2017 21st International Conference on Process Control (PC)
June 6–9, 2017, Štrbské Pleso, Slovakia

C Code Generation Applied to Nonlinear Model Predictive Control for an Artificial Pancreas*

Dimitri Boiroux[1,2] and John Bagterp Jørgensen[1]

***Abstract*— This paper presents a method to generate C code from MATLAB code applied to a nonlinear model predictive control (NMPC) algorithm. The C code generation uses the MATLAB Coder Toolbox. It can drastically reduce the time required for development compared to a manual porting of code from MATLAB to C, while ensuring a reliable and fairly optimized code. We present an application of code generation to the numerical solution of nonlinear optimal control problems (OCP). The OCP uses a sequential quadratic programming algorithm with multiple shooting and sensitivity computation. We consider the problem of glucose regulation for people with type 1 diabetes as a case study. The average computation time when using generated C code is 0.21 s (MATLAB: 1.5 s), and the maximum computation time when using generated C code is 0.97 s (MATLAB: 5.7 s). Compared to the MATLAB implementation, generated C code can run in average more than 7 times faster.**

I. INTRODUCTION

In optimal control, the underlying optimization problem is usually constrained by the dynamics of a system described by nonlinear ordinary differential equations (ODEs), differential algebraic equations (DAEs), or even partial differential equations (PDEs). Nonlinear model predictive control (NMPC) is a receding horizon control technology that repeatedly solves open-loop nonlinear optimal control problems (OCPs). At every sample, it implements the computed optimal input associated to the current time period. Research groups are investigating a large number of potential applications of NMPC, spanning from real-time applications that need to be implemented on an embedded device where the OCP has to be solved within milliseconds [1]–[3], to very large-scale applications requiring high performance computing, for instance for oil recovery [4]–[6]. Numerous software tools using code generation for solving nonlinear OCPs have been considered. The ACADO toolkit is an open-source software for control and nonlinear optimization [7]. CasADI is a symbolic package capable of C code generation [8]. JModelica.org is a tool for large-scale dynamic optimization problems [9]. CVXGEN generates a custom C code to solve convex optimization problems [10].

MATLAB provides a user-friendly and simple environment useful for prototyping and software development. However, its code usually runs slower than compiled code and cannot be ported to all architectures. Conversely, C is a programming language used in a large number of applications, but the development of optimized and reliable C code is a time-consuming task.

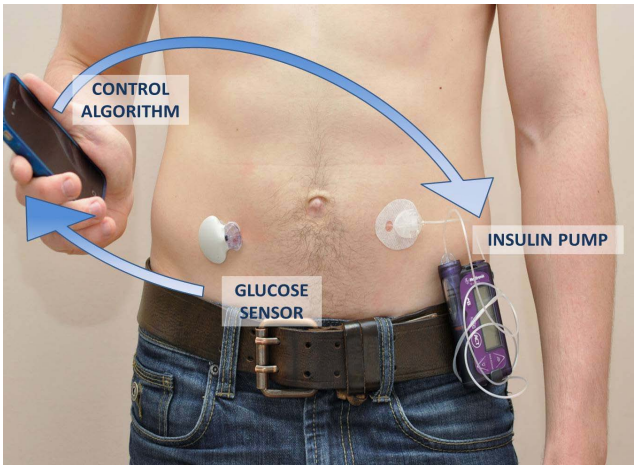

Fig. 1. The artificial pancreas.

In this paper, we present a new method for solving nonlinear OCPs arising in NMPC using the MATLAB Coder toolbox. The MATLAB Coder toolbox does not require any knowledge in C programming and only requires minor adaptations of the MATLAB code. The generated code is tailored to a specific OCP, such that the routines and the memory allocation are optimized. We provide comprehensive guidelines to generate a C code from MATLAB using the MATLAB Coder toolbox.

We present the problem of closed-loop control of blood glucose in people with type 1 diabetes, also referred as the artificial pancreas (AP), as a case study. The AP comprises a glucose sensor measuring glucose levels frequently, a control algorithm, and an insulin pump. It has the potential to improve the quality of life and reduce the burden of insulin therapy management. MPC and NMPC-based control algorithms are among the most popular for the design of the AP [11]–[14]. For this application, the control algorithm has to be implemented on a mobile platform, such as a smartphone, or even on a chip. Fig. 1 illustrates the AP.

The paper is structured as follows. Section II states the continuous-time OCP and presents a numerically tractable discrete-time approximation. Section III outlines the sequential quadratic program (SQP) algorithm. Section IV demonstrates the capabilities of the MATLAB Coder toolbox

* This work has been funded by the Danish Diabetes Academy supported by the Novo Nordisk Foundation.

[1]Dimitri Boiroux and John Bagterp Jørgensen are with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark {dibo, jbjo}@dtu.dk

[2]Dimitri Boiroux is with the Danish Diabetes Academy, DK-5000 Odense C, Denmark

and provides the required steps to obtained a compiled C code from MATLAB. Section V presents the application to the diabetes problem and highlights the difference in running time between MATLAB and the generated C code. A summary of the main contributions of the paper is provided in Section VI.

II. PROBLEM FORMULATION

We consider the bound constrained continuous-time OCP

$$\min_{[x(t),u(t)]_{t_0}^{t_f}} \quad \phi = \int_{t_0}^{t_f} g(x(t),u(t))dt + h(x(t_f)), \tag{1a}$$

$$\text{s.t.} \quad x(t_0) = x_0, \tag{1b}$$

$$\dot{x}(t) = f(x(t),u(t),d(t)), \quad t \in [t_0, t_f], \tag{1c}$$

$$u_{\min} \le u(t) \le u_{\max}, \quad t \in [t_0, t_f], \tag{1d}$$

where $x(t) \in \mathbf{R}^{n_x}$ is the state vector, $u(t) \in \mathbf{R}^{n_u}$ are the manipulated inputs, and $d(t) \in \mathbf{R}^{n_d}$ are known disturbances. $\dot{x}(t) = f(x(t),u(t),d(t))$ represents the model equations. The initial time, t_0, and the final time, t_f, are fixed parameters. The initial state, x_0, is a known parameter in (1). The inputs are bound constrained and must be in the interval $u(t) \in [u_{\min},\, u_{\max}]$. The objective function is stated with a stage cost term, $g(x(t),u(t))$, and a cost-to-go term, $h(x(t_f))$.

A. Zero-order hold parametrization

In general, the continuous-time bound constrained problem (1) is not tractable and is solved numerically by discretization using a zero-order hold (ZOH) parametrization of the manipulated variables, $u(t)$, and the known disturbance variables, $d(t)$. We sample the time interval, $[t_0,\, t_f]$, into N equidistant intervals each of length T_s. Let $\mathcal{N} = \{0, 1, ..., N-1\}$ and $t_k = t_0 + kT_s$ for $k \in \mathcal{N}$. The ZOH parametrization on $u(t)$ and $d(t)$ yields

$$u(t) = u_k, \qquad t_k \le t < t_{k+1}, \qquad k \in \mathcal{N}, \tag{2a}$$

$$d(t) = d_k, \qquad t_k \le t < t_{k+1}, \qquad k \in \mathcal{N}. \tag{2b}$$

Using this ZOH restriction on the inputs, the bound constrained continuous-time Bolza problem (1) may be expressed as

$$\min_{\{x_{k+1},u_k\}_{k=0}^{N-1}} \phi = \sum_{k=0}^{N-1} G_k(x_k,u_k,d_k) + h(x_N), \tag{3a}$$

$$\text{s.t.} \quad b_k := F_k(x_k,u_k,d_k) - x_{k+1} = 0, \; k \in \mathcal{N}, \tag{3b}$$

$$u_{\min} \le u_k \le u_{\max}, \quad k \in \mathcal{N}. \tag{3c}$$

The discrete-time state transition function is

$$F_k(x_k,u_k,d_k) = \{x(t_{k+1}) : \dot{x}(t) = f(x(t),u_k,d_k), \; x(t_k) = x_k\}, \tag{4}$$

and the discrete time stage cost is

$$G_k(x_k,u_k,d_k) = \left\{ \int_{t_k}^{t_{k+1}} g(x(t),u_k)dt : \dot{x}(t) = f(x(t),u_k,d_k),\, x(t_k) = x_k \right\}. \tag{5}$$

III. THE SEQUENTIAL QUADRATIC PROGRAM ALGORITHM

In this section, we describe a multiple-shooting based SQP algorithm [15]–[18]. The SQP algorithm is used for the numerical solution of (1). The quadratic sub-problems arising in the SQP algorithm are efficiently solved using Riccati iterations [19]–[22]. We use a fourth order Runge-Kutta scheme with fixed stepsize for numerical solution of the differential equation model and for computation of the sensitivities.

A. SQP algorithm

We define the parameter vector, p, as

$$p = \begin{bmatrix} u_0' & x_1' & u_1' & x_2' & \dots & x_{N-1}' & u_{N-1}' & x_N' \end{bmatrix}', \tag{6}$$

and the disturbance vector as

$$d = \begin{bmatrix} d_0' & d_1' & \dots & d_{N-1}' \end{bmatrix}'. \tag{7}$$

We can formulate the discrete-time dynamics as

$$b(p) = b(p,x_0,d) = \begin{bmatrix} F_0(x_0,u_0,d_0) - x_1 \\ F_1(x_1,u_1,d_1) - x_2 \\ \vdots \\ F_{N-1}(x_{N-1},u_{N-1},d_{N-1}) - x_N \end{bmatrix}. \tag{8}$$

The objective function is

$$\phi(p) = \phi(p,x_0,d) = \sum_{k=0}^{N-1} G_k(x_k,u_{k,d}) + h(x_N). \tag{9}$$

Let $c(p)$ denote the bound constraints, i.e.

$$c(p) = \begin{bmatrix} u_0 - u_{\min} \\ u_1 - u_{\min} \\ \vdots \\ u_{N-1} - u_{\min} \\ u_{\max} - u_0 \\ u_{\max} - u_1 \\ \vdots \\ u_{\max} - u_{N-1} \end{bmatrix}. \tag{10}$$

Using these notations, we can reformulate the discrete-time Bolza problem (3) as a constrained optimization problem in standard form

$$\min_p \quad \phi = \phi(p), \tag{11a}$$

$$\text{s.t.} \quad b(p) = 0, \tag{11b}$$

$$c(p) \ge 0. \tag{11c}$$

The Lagrangian of (11) is

$$\mathcal{L}(p,y,z) = \phi(p) - y'b(p) - z'c(p). \tag{12}$$

The nonlinear problem (11) can be solved iteratively using a SQP algorithm. In each iteration, (11) is locally approximated by the quadratic program (QP)

$$\min_{\Delta p} \quad \frac{1}{2}\Delta p' W^k \Delta p + \nabla_p \phi'(p^k)\Delta p, \tag{13a}$$

$$\text{s.t.} \quad \left[\nabla_p b(p^k)\right]' \Delta p = -b(p^k), \tag{13b}$$

$$\left[\nabla_p c(p^k)\right]' \Delta p \geq -c(p^k), \tag{13c}$$

where W^k is an approximation of the Hessian of the Lagrangian. It is obtained by the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [23].

The first order Karush-Kuhn-Tucker (KKT) conditions of the constrained nonlinear optimization problem (11) are

$$\nabla_p \mathcal{L}(p, y, z) = \nabla_p \phi(p) - \nabla_p b(p) y - \nabla_p c(p) z = 0, \tag{14a}$$

$$b(p) = 0, \tag{14b}$$

$$c(p) \geq 0, \tag{14c}$$

$$z \geq 0, \tag{14d}$$

$$c_i(p) = 0 \vee z_i = 0 \quad \forall i. \tag{14e}$$

These conditions are used to test the convergence of the SQP algorithm.

B. Gradient computation

The most time consuming computations in the SQP algorithm are the computation of the objective function $\phi(p)$, the derivatives of the objective function $\nabla_p \phi(p)$, the dynamics $b(p)$, and the sensitivities, $\nabla_p b(p)$. $b(p)$ and $\phi(p)$ are computed by evaluation of (4) and (5), respectively. Consequently, the equality constraints and their sensitivities with respect to states and inputs are

$$b_k = F_k(x_k, u_k, d_k) - x_{k+1}, \tag{15a}$$

$$\nabla_{x_k} b_k = \nabla_{x_k} F_k(x_k, u_k, d_k) = S_{x_k}(t_{k+1})' = A_k', \tag{15b}$$

$$\nabla_{u_k} b_k = \nabla_{u_k} F_k(x_k, u_k, d_k) = S_{u_k}(t_{k+1})' = B_k', \tag{15c}$$

$$\nabla_{x_{k+1}} b_k = -I, \tag{15d}$$

where $x(t_{k+1}) = F(x_k, u_k, d_k)$, and the sensitivities (15b-(15c)) follow the systems of ordinary differential equations

$$\dot{x}(t) = f(x(t), u_k, d_k), \tag{16a}$$

$$\dot{S}_{x_k}(t) = \left(\frac{\partial f}{\partial x}(x(t), u_k, d_k)\right) S_{x_k}(t), \tag{16b}$$

$$\dot{S}_{u_k}(t) = \left(\frac{\partial f}{\partial x}(x(t), u_k, d_k)\right) S_{u_k}(t) + \left(\frac{\partial f}{\partial u}(x(t), u_k, d_k)\right), \tag{16c}$$

with the initial conditions $x(t_k) = x_k$, $S_{x_k}(t_k) = I$, and $S_{u_k}(t_k) = 0$. The stage cost and the associated derivatives are computed as

$$G_k = G_k(x_k, u_k, d_k) = \int_{t_k}^{t_{k+1}} g(x(t), u_k, d_k) dt, \tag{17a}$$

$$q_k = \nabla_{x_k} G_k = \int_{t_k}^{t_{k+1}} \left(\frac{\partial g}{\partial x}(x(t), u_k, d_k)\right) S_{x_k}(t) dt, \tag{17b}$$

$$r_k = \nabla_{u_k} G_k = \int_{t_k}^{t_{k+1}} \left[\left(\frac{\partial g}{\partial x}(x(t), u_k, d_k)\right) S_{u_k}(t) + \left(\frac{\partial g}{\partial u}(x(t), u_k, d_k)\right)\right] dt. \tag{17c}$$

The derivatives $\nabla_{x_k} b_k$ and $\nabla_{x_k} G_k$ are computed for $\{x_k\}_{k=1}^{N-1}$ and $k \in \mathcal{N}$. These derivatives are not computed for x_0 as $x_0 \notin p$, i.e. x_0 is a fixed parameter of the optimization problem but not a decision variable. The derivatives $\nabla_{u_k} b_k$ and $\nabla_{u_k} G_k$ are computed for $k \in \mathcal{N}$. The derivatives with respect to x_N are

$$\nabla_{x_N} b_{N-1} = -I, \tag{18a}$$

$$p_N = \nabla_{x_N} \phi = \nabla_{x_N} h(x_N). \tag{18b}$$

Therefore, the gradients of the equality constraints b_k with respect to the parameter vector p can be written as

$$\nabla_p b = \begin{bmatrix} B_0 & & & & & \\ -I & A_1 & & & & \\ & B_1 & & & & \\ & -I & A_2 & & & \\ & & B_2 & & & \\ & & -I & & & \\ & & & \ddots & & \\ & & & & A_{N-1} & \\ & & & & B_{N-1} & \\ & & & & -I & \end{bmatrix}. \tag{19}$$

C. Numerical integration

In this paper, we use the classical explicit Runge Kutta solver of order 4 to numerically compute the system state transition (4), the stage cost (5), the sensitivities (15b-15c), as well as the stage cost derivatives (17b-17c). Given the state values x_n at time t_n, the solution at the next step x_{n+1} is given by

$$X_1 = x_n, \tag{20a}$$

$$X_2 = x_n + \frac{1}{2} h f(t_1, X_1), \tag{20b}$$

$$X_3 = x_n + \frac{1}{2} h f(t_2, X_2), \tag{20c}$$

$$X_4 = x_n + h f(t_3, X_3), \tag{20d}$$

$$x_{n+1} = x_n + \frac{1}{3} h \left(\frac{1}{2} f_1 + f_2 + f_3 + \frac{1}{2} f_4\right), \tag{20e}$$

where $h = t_{n+1} - t_n$ is a fixed step size. The times where internal stages are computed are $t_1 = t_n$, $t_2 = t_3 = t_n + \frac{1}{2}h$ and $t_4 = t_n + h = t_{n+1}$. Let $f_i = f(t_i, X_i)$, $i = 1, 2, 3, 4$, be the function evaluations at times t_i.

D. Interior point algorithm

We use a structured primal-dual interior point algorithm for the solution of the constrained QP (13). We implement the centering step correction proposed by Mehrotra [24]. We use a Riccati recursion to compute the Newton iterations in the primal-dual interior point algorithm [19], [20], [25] . This factorization can be used to compute the optimal variation in the manipulated variables Δu_k, the optimal change in states variables Δx_{k+1}, and the Lagrange multipliers y_{k-1}. For most problems, Riccati recursion-based solvers are considered as the most computationally efficient method to solve the linear quadratic (LQ) sub-problem arising in the interior point method [26].

IV. C CODE GENERATION

Since version R2011a, MATLAB contains a toolbox for C code generation [27], [28]. Before that date, several code generation tools in C or Fortran have been developed for control applications [29], [30]. The C code generation creates either a MATLAB executable (mex) version of the code, a dynamic library (dll), a stand-alone C library or an executable. The C code can be ported to a number of platforms, such as embedded systems, smartphones, or can be used for high performance computing. The source code is also editable and can for instance be integrated in an existing code. The C code generation consists of the four following steps

1) Implement the SQP-based control algorithm as described in Section III.
2) Include the model and the sensitivity functions also detailed in Section III.
3) Generate the C code from MATLAB command line or via the dedicated graphical user interface.
4) Compile the C code.

We configured the coder for further optimization by deactivating the responsiveness to CTRL+C and graphics refreshing, removing runtime checks and disabling dynamic runtime memory sizing, since these features are not required in our case [31, Section 8.2.4].

A. Example of generated C code

Listing 1 shows an example of generated C code for the matrix-matrix multiplication $C = AB$, assuming that A and B are 5×5 matrices. Setting the type (double precision floating point numbers) and the sizes of A and B (5 times 5) is done by adding the following four lines

```
assert(isa(A,'double'));
assert(isa(B,'double'));
assert(all(size(A)==[5 5]));
assert(all(size(B)==[5 5]));
```

at the beginning of the MATLAB function. In this case, the C code generator chooses to use nested for loops.

Listing 2 shows an example of generated C code for the matrix-matrix multiplication $C = AB$, assuming that A and B are 1000×1000 matrices. Similarly, we declare the types and sizes of A and B in the MATLAB function by adding the lines

Listing 1. Small-scale matrix-matrix multiplication

```
void matmult(const real_T A[25],const real_T
    B[25],real_T C[25])
{
  int32_T i0;
  int32_T i1;
  int32_T i2;

  for (i0 = 0; i0 < 5; i0++) {
    for (i1 = 0; i1 < 5; i1++) {
      C[i0 + 5 * i1] = 0.0;
      for (i2 = 0; i2 < 5; i2++) {
        C[i0+5*i1]+=A[i0+5*i2]*B[i2+5*i1];
      }
    }
  }
}
```

Listing 2. Large-scale matrix-matrix multiplication

```
void matmult(const real_T A[1000000], const
    real_T B[1000000], real_T C[1000000])
{
  real_T alpha1;
  real_T beta1;
  char_T TRANSB;
  char_T TRANSA;
  ptrdiff_t m_t;
  ptrdiff_t n_t;
  ptrdiff_t k_t;
  ptrdiff_t lda_t;
  ptrdiff_t ldb_t;
  ptrdiff_t ldc_t;

  alpha1 = 1.0;
  beta1 = 0.0;
  TRANSB = 'N';
  TRANSA = 'N';
  memset(&C[0],0,1000000U*sizeof(real_T));
  m_t = (ptrdiff_t)1000;
  n_t = (ptrdiff_t)1000;
  k_t = (ptrdiff_t)1000;
  lda_t = (ptrdiff_t)1000;
  ldb_t = (ptrdiff_t)1000;
  ldc_t = (ptrdiff_t)1000;
  dgemm(&TRANSA, &TRANSB, &m_t, &n_t, &k_t, &
      alpha1, &A[0], &lda_t, &B[0],
        &ldb_t, &beta1, &C[0], &ldc_t);
}
```

```
assert(isa(A,'double'));
assert(isa(B,'double'));
assert(all(size(A)==[1000 1000]));
assert(all(size(B)==[1000 1000]));
```

at the beginning of the MATLAB function. Since it is now a large-scale problem, the BLAS level 3 routine `dgemm` is more efficient to perform the matrix-matrix multiplication, and therefore is preferred to the nested for loops.

Since the generated code is tailored for the specific OCP to solve, the MATLAB Coder Toolbox optimizes the choice of the routines and the memory usage to a specific problem.

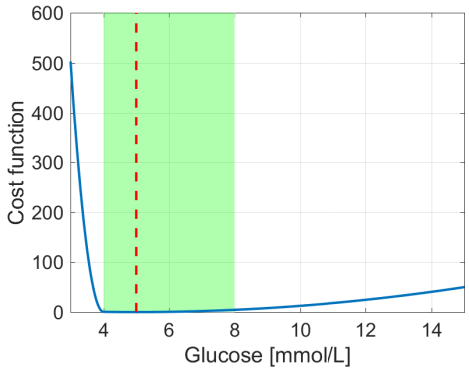

Fig. 2. The asymmetric cost function.

V. NUMERICAL RESULTS

The simulations are performed using MATLAB R2016a installed on a Dell Latitude E6540 (Intel Core i7-4800MQ processor, 16 Gb RAM). We use the compiler `gcc` version 6.2.0 on Linux Mint 17.3 to compile the generated C code.

The sampling time is $T_s = 5$ minutes. We use a continuous-discrete extended Kalman filter for state estimation at each sample [32], [33].

The objective of the insulin administration is to compensate glucose excursions caused by meals and variations in endogenous glucose production and utilization. We use a penalty function defined as

$$g(G(t)) = \frac{1}{2}\left(G(t) - \bar{G}\right)^2 + \frac{\kappa}{2}\max\left\{0, G_L - G(t)\right\}^2. \tag{21}$$

$G(t)$ is the blood glucose concentration, $\bar{G} = 5$ mmol/L is the target value for the blood glucose concentration, $G_L = 4$ mmol/L is a lower acceptable limit on the glucose concentration. The weight κ is used to heavily penalize hypoglycemia. Fig. 2 illustrates the penalty function used in the simulations.

The objective function used in the simulations is

$$\tilde{\phi} = \int_{t_0}^{t_f} g(x(t), u(t))dt + \frac{\eta}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_2^2, \tag{22}$$

where $\Delta u_k = u_k - u_{k-1}$. This objective function has no cost-to-go function, i.e. $h(x(t_f)) = 0$, and contains a regularization term, $\frac{\eta}{2}\sum_{k=0}^{N-1}\|\Delta u_k\|_2^2$. The objective function (22) can be brought into the standard form (3a) using state augmentation formulated by [34].

A. Simulation results

We use the Medtronic Virtual Patient (MVP) model [35] and the developed multiple shooting SQP algorithm for (1) to compute the optimal insulin administration profiles for people with type 1 diabetes. We run a MATLAB and a generated C version of the multiple shooting SQP algorithm.

Fig. 3 shows the glucose, insulin and meal profiles, as well as the CPU time for each time sample. We assume that the patient has three meals: a 75g carbohydrates (CHO) breakfast at 6AM, a 100g CHO lunch at 12PM and a 75g CHO dinner at 6PM. The meals are not anticipated, ie. they are announced to the MPC only at mealtimes. The average computation time when using generated C code is 0.21 s, versus 1.5 s for the MATLAB code, and the maximum computation time when using generated C code is 0.97 s, versus 5.7 s for the MATLAB code. Thus, the generated C code provides in average a speedup of more than 7 times compared to the MATLAB implementation.

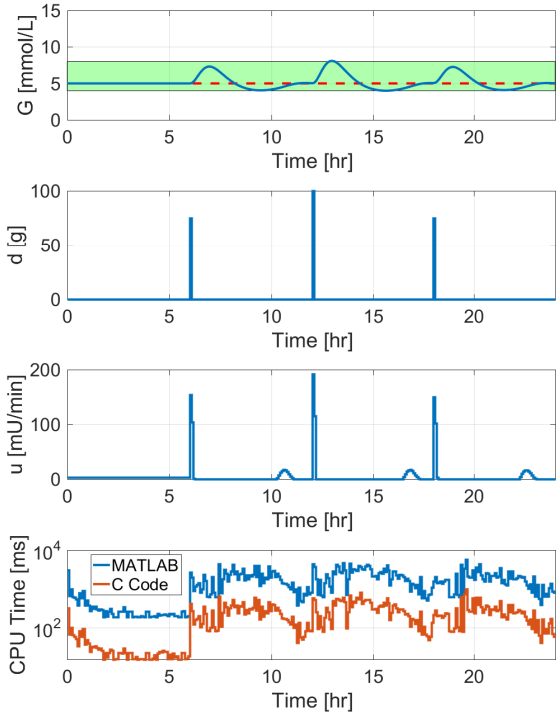

Fig. 3. First panel (top): Blood glucose trace. Second panel: Meals. Third panel: Insulin infusion rate trace. Fourth panel (bottom): CPU time for MATLAB and C code.

B. Discussion

MATLAB uses a number of optimized built-in functions, so it is not guaranteed that the generated code will provide any speedup if the initial code heavily relies on these built-in functions. In our OCP, the numerical integration routine for computation of the model dynamics, the objective function and the sensitivities represent the main computational workload in the local SQP algorithm, which accounts for approximately 70% of the total CPU time. Since it has a large number of function evaluations, it will benefit the most from code generation. Conversely, the interior point algorithm represents uses almost all the remaining CPU time.

The purpose of this paper is to compare the MATLAB implementation with the generated C code. Further optimization of the code can be obtained by using warm starts [1], [36]. Nevertheless, these optimizations would not benefit during

mealtimes due to the high disturbance level caused by the meal.

VI. CONCLUSION

In this paper, we presented a simple way to generate C code from Matlab with application to MPC. The generated C code combines the convenient development of MATLAB and the efficiency and portability of C code. It provides a convenient and efficient solution for the design and the implementation of optimal control algorithms, for instance to embedded systems. Moreover, the code is tailored to the problem size and can be used to solve small scale as well as large scale OCPs. An application to a case study (the blood glucose regulation in people with T1D) shows a significant speedup between the Matlab implementation and the generated C code.

REFERENCES

[1] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.

[2] H. Ferreau, G. Lorini, and M. Diehl, "Fast nonlinear model predictive control of gasoline engines," in *2006 IEEE Conference on Computer Aided Control System Design*, 2006, pp. 2754–2759.

[3] M. Vukov, S. Gros, G. Horn, G. Frison, K. Geebelen, J. B. Jørgensen, J. Swevers, and M. Diehl, "Real-time nonlinear MPC and MHE for a large-scale mechatronic application," *Control Engineering Practice*, vol. 45, pp. 64–78, 2015.

[4] P. Meum, P. Tøndel, J.-M. Godhavn, O. M. Aamo, *et al.*, "Optimization of smart well production through nonlinear model predictive control," in *Intelligent Energy Conference and Exhibition*. Society of Petroleum Engineers, 2008.

[5] A. Capolei, C. Völcker, J. Frydendall, and J. B. Jørgensen, "Oil reservoir production optimization using single shooting and ESDIRK methods," *IFAC Proceedings Volumes*, vol. 45, no. 8, pp. 286–291, 2012.

[6] B. Foss, "Process control in conventional oil and gas fieldschallenges and opportunities," *Control Engineering Practice*, vol. 20, no. 10, pp. 1058–1064, 2012.

[7] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range," *Automatica*, vol. 47, no. 10, pp. 2279–2285, 2011.

[8] J. Andersson, J. Åkesson, and M. Diehl, "CasADi: A symbolic package for automatic differentiation and optimal control," in *Recent Advances in Algorithmic Differentiation*. Springer, 2012, pp. 297–307.

[9] D. P. Word, J. Kang, J. Åkesson, and C. D. Laird, "Efficient parallel solution of large-scale nonlinear dynamic optimization problems," *Computational Optimization and Applications*, vol. 59, no. 3, pp. 667–688, 2014.

[10] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[11] R. Hovorka, V. Canonico, L. J. Chassin, U. Haueter, M. Massi-Benedetti, M. O. Federici, T. R. Pieber, H. C. Schaller, L. Schaupp, T. Vering, and M. E. Wilinska, "Nonlinear model predictive control of glucose concentration in subjects with type 1 diabetes," *Physiological Measurement*, vol. 25, pp. 905–920, 2004.

[12] C. Cobelli, C. Dalla Man, G. Sparacino, L. Magni, G. De Nicolao, and B. P. Kovatchev, "Diabetes: Models, signals, and control," *IEEE Reviews in Biomedical Engineering*, vol. 2, pp. 54–96, 2009.

[13] D. Boiroux, D. A. Finan, N. K. Poulsen, H. Madsen, and J. B. Jørgensen, "Implications and limitations of ideal insulin administration for people with type 1 diabetes," in *UKACC International Conference on Control 2010*, 2010, pp. 156 – 161.

[14] S. Schaller, J. Lippert, L. Schaupp, T. Pieber, A. Schuppert, and T. Eissing, "Robust PBPK/PD based model predictive control of blood glucose," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 7, pp. 1492 – 1504, 2014.

[15] H. Bock and K. Plitt, "A multiple shooting method for direct solution of optimal control problems," in *Proc. of the IFAC 9th World Congress*, Budapest, Hungary, 1984, pp. 242–247.

[16] D. B. Leineweber, I. Bauer, H. G. Bock, and J. P. Schlöder, "An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization (part I and II)." *Computers & Chemical Engineering*, vol. 27, no. 2, pp. 157–174, 2003.

[17] M. Diehl, J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear MPC and moving horizon estimation," in *Nonlinear Model Predictive Control. Towards New Challenging Applications*. Berlin, Germany: Springer, 2009, pp. 391–417.

[18] D. Boiroux, D. A. Finan, N. K. Poulsen, H. Madsen, and J. B. Jørgensen, "Nonlinear model predictive control for an artificial β-cell," in *Recent Advances in Optimization and its Applications in Engineering*. Springer, 2010, pp. 299 – 308.

[19] C. V. Rao, S. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *Journal of Optimization Theory and Applications*, vol. 99, no. 3, pp. 723 – 757, 1998.

[20] J. B. Jørgensen, J. B. Rawlings, and S. B. Jørgensen, "Numerical methods for large-scale moving horizon estimation and control," in *Proceedings of the 7th International Symposium on Dynamics and Control Process Systems (DYCOPS)*, 2004.

[21] J. B. Jørgensen, "Moving horizon estimation and control," Ph.D. dissertation, Department of Chemical Engineering, Technical University of Denmark, 2005.

[22] J. B. Jørgensen, G. Frison, N. F. Gade-Nielsen, and B. Damman, "Numerical methods for solution of the extended linear quadratic control problem," in *IFAC Conference on Nonlinear Model Predictive Control 2012 (NMPC 2012)*, 2012, pp. 187–193.

[23] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, USA: Springer, 2006.

[24] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal of Optimization*, vol. 2, no. 4, pp. 575–601, 1992.

[25] D. Boiroux, "Model predictive control algorithms for pen and pump insulin administration," Ph.D. dissertation, Department of Informatics and Mathematical Modeling, Technical University of Denmark, 2012.

[26] G. Frison and J. B. Jørgensen, "Efficient implementation of the Riccati recursion for solving linear-quadratic control problems," in *2013 IEEE International Conference on Control Applications (CCA)*, 2013, pp. 1117–1122.

[27] "R2011a - Updates to the MATLAB and Simulink product families," http://www.mathworks.com/products/new_products/release2011a.html, [Online; accessed 14-September-2016].

[28] H. Zarrinkoub, *Understanding LTE with MATLAB: From Mathematical Modeling to Simulation and Prototyping*. John Wiley & Sons, Ltd, 2014.

[29] L. Baresi, M. Mauri, A. Monti, and M. Pezzè, "PLCTools: design, formal validation, and code generation for programmable controllers," in *2000 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 4, 2000, pp. 2437–2442.

[30] R. Bucher and S. Balemi, "Rapid controller prototyping with matlab/simulink and linux," *Control Engineering Practice*, vol. 14, no. 2, pp. 185–192, 2006.

[31] Y. M. Altman, *Accelerating MATLAB Performance: 1001 tips to speed up MATLAB programs*. CRC Press, 2014.

[32] J. B. Jørgensen, M. R. Kristensen, P. G. Thomsen, and H. Madsen, "A numerically robust ESDIRK-based implementation of the continuous-discrete extended Kalman filter," in *European Control Conference 2007*. Kos, Greece: ECC 2007, 2007.

[33] D. Boiroux, D. A. Finan, N. K. Poulsen, H. Madsen, and J. B. Jørgensen, "Meal estimation in nonlinear model predictive control for type 1 diabetes," in *Symposium on Nonlinear Control Systems 2010 (NOLCOS 2010)*, 2010, pp. 1052 – 1057.

[34] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design*. Madison, Wisconsin, USA: Nob Hill Publishing, 2009.

[35] S. S. Kanderian, S. Weinzimer, G. Voskanyan, and G. M. Steil, "Identification of intraday metabolic profiles during closed-loop glucose control in individuals with type 1 diabetes," *Journal of Diabetes Science and Technology*, vol. 3, no. 5, pp. 1047 – 1057, 2009.

[36] L. E. Sokoler, A. Skajaa, G. Frison, R. Halvgaard, and J. B. Jørgensen, "A warm-started homogeneous and self-dual interior-point method for linear economic model predictive control," in *52nd IEEE Conference on Decision and Control*, 2013, pp. 3677–3683.