

Random Sampling of States in Dynamic Programming

Christopher G. Atkeson and Benjamin Stephens CMU 

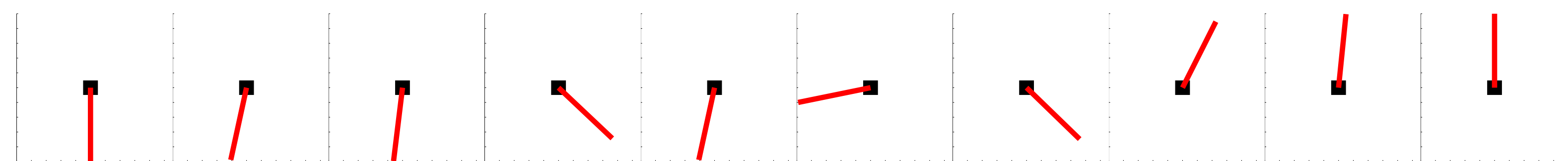
We combine three threads of research on approximate dynamic programming: sparse random sampling of states, value function and policy approximation using local models, and using local trajectory optimizers to globally optimize a policy and associated value function. Our focus is on finding steady state policies for deterministic time invariant discrete time control problems with continuous states and actions often found in robotics. In this paper we describe our approach and provide initial results on several simulated robotics problems, including bipedal standing balance.

Approach: Optimal control provides a potentially useful methodology to design nonlinear control laws (policies) $\mathbf{u} = \mathbf{u}(\mathbf{x})$ which give the appropriate action \mathbf{u} for any state \mathbf{x} . Dynamic programming provides a way to find globally optimal control laws, given a one step cost (a.k.a. “reward” or “loss”) function and the dynamics of the problem to be optimized. We focus on control problems with continuous states and actions, deterministic time invariant discrete time dynamics $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$, and a time invariant one step cost function $L(\mathbf{x}, \mathbf{u})$. Policies for such time invariant problems will also be time invariant. We assume we know the dynamics and one step cost function. Future work will address simultaneously learning a dynamic model, finding a robust policy, and performing state estimation with an erroneous partially learned model.

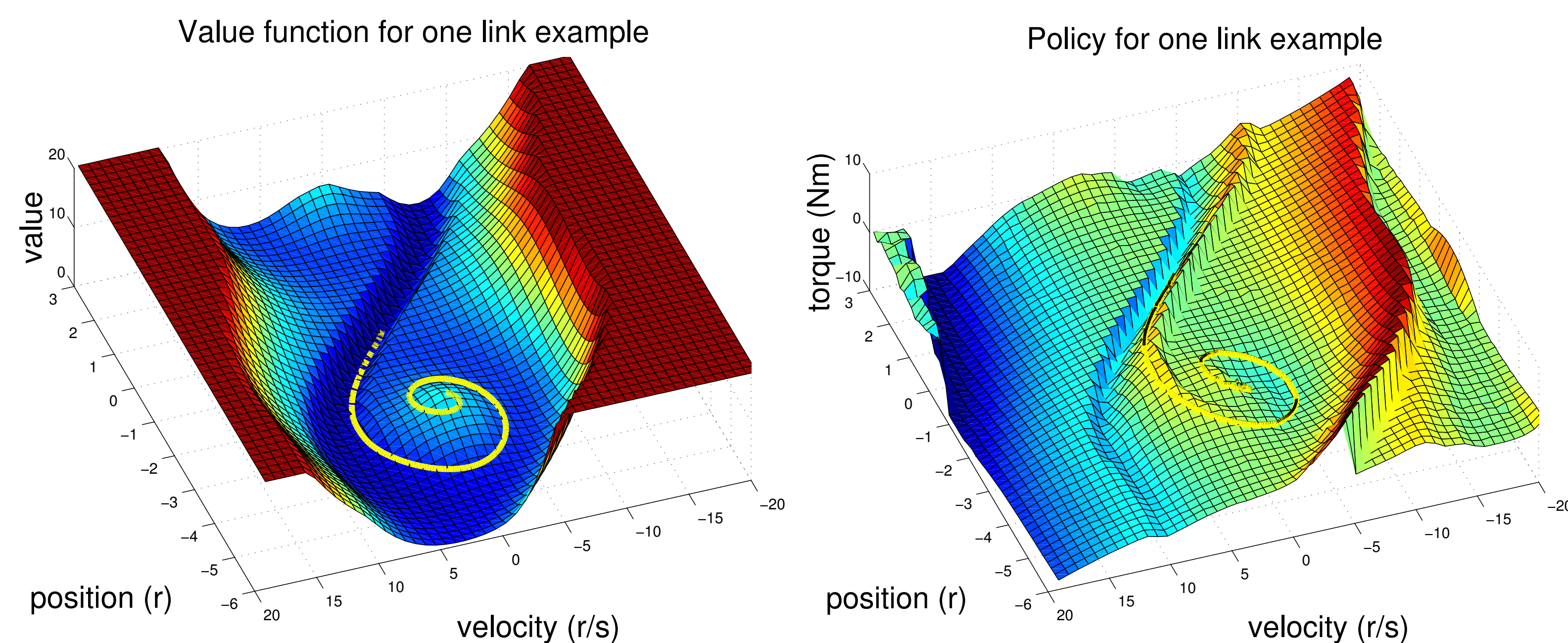
One approach to dynamic programming is to approximate the value function $V(\mathbf{x})$ (the optimal total future cost from each state $V(\mathbf{x}) = \sum_{k=0}^{\infty} L(\mathbf{x}_k, \mathbf{u}_k)$), and to repeatedly solve the Bellman equation $V(\mathbf{x}) = \min_{\mathbf{u}} (L(\mathbf{x}, \mathbf{u}) + V(\mathbf{f}(\mathbf{x}, \mathbf{u})))$ at sampled states \mathbf{x} until the value function estimates have converged to globally optimal values. We explore approximating the value function and policy using many local models, rather than using global parametric models. Typically states used for training during learning are sampled on a regular grid. If this grid has a resolution R in each of d dimensions, there are R^d total states. A tabular approximation of the value function or a training approach based on sampling states on a regular grid have memory and computational costs that grow quickly with both resolution R and state dimensionality d . In our approach we randomly select states, and associate with each state a local quadratic model of the value function and a local linear model of the policy.

Our approach generalizes value iteration, and has the following components: 1. There is a “global” function approximator for both the value function and the policy. In our current implementation the value function and policy are represented through a combination of sampled and parametric representations, building global approximations by combining local models. 2. It is possible to estimate the value of a state in two ways. The first is to use the approximated value function. The second is our analog of using the Bellman equation: use the cost of a trajectory starting from the state under consideration and following the current global policy. The trajectory is optimized using local trajectory optimization. 3. As in a Bellman update, there is a way to globally optimize the value of a state by considering many possible “actions”. In our approach we consider many local policies associated with different stored states.

An example system: actuated one link pendulum swingup.

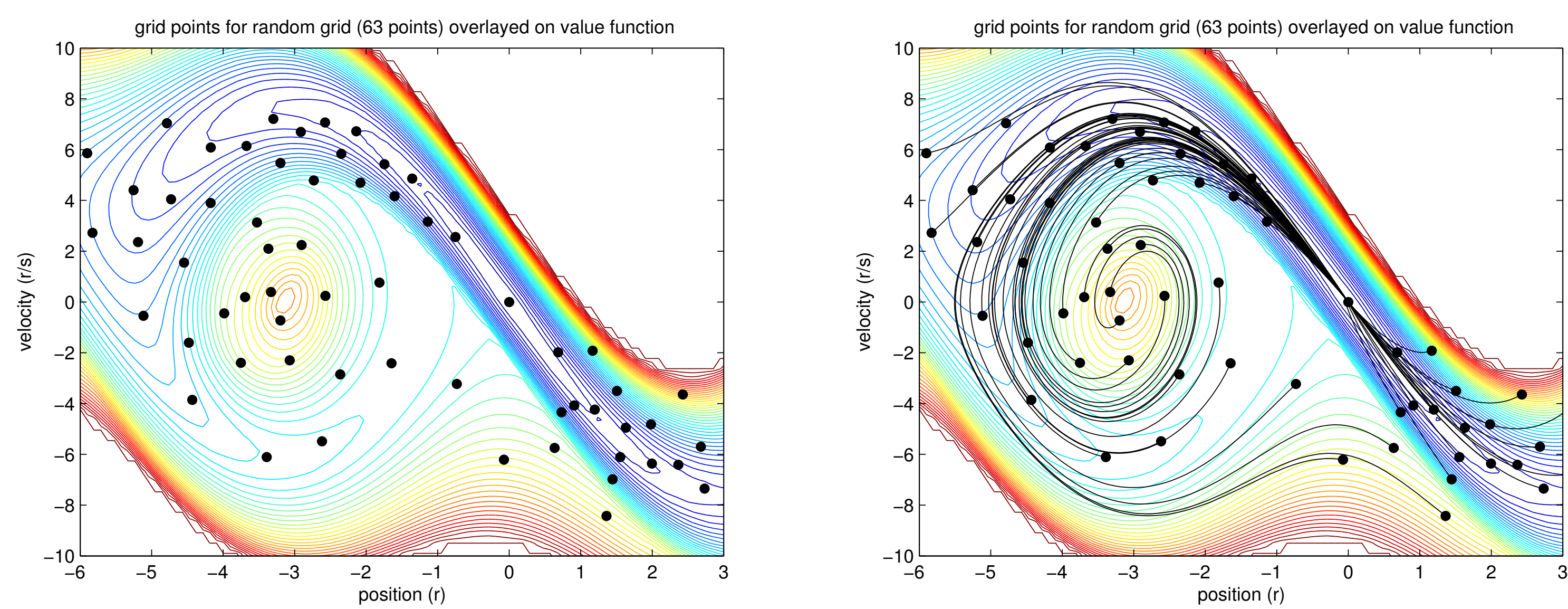


Configurations from the simulated one link pendulum swingup optimal trajectory every half a second and at the end of the trajectory.

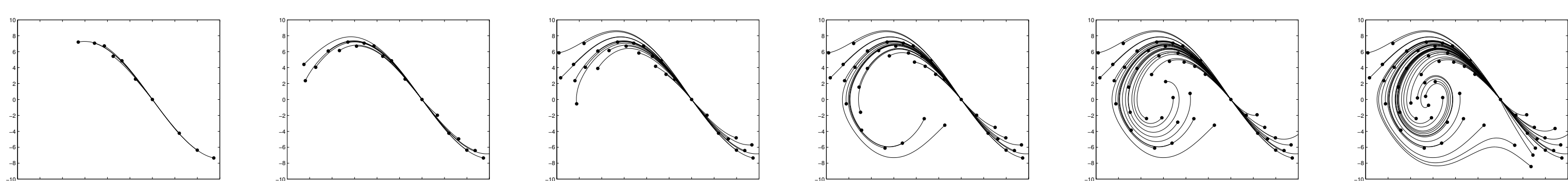


The value function and policy for a one link pendulum swingup. The optimal trajectory is shown as a yellow line in the value function plot, and as a black line with a yellow border in the policy plot. The value function is cut off above 20 so we can see the details of the part of the value function that determines the optimal trajectory. The goal is at the state (0,0).

Local models of the value function and policy: We need to represent value functions as sparsely as possible. We propose a hybrid tabular and parametric approach: parametric local models of the value function and policy are represented at sampled locations. This representation is similar to using many Taylor series approximations of a function at different points. At each sampled state \mathbf{x}^p the local quadratic model for the value function is: $V^p(\mathbf{x}) \approx V_0^p + V_x^p \cdot \mathbf{x} + \frac{1}{2} \mathbf{x}^T V_{xx}^p \cdot \mathbf{x}$ where $\mathbf{x} = \mathbf{x} - \mathbf{x}^p$ is the vector from the stored state \mathbf{x}^p , V_0^p is the constant term of the local model, V_x^p is the first derivative of the local model (and the value function) at \mathbf{x}^p , and V_{xx}^p is the second derivative of the local model (and the value function) at \mathbf{x}^p . The local linear model for the policy is: $\mathbf{u}^p(\mathbf{x}) = \mathbf{u}_0^p - \mathbf{K}^p \cdot \mathbf{x}$ where \mathbf{u}_0^p is the constant term of the local policy, and \mathbf{K}^p is the first derivative of the local policy and also the gain matrix for a local linear controller. These local models of the value function and policy can be created using Differential Dynamic Programming (DDP).

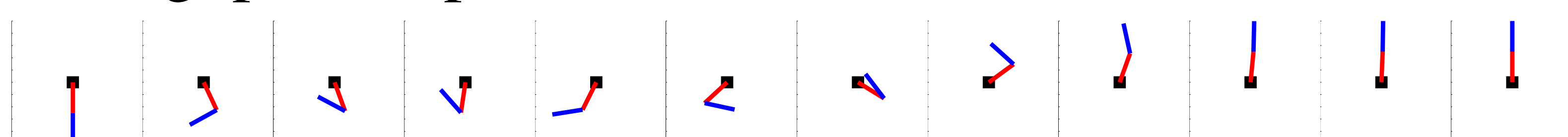


Left: Random states (dots) used to plan one link swingup, superimposed on a contour map of the value function. **Right:** Optimized trajectories (black lines) starting from the random states in the left figure.



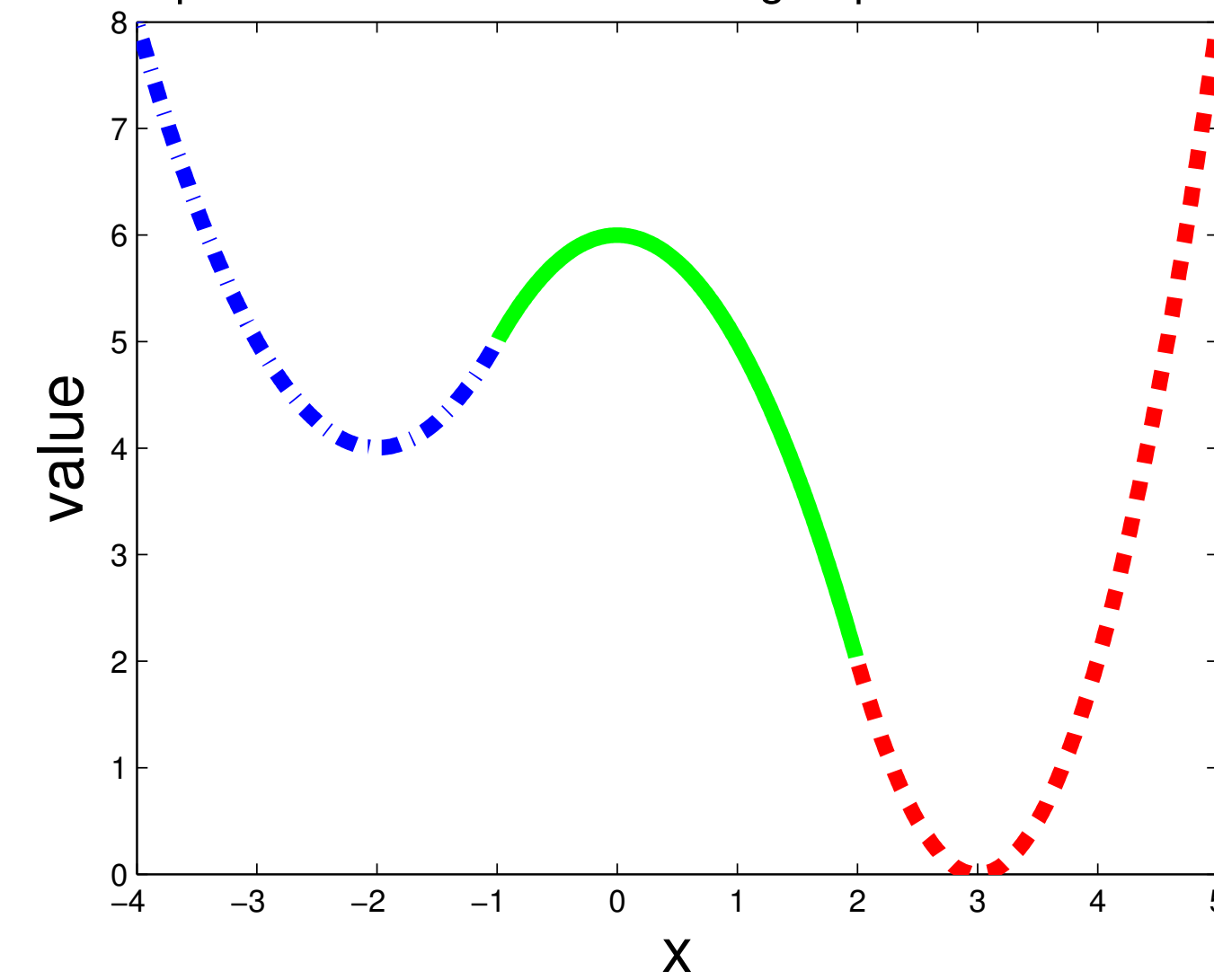
Sampled states and trajectories for the one link swingup problem after 10, 20, 30, 40, 50, and 60 states are stored.

Swingup Example: 2 links.

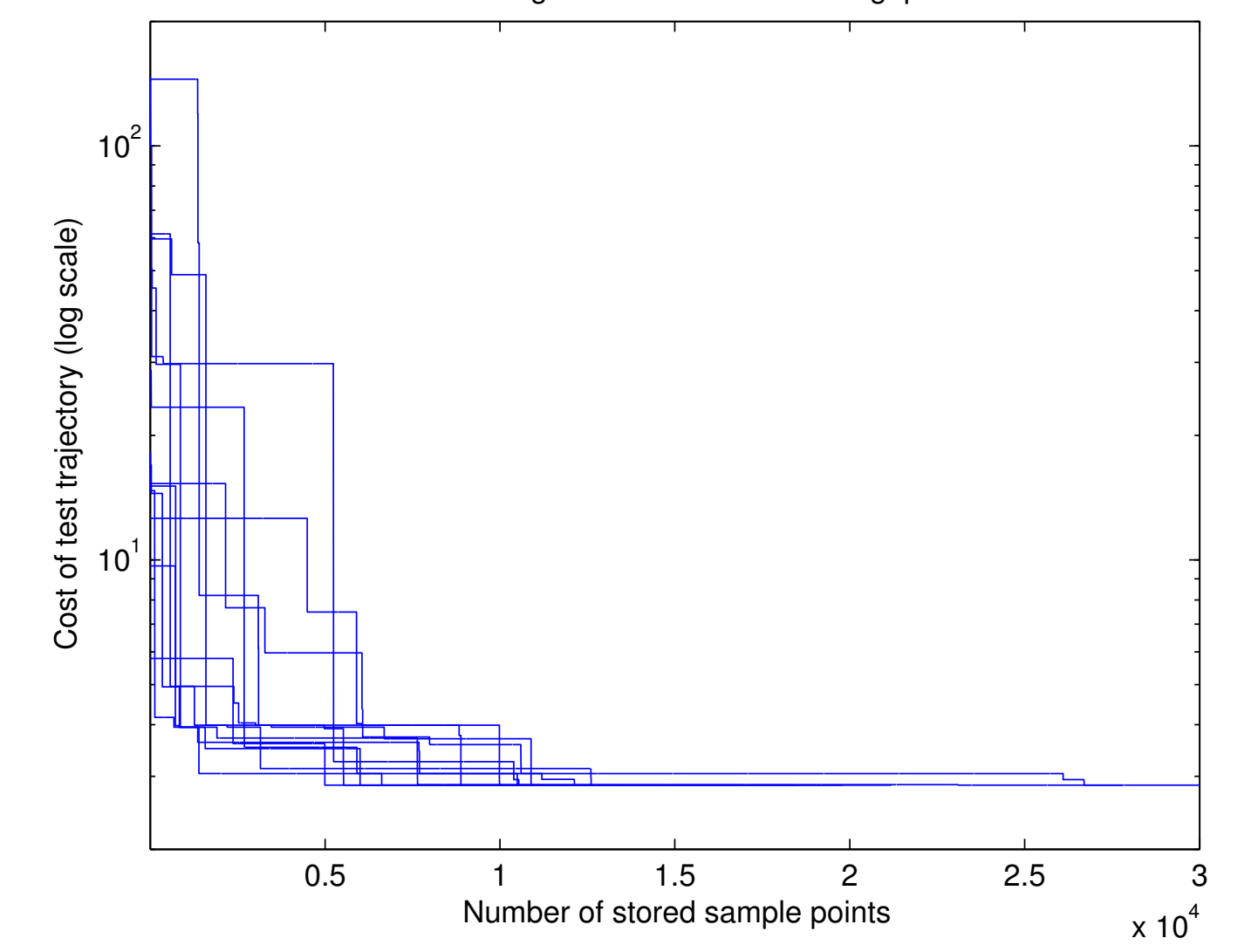


Configurations from the simulated two link pendulum optimal swing up trajectory every fifth of a second and the end of the trajectory.

Example 1D value function fit using 3 quadratic local models

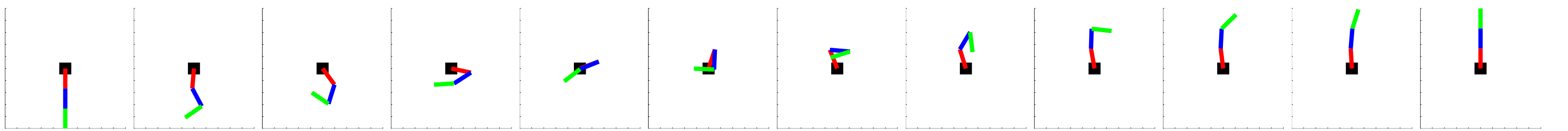


Convergence For Two Link Swingup



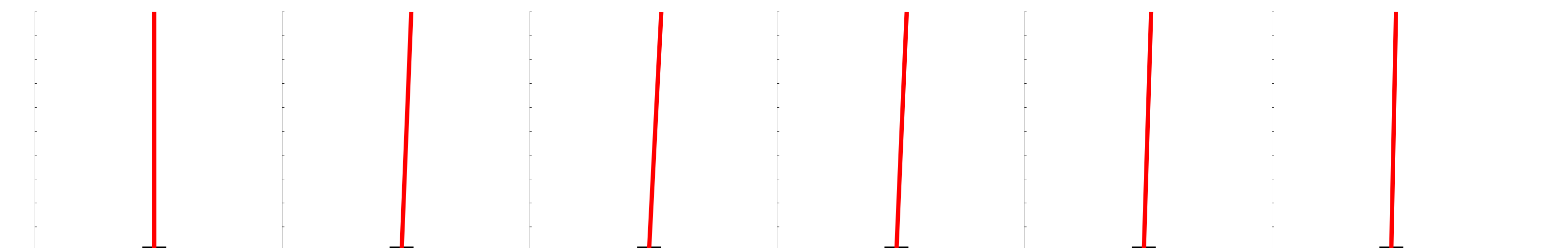
Left: Example of a local approximation of a 1D value function using three quadratic models. **Right:** Convergence curves for the two link case. These curves are obtained by testing the performance of the policy on the swingup problem after each new state is stored. 13 runs with different random number generator seeds are shown. The mean number of states stored at convergence is 11430. All but two of the runs converge after storing less than 13000 states, and all runs have converged after storing 27000 states. A 60x60x60x60 grid with almost 13 million states failed to find a trajectory as good as this one, while a 100x100x100x100 grid with 100 million states did find the same trajectory.

Swingup Example: 3 links.

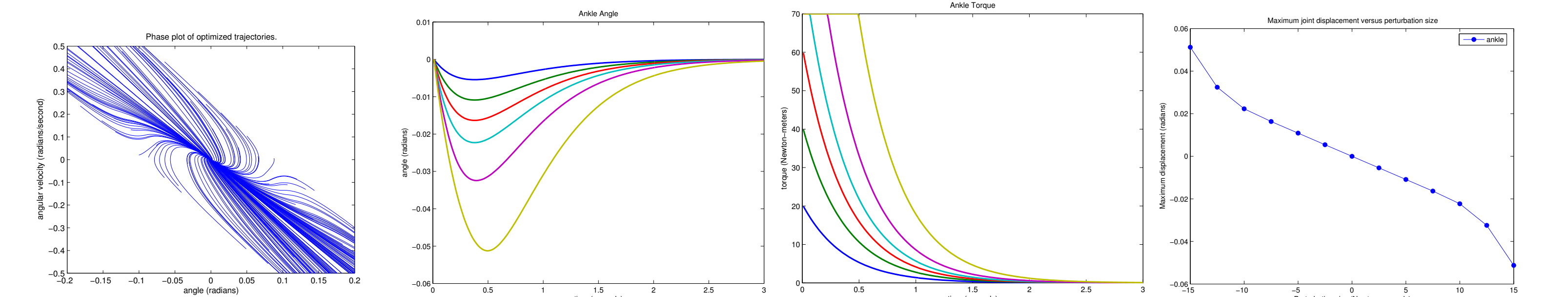


Configurations from the simulated three link pendulum optimal trajectory every tenth of a second and at the end of the trajectory.

Humanoid Balance Example: 1 link.

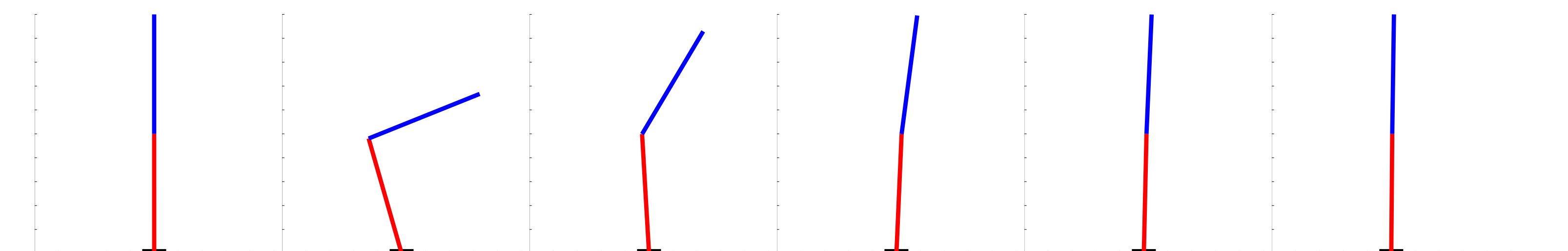


Configurations every quarter of a second of a simulated one link inverted pendulum response to an impulse of 15 Newton-seconds forward (to the right) The black rectangle indicates the extent of the symmetric foot.

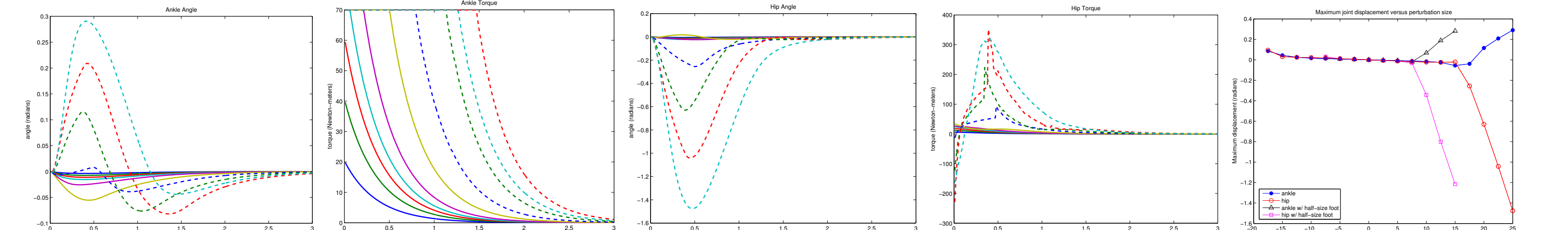


1: Optimized trajectories from different starting points. 2: Ankle angle trajectories for a range of perturbation sizes (2.5, 5, 7.5, 10, 12.5, and 15 Newton-seconds). 3: Ankle torque for the same range of perturbation sizes. 4: Maximum joint displacement versus perturbation size.

Humanoid Balance Example: 2 links.

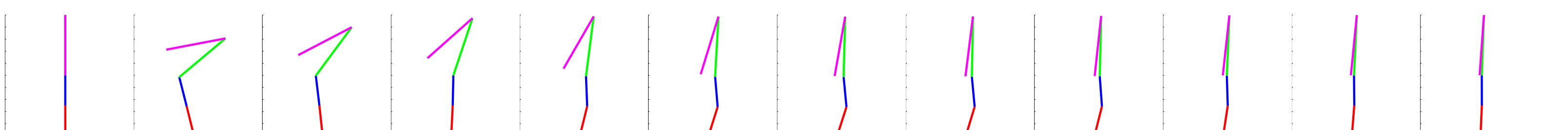


Configurations every half second of a simulated two link inverted pendulum response to an impulse of 25 Newton-seconds forward (to the right) The black rectangle indicates the extent of the symmetric foot.

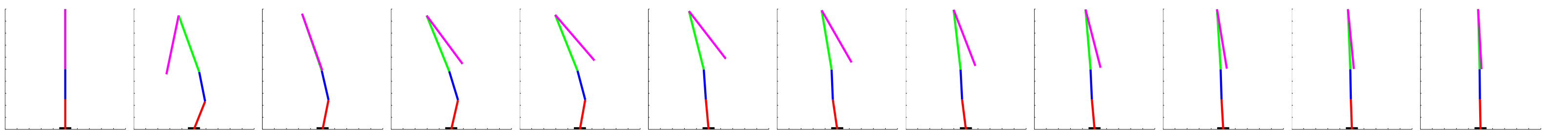


1: Ankle angle trajectories for a range of perturbation sizes (2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, and 25 Newton-seconds). 2: Ankle torque for the same range of perturbation sizes. 3: Hip angle trajectories for the same range of perturbation sizes. 4: Hip torque for the same range of perturbation sizes. 5: Maximum joint displacement versus perturbation size.

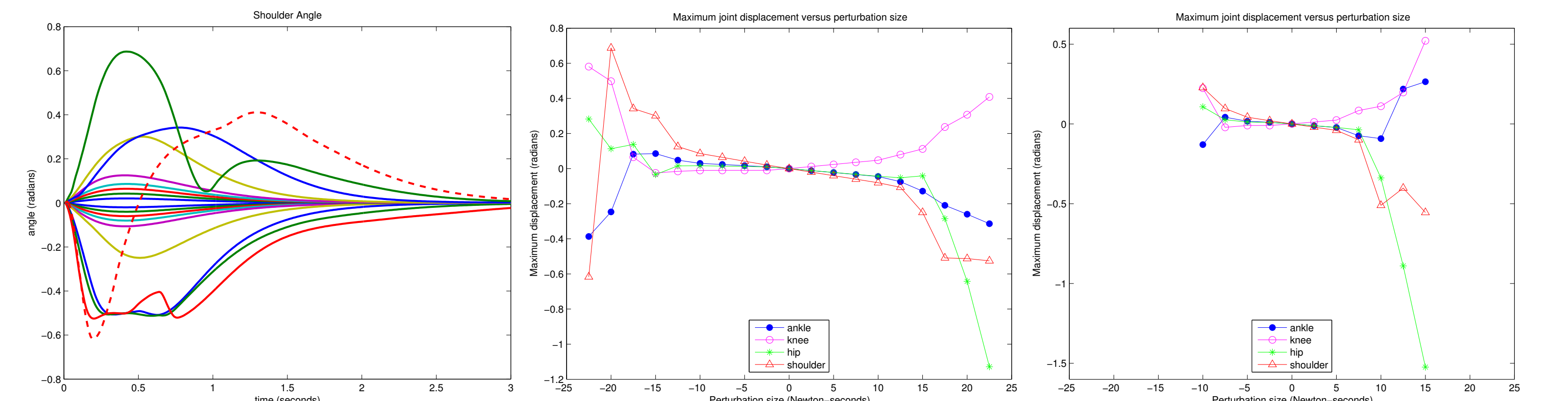
Humanoid Balance Example: 4 links.



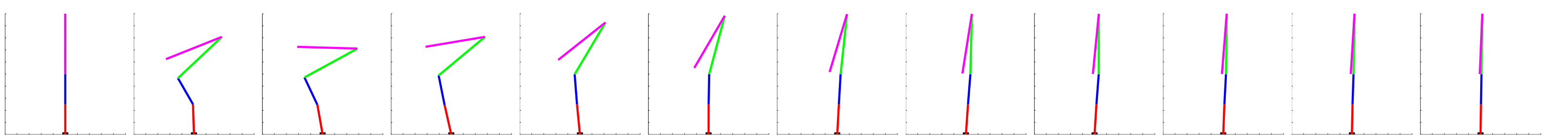
Configurations every quarter second from a simulated response to a forward impulse (to the right) of 22.5 Newton-seconds. The lower black rectangle indicates the extent of the symmetric foot.



Configurations every quarter second from a simulated response to a backward impulse (to the left) of 22.5 Newton-seconds



1: Shoulder responses to different size perturbations. 2 and 3: Maximum joint displacement versus perturbation size for a full sized foot (2) and a half sized foot (3).



Configurations every quarter second from a simulated response to a forward impulse of 15 Newton-seconds. The lower black rectangle indicates the extent of the half-sized foot.