



# Wrocław University of Technology

Chair of Cybernetics and Robotics



## Miscellaneous libraries

Mariusz Janiak  
p. 331 C-3, 71 320 26 44

© 2015 Mariusz Janiak  
All Rights Reserved



# Contents

- 1 Physic engines
- 2 Linear algebra
- 3 SUNDIALS
- 4 IPOPT
- 5 ACADO Toolkit
- 6 Motion planning



# Physic engines

## Physic engines

- Bullet
- ODE
- Box2D
- OpenTissue
- PhysicsJS
- Havok
- PhysX
- Newton Dynamics
- ...



# Physic engines

## Main functionality of physics engine

- Kinematic
- Dynamic (rigid objects, cloth, fluids)
- Constraints
- Collision detection



# Physic engines

## Simulation environments

- Gazebo
- v-rep
- Microsoft Robotics Developer Studio
- LabVIEW Robotics Module
- Webots
- USARsim



# Linear algebra

## Linear algebra libraries

- LAPACK
- Eigen
- uBLAS
- GSL
- Armadillo
- MKL (Intel Math Kernel Library)
- ACL (AMD Compute Libraries, ACML EoL)
- ...



# Linear algebra

## Example 1 (**Eigen**)

```
1  #include <iostream>
2  #include <Eigen/Dense>
3
4  using Eigen::MatrixXd;
5
6  int main()
7  {
8      MatrixXd m(2,2);
9      m(0,0) = 3;
10     m(1,0) = 2.5;
11     m(0,1) = -1;
12     m(1,1) = m(1,0) + m(0,1);
13     std::cout << m << std::endl;
14 }
```



# Linear algebra

## Example 2 (Eigen)

```
1  #include <iostream>
2  #include <Eigen/Dense>
3
4  using namespace Eigen;
5
6  int main()
7  {
8      Matrix2d mat;
9      mat << 1, 2,
10         3, 4;
11     Vector2d u(-1,1), v(2,0);
12
13     std::cout << "mat*mat: " << mat*mat << std::endl;
14     std::cout << "mat*u: " << mat*u << std::endl;
15     std::cout << "u^T*mat: " << u.transpose()*mat << std::endl;
16     std::cout << "u^T*v: " << u.transpose()*v << std::endl;
17     std::cout << "u*v^T: " << u*v.transpose() << std::endl;
```





# Linear algebra

## Example 3 (Eigen)

```
1  #include <iostream>
2  #include <Eigen/Dense>
3
4  using namespace std;
5  using namespace Eigen;
6
7  int main()
8  {
9      Matrix3f A;
10     A << 1, 2, 1,
11         2, 1, 0,
12         -1, 1, 2;
13
14     cout << "Matrix A:\n" << A << endl;
15     cout << "The determinant: " << A.determinant() << endl;
16     cout << "The inverse: \n" << A.inverse() << endl;
17 }
```



# Linear algebra

## Space transformations, Geometry (**Eigen**)

- rotation 2D

```
Rotation2D<float> rot2( angle_in_radian );
```

- rotation 3D (axis, angle)

```
AngleAxis<float> aa( angle_in_radian , Vector3f(ax,ay,az) );
```

- rotation 3D (quaternion)

```
Quaternion<float> q = AngleAxis<float>(angle_in_radian , axis );
```

- translation

```
Translation<float,2>(tx, ty)  
Translation<float,3>(tx, ty, tz)  
Translation<float,N>(s)  
Translation<float,N>(vecN)
```



# SUNDIALS

SUNDIALS (*SU*ite of *N*onlinear and *D*ifferential/*A*lgebraic equation *S*olvers) consists of the following six solvers:

- CVODE – solves initial value problems for ordinary differential equation (ODE) systems
- CVODES – solves ODE systems and includes sensitivity analysis capabilities (forward and adjoint)
- ARKODE – solves initial value ODE problems with additive Runge-Kutta methods, include support for IMEX methods
- IDA – solves initial value problems for differential-algebraic equation (DAE) systems
- IDAS – solves DAE systems and includes sensitivity analysis capabilities (forward and adjoint)
- KINSOL – solves nonlinear algebraic systems

In addition, SUNDIALS provides a Matlab interface to CVODES, IDAS, and KINSOL, sundialsTB



# SUNDIALS

## Unicycle example

$$\begin{cases} \dot{q}_1 = u_1 \cos q_3, \\ \dot{q}_2 = u_1 \sin q_3, \\ \dot{q}_3 = u_2, \end{cases} \quad t = [0, 1], \quad q(0) = (0, 0, 0).$$

Assuming fixed control  $u_1 = 1, u_2 = 0.1$ , we are looking for solution

$$q(t) = \varphi_{q_0, t}(u).$$



# SUNDIALS

## Example – initialization (CVCODE)

```
1 #include <cvode/cvode.h>
2 #include <nvector/nvector_serial.h>
3 #include <cvode/cvode_dense.h>
4 #include <sundials/sundials_dense.h>
5 #include <sundials/sundials_types.h>
6
7 int      i;
8 void     *cvode_mem;
9 realtype t, tb = RCONST(0.0), te = RCONST(1.0);
10 realtype reltol = RCONST(1.0e-3), abstol = RCONST(1.0e-4);
11 N_Vector q
12
13 q = N_VNew_Serial(3);
14 for(i = 0; i < 3; ++i){
15     NV_Ith_S(q, i) = RCONST(0.0);
16 }
17 cvode_mem = CVodeCreate(CV_BDF, CV_NEWTON);
18 CVodeInit(cvode_mem, fun, tb, q);
19 CVodeSStolerances(cvode_mem, reltol, abstol);
20 CVDense(cvode_mem, 3);
21 CVode(cvode_mem, te, q, &t, CV_NORMAL);
```



# SUNDIALS

## Example – Function $\dot{x} = f(x, u)$ (CVCODE)

```
1  int fun(realtype t, N_Vector q, N_Vector dq, void *user_data)
2  {
3      realtype q1, q2, q3;
4      realtype dq1, dq2, dq3;
5      realtype u1, u2;
6
7      q1 = NV_lth_S(q,0);      /* Get actual state */
8      q2 = NV_lth_S(q,1);
9      q3 = NV_lth_S(q,2);
10     u1 = 1.0;                /* Get control signals */
11     u2 = 0.1;
12     dq1 = cos(q3) * u1;      /* Calculate function */
13     dq2 = sin(q3) * u1;
14     dq3 = u2;
15     NV_lth_S(dq,0) = dq1;    /* Set output */
16     NV_lth_S(dq,1) = dq2;
17     NV_lth_S(dq,2) = dq3;
18     return(0);
19 }
```



# IPOPT

Ipopt (*Interior Point OPTimizer*) is a software package for large-scale nonlinear optimization. It is designed to find (local) solutions of mathematical optimization problems of the form

$$\min_{x \in \mathbb{R}^n} f(x)$$

with

$$g_d \leq g(x) \leq g_u$$

$$x_d \leq x \leq x_u$$

where  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $g(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ .



# IPOPT

Example – problem definition

$$\min_{x \in \mathbb{R}^4} \quad x_1 x_4 (x_1 + x_2 + x_3) + x_3$$

with constraints

$$\begin{aligned} x_1 x_2 x_3 x_4 &\geq 25 \\ x_1^2 + x_2^2 + x_3^2 + x_4^2 &= 40 \\ 1 &\leq x_1, x_2, x_3, x_4 \leq 5 \end{aligned}$$

Starting point  $x_0 = (1, 5, 5, 1)$ .





## IPOPT

Example – problem definition (cont.)

$$\nabla f(x) = \begin{bmatrix} x_1 x_4 + x_4(x_1 + x_2 + x_3) \\ x_1 x_4 \\ x_1 x_4 + 1 \\ x_1(x_1 + x_2 + x_3) \end{bmatrix}$$

$$\nabla g(x) = \begin{bmatrix} x_2 x_3 x_4 & x_1 x_3 x_4 & x_1 x_2 x_4 & x_1 x_2 x_3 \\ 2x_1 & 2x_2 & 2x_3 & 2x_4 \end{bmatrix}$$



# IPOPT

## Example – class definition

```
1 class HS071.NLP : public TNLP
2 {
3     public:
4         HS071.NLP();
5         virtual ~HS071.NLP();
6         virtual bool get_nlp_info (...);
7         virtual bool get_bounds_info (...);
8         virtual bool get_starting_point (...);
9         virtual bool eval_f (...);
10        virtual bool eval_grad_f (...);
11        virtual bool eval_g (...);
12        virtual bool eval_jac_g (...);
13        virtual bool eval_h (...);
14        virtual void finalize_solution (...);
15
16    private:
17        HS071.NLP(const HS071.NLP&);
18        HS071.NLP& operator=(const HS071.NLP&);
19 };
```



# IPOPT

## Example – implementation of the selected methods

```
1  bool HS071.NLP::get_nlp_info(Index& n, Index& m, Index& nnz_jac_g ,
2                               Index& nnz_h_lag , IndexStyleEnum& index_style)
3  {
4      // The problem dimation
5      n = 4;
6      // The constraints dimation
7      m = 2;
8      // Number of the Jacobian nonzero elemnts
9      nnz_jac_g = 8;
10     // Number of the Hessian nonzero elemnts
11     nnz_h_lag = 0;
12     // Use the C style indexing (0-based)
13     index_style = TNLP::C_STYLE;
14     return true;
15 }
```



# IPOPT

## Example – implementation of the selected methods (cont.)

```
1  bool UnicycleNlp::get_bounds_info(Index n, Number* x_l, Number* x_u,
2                                     Index m, Number* g_l, Number* g_u)
3  {
4      // The variables have lower bounds of 1
5      for (Index i=0; i<4; i++) {
6          x_l[i] = 1.0;
7      }
8      // The variables have upper bounds of 5
9      for (Index i=0; i<4; i++) {
10         x_u[i] = 5.0;
11     }
12     // The first constraint g1 has a lower bound of 25
13     g_l[0] = 25;
14     // The first constraint g1 has NO upper bound, here we set it to 2e19.
15     // Ipopt interprets any number greater than nlp_upper_bound_inf as
16     // infinity. The default value of nlp_upper_bound_inf and
17     // nlp_lower_bound_inf is 1e19 and can be changed through ipopt options.
18     g_u[0] = 2e19;
19     // The second constraint g2 is an equality constraint, so we set the
20     // upper and lower bound to the same value
21     g_l[1] = g_u[1] = 40.0;
22     return true;
23 }
```



# IPOPT

## Example – implementation of the selected methods (cont.)

```
1  bool HS071.NLP::get_starting_point(Index n, bool init_x, Number* x,
2                                     bool init_z, Number* z_L, Number* z_U,
3                                     Index m, bool init_lambda,
4                                     Number* lambda)
5  {
6      // Initialize to the given starting point
7      x[0] = 1.0;
8      x[1] = 5.0;
9      x[2] = 5.0;
10     x[3] = 1.0;
11     return true;
12 }
```



# IPOPT

## Example – implementation of the selected methods (cont.)

```
1 bool HS071_NLP::eval_f(Index n, const Number* x, bool newx, Number& obj_value)
2 {
3     obj_value = x[0] * x[3] * (x[0] + x[1] + x[2]) + x[2];
4     return true;
5 }
```

```
1 bool HS071_NLP::eval_grad_f(Index n, const Number* x, bool newx,
2                               Number* grad_f)
3 {
4     grad_f[0] = x[0] * x[3] + x[3] * (x[0] + x[1] + x[2]);
5     grad_f[1] = x[0] * x[3];
6     grad_f[2] = x[0] * x[3] + 1;
7     grad_f[3] = x[0] * (x[0] + x[1] + x[2]);
8     return true;
9 }
```

```
1 bool HS071_NLP::eval_g(Index n, const Number* x, bool newx, Index m, Number* g)
2 {
3     g[0] = x[0] * x[1] * x[2] * x[3];
4     g[1] = x[0]*x[0] + x[1]*x[1] + x[2]*x[2] + x[3]*x[3];
5     return true;
6 }
```



# IPOPT

## Example – implementation of the selected methods (cont.)

```
1  bool HS071.NLP::eval_jac_g(Index n, const Number* x, bool new_x, Index m,
2                               Index nele_jac, Index* iRow, Index *jCol,
3                               Number* values)
4  {
5      if (values == NULL) {
6          // Return the structure of the jacobian
7          // this particular jacobian is dense
8          iRow[0] = 0; jCol[0] = 0; iRow[1] = 0; jCol[1] = 1;
9          iRow[2] = 0; jCol[2] = 2; iRow[3] = 0; jCol[3] = 3;
10         iRow[4] = 1; jCol[4] = 0; iRow[5] = 1; jCol[5] = 1;
11         iRow[6] = 1; jCol[6] = 2; iRow[7] = 1; jCol[7] = 3;
12     }
13     else {
14         // Return the values of the jacobian of the constraints
15         values[0] = x[1]*x[2]*x[3]; // 0,0
16         values[1] = x[0]*x[2]*x[3]; // 0,1
17         values[2] = x[0]*x[1]*x[3]; // 0,2
18         values[3] = x[0]*x[1]*x[2]; // 0,3
19         values[4] = 2*x[0]; // 1,0
20         values[5] = 2*x[1]; // 1,1
21         values[6] = 2*x[2]; // 1,2
22         values[7] = 2*x[3]; // 1,3
23     }
24     return true;
25 }
```



# IPOPT

## Example – implementation of the selected methods (cont.)

```
1 void HS071.NLP::finalize_solution(SolverReturn status, Index n,
2                                 const Number* x, const Number* z_L,
3                                 const Number* z_U, Index m, const Number* g,
4                                 const Number* lambda, Number obj_value,
5                                 const IpoptData* ip_data,
6                                 IpoptCalculatedQuantities* ip_cq)
7 {
8     printf("\n\nSolution of the primal variables, x\n");
9     for (Index i=0; i<n; i++) {
10         printf("x[%d] = %e\n", i, x[i]);
11     }
12     printf("\n\nSolution of the bound multipliers, z_L and z_U\n");
13     for (Index i=0; i<n; i++) {
14         printf("z_L[%d] = %e\n", i, z_L[i]);
15     }
16     for (Index i=0; i<n; i++) {
17         printf("z_U[%d] = %e\n", i, z_U[i]);
18     }
19     printf("\n\nObjective value\n");
20     printf("f(x*) = %e\n", obj_value);
21     printf("\n\nFinal value of the constraints:\n");
22     for (Index i=0; i<m; i++) {
23         printf("g(%d) = %e\n", i, g[i]);
24     }
25 }
```





# ACADO Toolkit

ACADO Toolkit is a software environment and algorithm collection for automatic control and dynamic optimization. It provides a general framework for using a great variety of algorithms for

- Nonlinear Optimal Control
- Multi-Objective Optimal Control
- State and Parameter Estimation
- Model Predictive Control
- Code Generation for Fast NMPC and MHE



# ACADO Toolkit

Example – motion planning. Problem definition

$$\min_{u(\cdot)} \mathcal{J}(u(\cdot))$$

where

$$\mathcal{J}(u(\cdot)) = (q(T) - q_d)^T Q_1 (q(T) - q_d) + \int_0^T (u(t) - u_d)^T Q_2 (u(t) - u_d) dt$$

with constraints

$$\dot{q} = G(q)u$$

$$|q(T) - q_d| \leq q_{ub}$$

$$u_{lb} \leq u \leq u_{ub}$$



# ACADO Toolkit

## Example

```
1 #include <algorithm>
2 #include <acado_optimal_control.hpp>
3 #include <gnuplot/acado2gnuplot.hpp>
4
5 int main( ){
6     USING_NAMESPACE_ACADO;
7     // INTRODUCE THE VARIABLES:
8     DifferentialState q1;
9     DifferentialState q2;
10    DifferentialState q3;
11    Control          u1;
12    Control          u2;
13    // DEFINE A DIFFERENTIAL EQUATION:
14    DifferentialEquation f;
15    f << dot(q1) == cos(q3) * u1;
16    f << dot(q2) == sin(q3) * u1;
17    f << dot(q3) == u2;
18    // DEFINE INITIAL STATE
19    Vector q0(3);
20    q0(0) = Q0_1;
21    q0(1) = Q0_2;
22    q0(2) = Q0_3;
23    Vector u0(2);
24    u0(0) = U0_1;
25    u0(1) = U0_2;
```



# ACADO Toolkit

## Example (cont.)

```
1  Function h1, h2;
2  h1 << q1;
3  h1 << q2;
4  h1 << q3;
5  h2 << u1;
6  h2 << u2;
7
8  Vector qd(3);
9  qd(0) = QD_1;
10 qd(1) = QD_2;
11 qd(2) = QD_3;
12
13 Vector ud(2);
14 ud(0) = 0;
15 ud(1) = 0;
16
17 Matrix Q1(3,3);
18 Q1.setIdentity();
19 Q1(0,0) = 1.0;
20 Q1(1,1) = 1.0;
21 Q1(2,2) = 1.0;
22
23 Matrix Q2(2,2);
24 Q2.setIdentity();
25 Q2(0,0) = 0.1;
26 Q2(1,1) = 0.1;
```



# ACADO Toolkit

## Example (cont.)

```
1 // DEFINE AN OPTIMAL CONTROL PROBLEM:
2 const double t_start = 0.0;
3 const double t_end   = 1.0;
4
5 OCP ocp( t_start , t_end , 20 );
6 ocp.minimizeLSQEndTerm(Q1, h1, qd);
7 ocp.minimizeLSQ(Q2, h2, ud);
8 ocp.subjectTo(f);
9 ocp.subjectTo(AT_START, q1 == q0(0));
10 ocp.subjectTo(AT_START, q2 == q0(1));
11 ocp.subjectTo(AT_START, q3 == q0(2));
12 ocp.subjectTo(AT_START, u1 == u0(0));
13 ocp.subjectTo(AT_START, u2 == u0(1));
14 ocp.subjectTo(U2_LB <= u2 <= U2_UB);
15 ocp.subjectTo(AT_END, qd(0) - 0.01 <= q1 <= qd(0) + 0.01);
16 ocp.subjectTo(AT_END, qd(1) - 0.01 <= q2 <= qd(1) + 0.01);
17 ocp.subjectTo(AT_END, qd(2) - 0.1 <= q3 <= qd(2) + 0.1);
18
19 // DEFINE AN OPTIMIZATION ALGORITHM AND SOLVE THE OCP:
20 OptimizationAlgorithm algorithm(ocp);
21
22 algorithm.solve();
23 return 0;
```



# Motion planning

## Motion planning libraries

- MPK (*Motion Planning Kit*),  
<http://ai.stanford.edu/~mitul/mpk>
- OMPL (*Open Motion Planning Library*),  
<http://ompl.kavrakilab.org/index.html>
- OOPSMP (*Object-Oriented Programming System for Motion Planning*),  
<http://www.kavrakilab.org/OOPSMP/index.html>
- MSL (*Motion Strategy Library*)  
<http://mssl.cs.uiuc.edu/mssl>
- SIMOX  
<http://simox.sourceforge.net>



# The End

Thank you for your kind attention.