} \tag{1}$$

where for $k=1,\cdots,N,\ \mathbf{x}_k \in \mathcal{R}_k^m, \mathbf{u}_k \in \mathcal{R}_k^n$, C_k, D_k have l_k rows, and

$$\begin{array}{rcl} l_k(\mathbf{x}_k,\mathbf{u}_k) & = & \mathbf{r}_k^T\mathbf{u}_k + \mathbf{z}_k^T\mathbf{x}_k + \frac{1}{2}(\mathbf{x}_k^TQ_k\mathbf{x}_k + 2\mathbf{u}_k^TS_k\mathbf{x}_k + \mathbf{u}_k^TR_k\mathbf{u}_k) \\ l_{N+1}(\mathbf{x}_{N+1}) & = & \mathbf{z}_{N+1}^T\mathbf{x}_{N+1} + \frac{1}{2}\mathbf{x}_{N+1}^TQ_{N+1}\mathbf{x}_{N+1} \end{array} \tag{2}$$

Methods used to solve (1) are stage wise active set methods (primal, dual and primal-dual) and the primal-dual interior point method [3, 2]. The user has the freedom to chose any of these four methods to solve his/her problem by using the basic function **qdtoc_prob_solve**. For more details, see **F7** below.

2 Installation

To install the QDTOC package, you need

1. A C compiler.
2. GNU make.
3. Meschach [4].
   Meschach is a C package for numerical linear algebra. It can be obtained from **netlib:** pub/netlib/c/meschach via ftp. It is assumed that Meschach has been installed. Note that QDTOC only requires a **basic** installation of Meschach. Meschach can also be retrieved from

   http://www.ms.unimelb.edu.au/ danny/DTOC/QDTOC/meschach/meschach.tgz

   and then extracted as follows:

```
gunzip meschach.tgz
tar -xvf meschach.tar
```

This will create a directory meschach, which contains the following files:

```
meschach0.shar,  meschach1.shar,  meschach2.shar, README
```

Following the instruction in **README**, install meschach.
NOTE: To install meschach under MSDOS using DJGPP (see below), in the makefle, change compiler from 'cc' to 'gcc'.

QDTOC is written in ANSI C, and it has been compiled and tested on the following platforms:

Digital Unix, on DEC Alpha, using gcc.
Linux on Intel, using gcc.
MSDOS, Using DJGPP (gcc for MSDOS)

A. To Install QDTOC Under Unix

1. Retrieve the file **qdtoc_1_02.tgz** from

   http://www.ms.unimelb.edu.au/d̃anny/DTOC/QDTOC/qdtoc_1_02.tgz

2. Extract the files from the archive.

```
gunzip qdtoc_1_02.tgz
tar -xvf qdtoc_1_02.tar
```

   This will create a directory qdtoc_c, which contains the following subdirectories:

```
src, include, test, doc, lib
```

3. Customise makefile **make.unx** in the qdtoc_c directory.

```
#
# C compiler
#
CC      = cc

#
```

```
# Linker
#
LINK    = cc

#
# Meschach installation directory
#
MESCHACH_ROOT   = $(HOME)/meschach
```

4. Go to the directory qdtoc_c/src, type

   ```
   make  -f makefile.unx
   ```

   This will compile the QDTOC source code, and create the QDTOC library **libqdtoc.a**.

5. Go to the directory qdtoc_c/test, type

   ```
   make  -f makefile.unx
   ```

   This will compile the QDTOC test program,

   ```
   make -f makefile.unx test
   ```

   will solve the test program, and

   ```
   make -f makefile.unx warmstart
   ```

   will solve the test program using the warm start procedure.

B. To Install QDTOC Under MSDOS

DJGPP is a port of gcc to MSDOS. For more information, see

http://www.delorie.com/djgpp/v2faq/faq.html

Follow the same steps as listed above in **A**, except in

3. Customise makefile make.djg in the qdtoc_c directory.

4. Go to the directory qdtoc_c/src, type

```
make  -f makefile.djp
```

5. Go to the directory qdtoc_c/test, type

   ```
   make  -f makefile.djp
   ```

   This will compile the QDTOC test program,

   ```
   make -f makefile.djp test
   ```

   will solve the test program, and

   ```
   make -f makefile.djp warmstart
   ```

   will solve the test program using the warm start procedure.

3 How to use it

3.1 Basic functions

To use the package QDTOC for solving the Q-DTOC problem, the user needs to

1. create the problem;
2. tell the solver, if you have same number control and state variables in all stages;
3. for each stage, define the objective function, transitions and constraints;
4. define the initial state variable x_1, and the estimation on the optimal control.

In addition, the user can define the initial working set if an active set method is used. The user can also choose the initial multipliers, simple bounds on control variables if approiate. After these definiations, the user can choose one of the available methods to solve the Q-DTOC problem. Once the problem is solved, the user can obtain the solution and then destroy the problem.

All these tasks that the user needs to performed can be easily done by calling some basic functions in QDTOC.

A few data types are needed:

```
MAT             /* meschach matrix */
VEC             /* meschach vector */

qdtoc_prob_t    /* Q-DTOC problem */
qdtoc_meth_t    /* The method to be used for solving a Q-DTOC problem.
                   It can be chosen from
                   {qdtoc_acddp, qdtoc_dacddp, qdtoc_pdacddp, qdtoc_ip}
                   for primal, dual, primal-dual active set method
                   and primal-dual interior point method respectively. */
```

F1. `qdtoc_prob_t *qdtoc_prob_create(int n_stage)`

This function is used to create a Q-DTOC problem that the QDTOC package will solve. This function can be used as follows

```
qdtoc_prob_t *p;
int          N = 5;
   ......
p = qdtoc_prob_create(N);
```

Here p is a pointer to the Q-DTOC problem, N is the number of stages as defined in (1). For example, we have $N = 5$.

F2. `int qdtoc_prob_set_sz_u_same(qdtoc_prob_t *p, int same)`
`int qdtoc_prob_set_sz_x_same(qdtoc_prob_t *p, int same)`

These two functions tell the solver if the control variables $\mathbf{u}_k$ $(k = 1, \cdots, N)$ have the same dimension through all stages; and if the state variables $\mathbf{x}_k$ $(k = 1, \cdots, N + 1)$ have the same dimension through all stages. For example, if **yes** for the control variabls, then pass 1 as the value for the **integer** *same* in the function

int qdtoc_prob_set_sz_u_same(qdtoc_prob_t *p, int same)

as follows

```
qdtoc_prob_set_sz_u_same(p, 1);
```

Otherwise, i.e. if the $\mathbf{u}_k$ do not have the same dimension from stage 1 to stage N, then pass 0 as the value for the **integer** *same* to the same function as follows

```
qdtoc_prob_set_sz_u_same(p, 0);
```

The same applies to the function

int qdtoc_prob_set_sz_x_same(qdtoc_prob_t *p, int same)

F3. `int qdtoc_prob_set_transition(qdtoc_prob_t *p, int stage, MAT *A, MAT *B, VEC *b)`

This function is used to set the transition (1e) for the given stage, where matrices A, B and the vector b are as defined in (1). This function should only be called after the Q-DTOC problem has been created, and the matrices A, B and the vector b have been defined (see § 3.3 for more details). For example, the transition for stage k can be set as follows

```
MAT   *A = MNULL, *B = MNULL;
VEC   *b = VNULL;
int   m = 10, n = 6, k = 10;

      ......

qdtoc_prob_set_transition(p, k, A, B, b);
```

If the transition is set successfully, an non-zero value is returned;

Remark This function takes the copy of the supplied matrices and vector, so it is the responsibility of the caller to free their storage. The same applies to **F3** – **F7**.

F4. `int qdtoc_prob_set_constraint(qdtoc_prob_t *p, int stage, MAT *C, MAT *D, VEC *d)`

This function is used to set the constraints (1d) and (1e) for the given stage. Here the matrices C, D and the vector $\mathbf{d}$ are as defined in (1). This function should only be called after the Q-DTOC problem has been created, and the matrices C, D and the vector $\mathbf{d}$ have been defined (see § 3.3 for more details). For example, the constraint for stage k can be set as follows:

```
qdtoc_prob_set_constraint(p, k, C, D, d);
```

If the constraint is set successfully, an non-zero value is returned;

Remark: For the stage where there is no constraint, zero dimension matrices C, D and the zero dimension vector $\mathbf{d}$ should be passed when this function is called to set constraints for the given stage. For more detail, see **ricker.c**

F5. `int qdtoc_prob_set_objective(qdtoc_prob_t *p, int stage, MAT *Q,`
```
                                  MAT *R, MAT *S, VEC *r, VEC *z)
```

This function is used to set the objective function (2) for the given stage. Here the matrices Q, R, S and vectors $\mathbf{z}, \mathbf{p}$ are as defined in (1). This function should only be called after the Q-DTOC problem has been created, and the matrices P, Q, R and vectors $\mathbf{p}, \mathbf{z}$ have been defined (see § 3.3 for more details). For example, the objective function for stage k can be set as follows:

```
qdtoc_prob_set_objective(p, k, p, Q, R, S, z);
```

If the objective function is set successfully, an non-zero value is returned;

F6. `int qdtoc_prob_set_initial_state(qdtoc_prob_t *p, VEC *x)`

This function is used to set the initial state $\mathbf{x}_1$ as in (1). This function should only be called after the Q-DTOC problem has been created, and the vector $\mathbf{x}$ has been defined (see § 3.3 for more details). For example, the initial state can be set as follows

```
qdtoc_prob_set_initial_state(p, x);
```

If the initial state is set successfully, a non-zero value is returned.
The same responsibility should also be taken by the caller to free up the memory used as it is mentioned in **F3**.

```
int qdtoc_prob_set_control_vars(qdtoc_prob_t *p, int stage, VEC *u)
```

This function is used to set the control variable $\mathbf{u}$ for a given stage. It should only be called after the Q-DTOC problem has been defined, and the vector $\mathbf{u}$ has been defined. The same storage free responsibility, as mentioned in **F3**, should be taken by the caller. For example, for stage k, the control variable can be set as follows

```
qdtoc_prob_set_control_vars(p, k, u);
```

F7. `int qdtoc_prob_solve(qdtoc_prob_t *p, int max_iter, FILE *fp,`
```
                        qdtoc_meth_t meth, int ubd, int resolve,
                        double max_gap)
```

Here

```
int           max_iter;    /* The maximum number of iterations */
FILE          *fp;         /* Pointer to the file to which the results
                              will be writen into */
qdtoc_meth_t meth;         /* The method to be used. It can be chosen
                              from {qdtoc_acddp, qdtoc_dacddp, qdtoc_pdacddp
                              qdtoc_ip} for primal active, dual active,
                              primal-dual active set method and
                              primal-dual interior point method
                              respectively */
int         ubd;          /* If has simple bounds on controls as
                              constraints. Boolean */
int         resolve;      /* If this is a resolve; Boolean */
double      max_gap;      /* The stop criterion for on the duality gap
                              in the primal-dual interior point method */
```

If the Q-DTOC problem is solved successfully, the function returns 1, otherwise 0. For example, if the primal-dual interior point method is to be used for solving a Q-DTOC problem, which has simple bounds on control variabls as constraints only, the Q-DTOC problem can be solved as follows

```
int           max_iter = 40, ubd = 1, resolve = 0;
FILE          *fp = NULL, fp_wkst = NULL;
qdtoc_meth_t method = qdtoc_ip;
double        max_gap = 1e-8;
int           res = 0;

/* Other declarations */

/* Create the problem, set transition, constraints, objective
   functions etc */
   ......

if ((fp = fopen("result.txt", "a")) != NULL)
    res = qdtoc_prob_solve(p, max_iter, fp, method, ubd, resolve,
                           max_gap);
else
    ......
```

F8. `VEC *qdtoc_get_solution_state(qdtoc_prob_t *p, int stage, VEC *x)`
`VEC *qdtoc_get_solution_control(qdtoc_prob_t *p, int stage, VEC *u)`

Once the Q-DTOC problem is solved as in **F7**, these functions can be used to get the solution of the problem, in terms of state and control variables respectively. For example, for the given stage k, the solution can be obtained as follows

```
VEC    *x = VNULL, *u = VNULL;
/* Other decalration */

x = v_get(x, m);         /* x has m elements */
u = v_get(u, n);         /* u has n elements */

/*
// Get the solution for stage k
*/
x = qdtoc_get_solution_state(p, k, x);       /* state in solution */
u = qdtoc_get_solution_control(p, k, u);     /* control in solution */
........
```

F9. `void qdtoc_prob_destroy(qdtoc_prob_t *p)`

This function is used to destroy the problem after it is solved. It is used as follows

```
......
qdtoc_prob_destroy(p);
```

F10. Other functions.

```
int qdtoc_prob_set_ubound(qdtoc_prob_t *p, int stage, VEC *ub, VEC *lb)
int qdtoc_prob_set_control(qdtoc_prob_t *p, int stage, VEC *u)
int qdtoc_prob_set_lambda(qdtoc_prob_t *p, int stage, VEC *lambda)
int qdtoc_prob_set_mu(qdtoc_prob_t *p, int stage, VEC *mu)
```

For the given stage, these functions are used to set simple bounds (lower bound lb, upper bound ub) on controlse, set control vector, and set the multipliers (μ for constraints (1e), λ for transition (1d)).

3.2 Warm/Hot Start

When active set methods are used to solve the Q-DTOC problem (1), if good information about the solution is available, the problem can be solved efficiently. For example,

1. If a good initial point, which is primal feasible, is given, the primal active set method (qdtoc_acddp) can be used to solve the Q-DTOC problem.
2. If a good initial point, which is dual feasible, is given, the dual active set method (qdtoc_dacddp) can be used.
3. If a good initial working set is given, and/or a good initial point, and/or good multipliers are available, the primal-dual active set method (qdtoc_pdacddp) can be used.

By **Warm Start**, we mean that, an initial working set, and/or good initial point and/or multipliers are given, and they are used to start the primal-dual active method for solving the Q-DTOC problem. If any of them is not provided, the dafult value will be used. In this case, an empty initial working set is assumed.

By **Hot Start** we mean that, the initial point, the working set and corresponding matrix factorization that are left from previous solve (by the primal-dual active set method), as well as the multipliers from previous solve are used to start primal-dual active set method. The **Hot Start** procedure is called by passing the parameter *resolve* with value 1 in the function qdtoc_prob_solve.

To use the **Warm Start**, the user is required to provide the available information on the working set, and/or initial point, and/or multipliers. This information is to be provided in the given formats. In the current version, we assume that the initial control values, working set and multipliers are provided by using files.

F1. **The control values**

The file for control value contains $N+1$ lines, where N is the number of stages. On the first N lines, the first element k is the stage index, the second element *size_u* is the size of the control vector for that stage, followed by *size_u* numbers, which are the entry values of the control vector for the given stage. The last line has one integer -1 only.

```
1 size_u u_val ... u_val
                .
                .
                .
N size_u u_val ... u_val
-1
```

An example of the file for control values is given the directory *test*, where the file is called *init_u.dat*.

For the file of control values, it is important to notice that, on each line, the stage index and the size of the control vector for that stage *size_u* **MUST BE** provided correctly. If not, the **solver** may fail. For the rest of each line, if any error is detected, the default value will be used, and the **solver** will assume that the control vector has the same size at each stage.

F2. The working set

The working set is defined by the indices for each stage in the following format:

```
stage cons_idx ... cons_idx -1
                .
                .
                .
-1
```

where *stage* is the stage index, *cons_idx* is the index of the constraint that is in the initial working set. The working set for each stage is terminated by -1, so is the file itself. An example is given in the directory *test*, where the file is called *initwkst.dat*.

For the file of working set, it is important to notice that each line **MUST BE** termined by -1, so is the file itself.

F3. The multiplers λ

The format of the file file for the multipliers λ contains N lines. On each line, the first element is the stage index, followed by the m real values, which are the entry values of the multiplier λ for the given stage. For stage k, m is the number of the states for stage $k+1$.

```
1 lambda_val ... lambad_val
     .
     .
     .
N lmabda_val ... lambda_val
```

It is important to notice that the order of the stage index **MUST BE** from 1 to N. If any other format error is detected, the default value will be used. An example of the file for multipliers λ can be found in the directory *test*, which is called *lambda.dat*.

F4. **The multiplers μ**

The format of the file for the multipliers μ is the same as the format of the file for the multipliers λ as following:

```
1 mu_val ... mu_val
           .
           .
           .
N mu_val ... mu_val
```

On each line, followed the stage index k, there are *size_wkst* nonnegative mumbers, where *size_wkst* is the size of the working set for that stage. It is important to notice that the stage index **MUST BE** from 1 to N. For the stage where the working set is empty, a correct line should also be provided. If any error has been detected, the default value will be used. An example of the file is in the directory *test*, and is called *mu.dat*.

In the main function, these files are opened, and the pointers to these files are passed to the function where the problem is defined. For more details, see the main function in *ricker.c*, which is in the *test* directory.

It is the user's chice on which initial information is to be provided when using warm start. If any of the four information mentioned above is not provided, the default value will be used. A good initial working set is very important to the warm start procedure. Other information maybe less important. If no initial information is provided, we do not expect the warm start to be efficient.

3.3 An Example – The Ricker Problem

3.3.1 Problem Describtion

Consider the **Evaporator** problem (Ricker et al.) [1]. When this probloem is reformulated in the Q-DTOC form as in (1), we have $N = 60$, $m = 11$ and $n = 5$; the initial state $\hat{\mathbf{x}}_1$ is given as described in function ricker() below.

Matrices $A_k, B_k, C_k, D_k, Q_k, R_k, S_k$ and the vectors $\mathbf{b}_k, \mathbf{d}_k, \mathbf{r}_k, \mathbf{z}_k$ are the same for $k = 1, \cdots, N$, and they are defined as follows

$$
\begin{aligned}
A_k &= \begin{bmatrix} A & 0 \\ 0 & 0_{2\times 2} \end{bmatrix}, \; B_k = \begin{bmatrix} B & 0 \\ I_2 & 0 \end{bmatrix} \\
C_k &= \begin{bmatrix} 0 & 0 \\ 0 & -I_2 \\ 0 & 0 \\ 0 & I_2 \\ H \times A & 0 \\ -H \times A & 0 \\ 0 & 0 \end{bmatrix}, \; D_k = \begin{bmatrix} I_2 & 0 \\ I_2 & 0 \\ -I_2 & 0 \\ -I_2 & 0 \\ H \times B & -I_3 \\ -H \times B & I_3 \\ 0 & -I_3 \end{bmatrix} \\
Q_k &= \begin{bmatrix} H^T \times H & 0 \\ 0 & I_2/2 \end{bmatrix}, \; R_k = \begin{bmatrix} I_2/2 & 0 \\ 0 & I_3 \end{bmatrix}, \; S_k = 0 \\
\mathbf{b}_k &= \mathbf{0}, \; \mathbf{d}_k = [u_{hi}, roc, -u_{lo}, roc, h_{hi}, -h_{lo}, \mathbf{0}] \\
\mathbf{z}_k &= \mathbf{0}, \; \mathbf{r}_k = [0, \mathbf{z}]
\end{aligned} \tag{3}
$$

Where I_n is the $n \times n$ identity matrix; $A \in \mathcal{R}^{9\times 9}, B \in \mathcal{R}^{9\times 2}, H \in \mathcal{R}^{3\times 9}$ are given; $\mathbf{z} = [1000, 1000, 1000]$, and $u_{hi} = -u_{lo} = 0.2, h_{hi} = -h_{lo} = roc = 0.05$.

For stage $N + 1$, there is no constraint; for the objective function, $\mathbf{z}_{N+1} = \mathbf{0}$ and

$$
Q_{N+1} = \begin{bmatrix} \tilde{Q} & 0 \\ 0 & I_2/2 \end{bmatrix}
$$

where $\tilde{Q} \in \mathcal{R}^{9\times 9}$ is given.

3.3.2 Function ricker

To solve the problem described in § 3.3.1, a function, called ricker (see below), is written to define the problem in the form that QDTOC package requires. Within this function, some **Meschach** functions are used; details on how to set matrices and vectors in (3) are omitted, and can be found in the package itself.

```
/*---------------------------------------------------------------------
ricker ()
-----------------------------------------------------------------------
Use problem data that are read in from files to initialise a qdtoc_prob_t.
----------------------------------------------------------------------*/
```

```
qdtoc_prob_t *ricker(int max_iter, FILE *fp, FILE *fp_wkst, FILE *fp_lambda,
                   FILE *fp_mu, FILE *fp_u)
{
    qdtoc_prob_t *p;

    int    res = 0, i = 0, k = 0;
    int    ubd = 0, resolve = 0;
    int    N, m, n, l, m_true, np;
    int    control_is_defined = 0;

    double u_time;
    double ubound= 0.2, ybound = 0.05, roc = 0.05;

    VEC    *d = VNULL, *b = VNULL, *r = VNULL;
    VEC    *z = VNULL, *x1 = VNULL, *u = VNULL, *x = VNULL;
    MAT    *C = MNULL, *D = MNULL, *Q = MNULL, *R = MNULL, *S = MNULL;
    MAT    *B = MNULL, *A = MNULL, *H = MNULL;
    MAT    *Ak = MNULL, *Bk = MNULL, *Ck = MNULL, *Dk = MNULL;
    MAT    *HA = MNULL, *HB = MNULL, *bar_Q = MNULL;
    FILE   *fp_A = NULL, *fp_B = NULL, *fp_H = NULL, *fp_Q = NULL;

    N = 60; m = 11; n = 2;
    m_true = 9; np = 3;
    l = 4 * n + 3 * np;

    /*
    // Write some info about the problem into the given file
    */
    fprintf(fp, "%4d%2c%4d%2c%4d%2c%4d%2c", N, '&', m, '&', n + np,'&',  l,'&');

    /*
    // allocate matrices and vectors
    */
    C = m_get(l, m); D = m_get(l, n); d = v_get(l);
    Q = m_get(m, m); R = m_get(np + n, m); S = m_get(np + n, np + n);
    B = m_get(m, n); A = m_get(m, m);
    H = m_get(m, m); HA = m_get(m, m); HB = m_get(m, m);
    Ak = m_get(m, m); Bk = m_get(m, n + np);
    Ck = m_get(l, m); Dk = m_get(l, n + np);
    bar_Q = m_get(m, m);
```

```
z = v_get(m); r = v_get(n + np);
u = v_get(n); x = v_get(m);
b = v_get(m); d = v_get(l);

/*
// Read in A;
*/
A =  m_resize(A, m_true, m_true);
fp_A = fopen("A.dat", "r");
A = read_matrix(fp_A, A);
fclose(fp_A);

/*
// Read in B;
*/
B = m_resize(B, m_true, n);
fp_B = fopen("B.dat", "r");
B = read_matrix(fp_B, B);
fclose(fp_B);

/*
// Read in H(3, 9)
*/
H = m_resize(H, np, m_true);
fp_H = fopen("H.dat", "r");
H = read_matrix(fp_H, H);
fclose(fp_H);

/*
// Calculate HA = H * A, HB = H * B
*/

HA = m_resize(HA, np, m_true);
HA = m_mlt(H, A, HA);

HB = m_resize(HB, np, n);
HB = m_mlt(H, B, HB);

/*
```

```
// Create the QDTOC problem
*/
p = qdtoc_prob_create(N);

/*
// u and x have the same dimension through all stages
*/
qdtoc_prob_set_sz_u_same(p, 1);
qdtoc_prob_set_sz_x_same(p, 1);

/*
// Set the initial state x1 = [x1; 0]
*/
x1 = v_get(m);

/*
// Initialise x1 as a zero vector
*/
x1 = v_zero(x1);

/*
// Set x1(1:m_true)
*/
v_set_val(x1, 0, 9.5564e-01);
v_set_val(x1, 1, 1.0130e-01);
v_set_val(x1, 2, -4.6596e-02);
v_set_val(x1, 3, -1.4930e+00);
v_set_val(x1, 4, -4.2337e+00);
v_set_val(x1, 5, 5.7746e-01);
v_set_val(x1, 6, 2.9066e+00);
v_set_val(x1, 7, 5.0683e-01);
v_set_val(x1, 8, 1.0589e+00);

/*
// Set the initial state
*/
qdtoc_prob_set_initial_state(p, x1);

/*
// For stage 1 to stage N, set transitions, constraints,
// objective functions and controls
*/
for (i = 1; i <= N; ++i) {
    /*
    // Ak, Bk, and bk are the same through all stages
    // as defined in (3)
    */
    if (i == 1) {
        /*
        // set Ak and Bk
        */
        Ak = ricker_set_A(A, m, Ak);
        Bk = ricker_set_B(B, m, n, Bk);

        /*
        // bk = 0
        */
        b = v_resize(b, m);
        b = v_zero(b);
    }

    /*
    // Set transition
    */

    qdtoc_prob_set_transition(p, i, Ak, Bk, b);

    /*
    // Set u with correct size and initialise u as init_u
    // unless wish set as other value
    */
    u = v_resize(u, n + np);
    if (fp_u != NULL) {
        /*
        // set control values with the function
        // qdtoc_set_control_from_file()
        // after the problem is defined. Unless
        // the user want to specify the values
        // differently. e.g. as in the else block
        // below
        */
    } else {
        /*
        // set u = 0
        */
        u = v_zero(u);
        qdtoc_prob_set_control_vars(p, i, u);
        control_is_defined = 1;
    }

    /*
    // Set constraints Ck x_i + Dk u_i <= dk
    // with Ck, Dk and dk being defined in (3)
    // Note that Ck, Dk and dk are the same through all
    // stages
    */

    if (i == 1) {
        /*
        // Set Ck, Dk and d
        */
        Ck = ricker_set_C(HA, m, n, np, l, Ck);
        Dk = ricker_set_D(HB, m, n, np, l, Dk);
        d = ricker_set_d(ubound, roc, ybound, n, np, l, d);
    }

    /*
    // Set constraints
    */
    qdtoc_prob_set_constraint(p, i, Ck, Dk, d);

    /*
    // Set objective function
    // Qk, Sk, Rk and zk, rk are defined in (3)
    // Note that they are the same through all stages
    */

    if (i == 1) {
        /*
        // Set Q as Qk
```

```
            */
            Q = ricker_set_Q(H, m, m_true, Q);

            /*
            // Set S as Sk
            */
            S = ricker_set_S(n, np, S);

            /*
            // Set R as Rk
            */
            R = ricker_set_R(n, np, m, R);

            /*
            // set z as zk
            */
            z = ricker_set_z(m, z);

            /*
            // set r as rk
            */
            r = ricker_set_r(n, np, r);
        }

        /*
        // Set the objective function
        */
        qdtoc_prob_set_objective(p, i, Q, R, S, r, z);

    } /*End of for*/

    /*
    // For stage N + 1
    */

    /*
    // No constraints for stage N + 1
    */

    /*
```

```
// Set Ck, Dk and d as zero dimension arrays
*/
Ck = m_resize(Ck, 0, 0);
d = v_resize(d, 0);
Dk = m_resize(Dk, 0, 0);

/*
// Set constraints stage N + 1
*/
qdtoc_prob_set_constraint(p, N + 1, Ck, Dk, d);

/*
// Objective function for stage N + 1 as described in 3.2.1
*/

/*
// Read in bar_Q
*/
bar_Q = m_resize(bar_Q, m_true, m_true);
fp_Q = fopen("bar_Q.dat", "r");
bar_Q = read_matrix(fp_Q, bar_Q);
fclose(fp_Q);

/*
// Set Q with correct correct size m,
*/
Q = m_resize(Q, m, m);

/*
// Set Q for stage N + 1
*/
Q = ricker_set_Q_N1(bar_Q, m, m_true, Q);

/*
// Rk, Sk, rk do not appear in the objective function
// set them as zero dimension arrays
*/
R = m_resize(R, 0, 0);
r = v_resize(r, 0);
```

```
/*
// Set z as z_{N+1}
*/
z = ricker_set_z(m, z);

/*
// Set the objective function for stage N + 1
*/
qdtoc_prob_set_objective(p, N + 1, Q, R, R, z, z);

/*
// set control values from the file if wanted
*/
if (fp_u) {
    rewind(fp_u);
    if (qdtoc_set_control_from_file(p, fp_u) == 0)
        log_error("Failed at set control values from file");
    rewind(fp_u);
}
/*
// Tell the solver if control is defined
*/
qdtoc_prob_set_control_defined(p, control_is_defined);

/*
// Solve this problem
*/

ubd = 0;            /* not a problem with simple bounds on u only */
resolve = 0;        /* not for resolve */

start_time();       /* start time for solving the problem */

/*
// Solve the problem with the give method (See F7)
*/
if (warmstart)
    res = qdtoc_warm_start(p, max_iter, fp, resolve, fp_wkst, fp_lambda,
                           fp_mu, fp_u);
else
```

```
    res = qdtoc_prob_solve(p, max_iter, fp, method, ubd, resolve, MAX_GAP);

u_time = prn_time(); /* User time for solving the problem */

/*
// Print the user time into the given file
*/
fprintf(fp, "  %.4f%2c", u_time, '&');

/*
// Get solution
*/

/*
// Get controls u for stage 1 to stage N
*/
for (i = 1; i <= N; ++i) {
    /*
    // Set u with the correct size
    */
    u = v_resize(u, n + np);

    /*
    // Get the predicted optimal controls
    */
    u = qdtoc_get_solution_control(p, i, u);
}

/*
// Get states x for stage 1 to stage N
*/

for (i = 1; i <= N; ++i) {

    /*
    // Set x with correct size
    */
    x = v_resize(x, m);

    /*
```

```
        // Get the state part of the solution
        */
        x = qdtoc_get_solution_state(p, i, x);
    }

    /*
    // Destroy the problem
    */
    qdtoc_prob_destroy(p);

/*----------------------------------------------------------------------
    Cleanup
---------------------------------------------------------------------- */
    M_FREE(C); M_FREE(D); M_FREE(Q); M_FREE(R); M_FREE(S);
    V_FREE(d); V_FREE(x1); V_FREE(u); V_FREE(z); V_FREE(x);
    V_FREE(b); V_FREE(r);
    M_FREE(Ak); M_FREE(Bk); M_FREE(Ck); M_FREE(Dk);
    M_FREE(A); M_FREE(B);
    M_FREE(H); M_FREE(HA); M_FREE(HB); M_FREE(bar_Q);

    return p;
}
```

A **main** function is needed to use the function **ricker**. An example, is given in the module **ricker.c** in the directory

```
    qdtoc_c/test
```

In this directory, when run

```
    make test
```

the problem will be solved, and the result is written into the file *result.txt*, some log message is written into the file *ricker.log*. By changing the value of *diag_mask* (e.g. 0x1, 0x2, 0xf etc., you can control the amount of messages written into the log file. For more details, see *ricker.c*). With the current *diag_mask* = 0x0, the minimum amount information about the problem and the solver is written into the log file *ricker.log*.

If the **Warm Start** procedure is to be used, the problem can be solved by the primal-dual active set method qdtoc_pdacddp. Instead of running *make test* as mentioned above, run

```
        make warmstart
```

the summary of the result will be written into the file *result.txt*. For this example, examples of the initial working set, initial control values, multipliers λ and μ are in the files

```
        initwkst.dat, init_u.dat, lambda.dat, mu.dat
```

respectively.

4 Acknowledgements

References

[1] S. J. Wright C. R. Rao and J. B. Rawlings. Application of interior-point methods to model predictive control. Technical report, Preprint ANL/MCS-P664-0597, May 1997.

[2] D. Ralph and Y. Xiao. Efficient stagewise methods in the application to model predictive control. Technical report, The university of Melbourne, in prapertion 1998.

[3] D. Ralph and Y. Xiao. Stagewise methods for q-dtoc problem: Algorithms, implementation and applications. Technical report, The university of Melbourne, in prapertion 1998.

[4] D. E. Stewwart and Z. Leyk. *Meschach: Matrix Computations in C.* Proceeding of the Centre for Mathematics and its Applications, The Australian National University, Auatralia, 1994.