Nonlinear Optimization for Optimal Control
Part 2

Pieter Abbeel
UC Berkeley EECS

Outline

- **From linear to nonlinear**
- Model-predictive control (MPC)
- POMDPs

From Linear to Nonlinear

- We know how to solve (assuming g_t, U_t, X_t convex):

$$
\begin{aligned}
\min_{u,x} \quad & \sum_{t=0}^{H} g_t(x_t, u_t) \\
\text{subject to} \quad & x_{t+1} = A_t x_t + B_t u_t + c_t \quad \forall t \\
& u_t \in \mathcal{U}_t, x_t \in \mathcal{X}_t \quad \forall t
\end{aligned}
\qquad (1)
$$

- How about nonlinear dynamics: $x_{t+1} = f(x_t, u_t) \quad \forall t$

Shooting Methods (feasible)

Iterate for i=1, 2, 3, …

Execute $u_0^{(i)}, u_1^{(i)}, \ldots, u_T^{(i)}$ (from solving (1))

Linearize around resulting trajectory

Solve (1) for current linearization

Collocation Methods (infeasible)

Iterate for i=1, 2, 3, …

--- (no execution)---

Linearize around current solution of (1)

Solve (1) for current linearization

Sequential Quadratic Programming (SQP) = either of the above methods, but instead of using linearization, linearize equality constraints, convex-quadratic approximate objective function

Example Shooting

```
%% a nonlinear control problem: cartpole
clear; clc; close all;

% shooting:
T = 100;
u = randn(1,T)*0.1;
max_iters = 10;
x_init = [-10; 0; 0; 0];
nX = 4; nU = 1;
x_eps = 0.1;
u_eps = 0.1;
dt = 0.1;
Q = eye(nX); R = eye(nU); Q_final = 100*eye(nX);
clear A B c
for iter = 1:max_iters
    % simulate and linearize
    x(:,1) = x_init;
    for t=1:T-1
        x(:,t+1) = sim_cartpole(x(:,t), u(:,t), dt);
        [A{t} B{t} c{t}] = compute_jacobian(@sim_cartpole, x(:,t), u(:,t), dt);
        %cartpole_draw(t*dt, x(:,t));
    end
    figure(1); subplot(3,1,1); hold on; plot(u); ylabel('u');
    subplot(3,1,2); hold on; plot(x(1,:)); ylabel('x'); subplot(3,1,3); hold on; plot(x(2,:));ylabel('\theta');
    cost(iter) = 0;
    for t=1:T-1
        cost(iter) = cost(iter) + x(:,t)'*Q*x(:,t) + u(:,t)'*R*u(:,t) + norm(u(:,t+1)-u(:,t),2);
    end
    cost(iter) = cost(iter) + x(:,T)'*Q_final*x(:,T);
    cost
    % solve convex problem
    cvx_begin
    variables x_cvx(nX,T)  u_cvx(nU,T) s_cvx(1,T);
    minimize( sum(s_cvx(1:T)) )
    subject to
    for t=1:T-1
        x_cvx(:,t+1) == A{t}*(x_cvx(:,t)-x(:,t)) + B{t}*(u_cvx(:,t)-u(:,t)) + c{t};
    end
    for t=1:T-1
        s_cvx(1,t) >= x_cvx(:,t)'*Q*x_cvx(:,t) + u_cvx(:,t)'*R*u_cvx(:,t) + norm(u_cvx(:,t+1)-u_cvx(:,t),2);
    end
    s_cvx(1,T) >= x_cvx(:,T)'*Q_final*x_cvx(:,T);
    for t=1:T
        norm(x_cvx(:,t) - x(:,t),2) <= x_eps;
        norm(u_cvx(:,t) - u(:,t),2) <= u_eps;
    end
    x_cvx(:,1) == x_init;
    cvx_end
    u = u_cvx;
end
```

Example Collocation

```
clear; clc; close all;

T = 100;
u = randn(1,T)*0.1;
max_iters = 10;
x_init = [-10; pi/10; -0.1; 0.1];
nX = 4; nU = 1;
x_eps = 100;
u_eps = 1;
dt = 0.1;
Q = eye(nX); R = eye(nU); Q_final = 100*eye(nX);
clear A B c
x_target = [0;0;0;0];
for i=1:nX
    x_iter1(i,:) = x_init(i):(x_target(i)-x_init(i))/(T-1):x_target(i);
end
u_iter1=zeros(nU,T);

for iter = 1:max_iters
    % linearize (but no simulation)
    if(iter==1)
        x = x_iter1; u = u_iter1;
    else
        x = x_cvx; u = u_cvx;
    end
    for t=1:T-1
        %   x(:,t+1) = sim_cartpole(x(:,t), u(:,t), dt);
        [A{t} B{t} c{t}] = compute_jacobian(@sim_cartpole, x(:,t), u(:,t), dt);
        %cartpole_draw(t*dt, x(:,t));
    end
    figure(2); subplot(3,1,1); hold on; plot(u); ylabel('u');
    subplot(3,1,2); hold on; plot(x(1,:)); ylabel('x'); subplot(3,1,3); hold on; plot(x(2,:));ylabel('\theta');
    cost(iter) = 0;
    for t=1:T-1
        cost(iter) = cost(iter) + x(:,t)'*Q*x(:,t) + u(:,t)'*R*u(:,t) + norm(u(:,t+1)-u(:,t),2);
    end
    cost(iter) = cost(iter) + x(:,T)'*Q_final*x(:,T);
    cost
    % solve convex problem
    cvx_begin
    variables x_cvx(nX,T)  u_cvx(nU,T) s_cvx(1,T);
    minimize( sum(s_cvx(1:T)) )
    subject to
    for t=1:T-1
        x_cvx(:,t+1) == A{t}*(x_cvx(:,t)-x(:,t)) + B{t}*(u_cvx(:,t)-u(:,t)) + c{t};
    end
    for t=1:T-1
        s_cvx(1,t) >= x_cvx(:,t)'*Q*x_cvx(:,t) + u_cvx(:,t)'*R*u_cvx(:,t) + norm(u_cvx(:,t+1)-u_cvx(:,t),2);
    end
    s_cvx(1,T) >= x_cvx(:,T)'*Q_final*x_cvx(:,T);
    for t=1:T
        norm(x_cvx(:,t) - x(:,t),2) <= x_eps;
        norm(u_cvx(:,t) - u(:,t),2) <= u_eps;
    end
    x_cvx(:,1) == x_init;
    cvx_end
    u = u_cvx;
end

% let's evaluate the resulting open-loop sequence:
x(:,1) = x_init;
for t=1:T-1
    x(:,t+1) = sim_cartpole(x(:,t), u(:,t), dt);
end
figure(3); subplot(3,1,1); hold on; plot(u); ylabel('u');
subplot(3,1,2); hold on; plot(x(1,:)); ylabel('x'); subplot(3,1,3); hold on; plot(x(2,:));ylabel('\theta');
final_cost = 0;
for t=1:T-1
  final_cost = final_cost + x(:,t)'*Q*x(:,t) + u(:,t)'*R*u(:,t) + norm(u(:,t+1)-u(:,t),2);
end
final_cost = final_cost + x(:,T)'*Q_final*x(:,T);
final_cost
```

Practical Benefits and Issues with Shooting

+ At all times the sequence of controls is meaningful, and the objective function optimized directly corresponds to the current control sequence

- For unstable systems, need to run feedback controller during forward simulation

- Why? Open loop sequence of control inputs computed for the linearized system will not be perfect for the nonlinear system. If the nonlinear system is unstable, open loop execution would give poor performance.
- Fixes:
  - Run Model Predictive Control for forward simulation
  - Compute a linear feedback controller from the 2nd order Taylor expansion at the optimum (exercise: work out the details!)

Practical Benefits and Issues with Collocation

+ Can initialize with infeasible trajectory. Hence if you have a rough idea of a sequence of states that would form a reasonable solution, you can initialize with this sequence of states without needing to know a control sequence that would lead through them, and without needing to make them consistent with the dynamics

- Sequence of control inputs and states might never converge onto a feasible sequence

Iterative LQR versus Sequential Convex Programming

- Both can solve

$$\begin{aligned} \min_{u,x} \quad & \sum_{t=0}^{H} g_t(x_t, u_t) \\ \text{subject to} \quad & x_{t+1} = f_t(x_t, u_t) \quad \forall t \\ & u_t \in \mathcal{U}_t, x_t \in \mathcal{X}_t \quad \forall t \end{aligned}$$

- Can run iterative LQR both as a shooting method or as a collocation method, it's just a different way of executing "Solve (1) for current linearization." In case of shooting, the sequence of linear feedback controllers found can be used for (closed-loop) execution.

- Iterative LQR might need some outer iterations, adjusting "t" of the log barrier

Shooting Methods (feasible)	**Collocation Methods (infeasible)**
Iterate for i=1, 2, 3, …	Iterate for i=1, 2, 3, …
Execute feedback controller (from solving (1))	--- (no execution)---
Linearize around resulting trajectory	Linearize around current solution of (1)
Solve (1) for current linearization	Solve (1) for current linearization

Sequential Quadratic Programming (SQP) = either of the above methods, but instead of using linearization, linearize equality constraints, convex-quadratic approximate objective function

Outline

- From linear to nonlinear
- **Model-predictive control (MPC)**

  For an entire semester course on MPC: see Francesco Borrelli
- POMDPs

Model Predictive Control

- Given: $\bar{x}_0$
- For k=0, 1, 2, …, T
  - Solve

$$\begin{aligned} \min_{x,u} \quad & \sum_{t=k}^{T} g_t(x_t, u_t) \\ \text{s.t.} \quad & x_{t+1} = f_t(x_t, u_t) \quad \forall t \in \{k, k+1, \dots, T-1\} \\ & x_k = \bar{x}_k \end{aligned}$$

  - Execute u_k
  - Observe resulting state, $\bar{x}_{k+1}$

Initialization

- Initialization with solution from iteration k-1 can make solver very fast
  - can be done most conveniently with infeasible start Newton method

Terminal Cost

- Re-solving over full horizon can be computationally too expensive given frequency at which one might want to do control
- Instead solve

Estimate of cost-to-go

$$
\begin{aligned}
\min_{x,u} \quad & \sum_{t=k}^{t+H-1} g_t(x_t, u_t) + \hat{J}^{(t+H)}(x_{t+H}) \\
\text{s.t.} \quad & x_{t+1} = f_t(x_t, u_t) \quad \forall t \in \{k, k+1, \ldots, t+H-1\} \\
& x_k = \bar{x}_k
\end{aligned}
$$

- Estimate of cost-to-go
  - If using iterative LQR can use quadratic value function found for time t+H
  - If using nonlinear optimization for open-loop control sequence→can find quadratic approximation from Hessian at solution (exercise, try to derive it!)

Car Control with MPC Video

- Prof. Francesco Borrelli (M.E.) and collaborators
  - http://video.google.com/videoplay?docid=-8338487882440308275

Outline

- From linear to nonlinear
- Model-predictive control (MPC)
- **POMDPs**

POMDP Examples

- Localization/Navigation

  → Coastal Navigation

- SLAM + robot execution

  → Active exploration of unknown areas

- Needle steering

  → maximize probability of success

- "Ghostbusters" (188)

→ Can choose to "sense" or "bust" while navigating a maze with ghosts

- "Certainty equivalent solution" does not always do well

Robotic Needle Steering

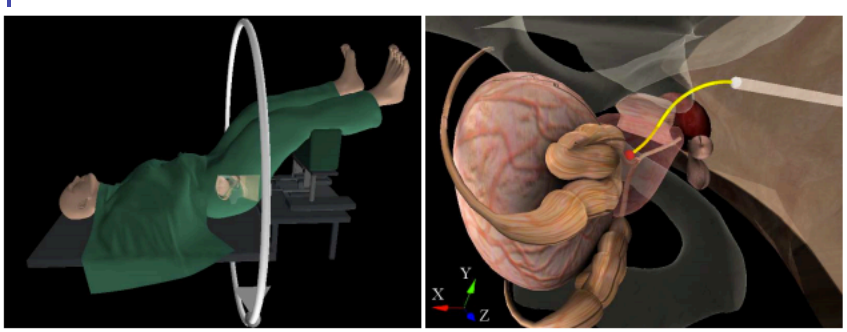

Fig. 5 Using an x-ray imager mounted on a rotating C-arm, it is possible to rotate the sensor about the horizontal axis along which the patient is positioned (left). The anatomy as viewed from the computed optimal sensor placement (right). The optimal path predominantly lies in the imaging plane to minimize uncertainty in the viewing direction.

[from van den Berg, Patil, Alterovitz, Abbeel, Goldberg, WAFR2010]

Robotic Needle Steering

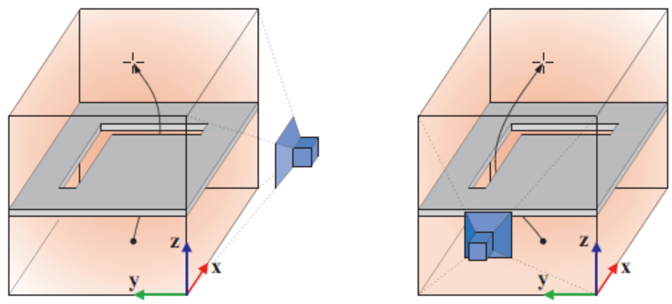

Fig. 1 Two examples of sensor placement, in which (left) only the *x*- and *z*-coordinate and (right) only the *y*- and *z*-coordinate of the needle-tip are measured by the imaging device (blue). Different paths will be optimal even as the obstacles (grey) and target location (cross) are the same.

[from van den Berg, Patil, Alterovitz, Abbeel, Goldberg, WAFR2010]

POMDP: Partially Observable Markov Decision Process

- Belief state B_t, $B_t(x) = P(x_t = x \mid z_0, \ldots, z_t, u_0, \ldots, u_{t-1})$
- If the control input is u_t, and observation z_{t+1} then

$$B_{t+1}(x') = \sum_x B_t(x)\, P(x'|x,u_t)\, P(z_{t+1}|x')$$

POMDP Solution Methods

- Value Iteration:
  - Perform value iteration on the "belief state space"
  - High-dimensional space, usually impractical
- Approximate belief with Gaussian
  - Just keep track of mean and covariance
  - Using (extended or unscented) KF, dynamics model, observation model, we get a nonlinear system equation for our new state variables, $(\mu_{t+1}, \Sigma_{t+1})$:

$$(\mu_{t+1}, \Sigma_{t+1}) = f(\mu_t, \Sigma_t, u_t, E[Z_{t+1}])$$

  - Can now run any of the nonlinear optimization methods for optimal control

Example: Nonlinear Optimization for Control in Belief Space using Gaussian Approximations

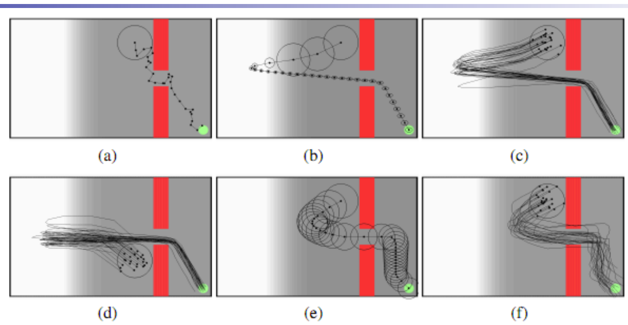

Fig. 1 Point robot moving in a 2-D environment with obstacles. (a) An initial collision-free trajectory is computed using an RRT planner. (b) Nominal trajectory and the associated beliefs of solution computed using our method. The robot moves away from the goal to better localize itself before reaching the goal with significantly reduced uncertainty. Execution traces of the robot's true state starting from the initial belief (c) and a different initial belief (d), while following the computed control policy. (e) Nominal trajectory computed by ignoring the innovation term in the belief dynamics. The optimization is unable to progress sufficiently to the region of the environment with reliable sensing, resulting in considerable uncertainty in the robot state near the obstacles and at the goal. (f) Execution traces of the robot's true state starting from the initial belief and ignoring the innovation term are much noisier as compared to the execution traces shown in (c).

[van den Berg, Patil, Alterovitz, ISSR 2011]

Example: Nonlinear Optimization for Control in Belief Space using Gaussian Approximations

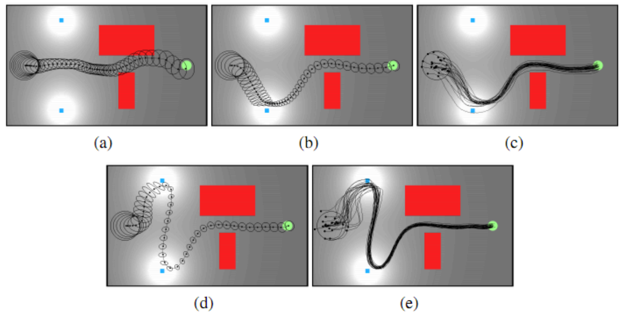

Fig. 2 A car-like robot with second order dynamics moving in a 2-D environment with obstacles. The robot obtains measurements from two beacons (marked by blue squares) and an on-board speedometer. (a) An initial collision-free trajectory is computed using an RRT planner. (b) Nominal trajectory computed using our method. Notice how the car-like robot localizes itself by moving closer to the beacon before reaching the goal. (c) Execution traces of the robot's true state starting from the initial belief for the control policy computed in (b). The jaggedness of the paths is due to the large amount of artificial motion and measurement noise introduced in the simulation. The control policy is safely able to guide the robot to the goal, in spite of the large amount of noise. (d) Nominal trajectory computed by varying the cost matrices ($Q_t = 10I$). The robot tries to reduce the uncertainty in its state by visiting both the beacons. (e) Execution traces of the robot's true state starting from the initial belief for the control policies computed in (d).

[van den Berg, Patil, Alterovitz, ISSR 2011]

Linear Gaussian System with Quadratic Cost: Separation Principle

- Very special case:
  - Linear Gaussian Dynamics
  - Linear Gaussian Observation Model
  - Quadratic Cost
- **Fact:** The optimal control policy *in belief space* for the above system consists of running
  - the optimal feedback controller for the same system when the state is fully observed, which we know from earlier lectures is a time-varying linear feedback controller easily found by value iteration
  - a Kalman filter, which feeds its state estimate into the feedback controller