# Working with standard optimization packages

Jonathan F.C. Herman
*Colorado Center for Astrodynamics Research*
*University of Colorado at Boulder*

Optimal Trajectories

# Overview

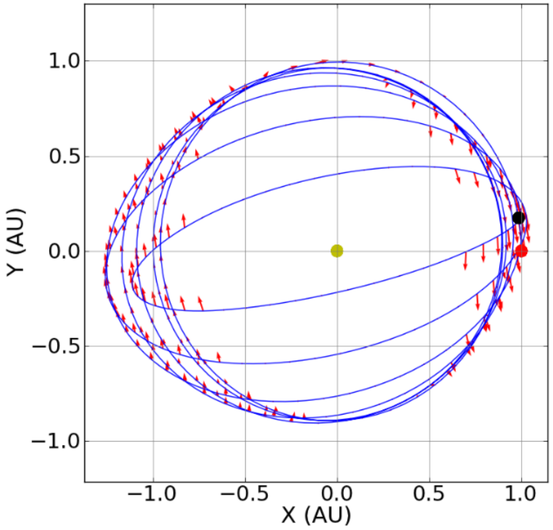
- Personal Background
- Common elements
  - General description
  - Toy problem formulation
  - Scaling
  - Sparsity
  - Local optima
- Optimizer differences
  - SNOPT vs IPOPT vs MATLAB (fmincon, …)
- Advanced example
- Project suggestion

# Personal background

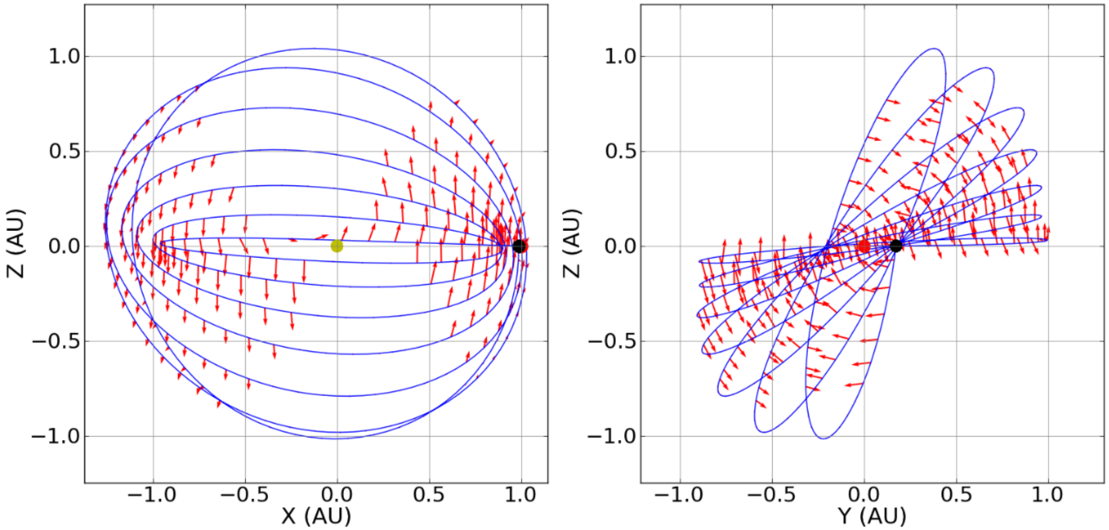
- Spent my master's thesis writing a Sims-Flanagan type low-thrust trajectory optimization tool
  - Essentially a watered down version of JPL's MALTO
  - Interfaced with SNOPT for optimization (same as MALTO)

- Used for optimizing far-side, highly inclined solar observatory trajectories

# Solar observatory




- 7.5 year time of flight
- Orbital period of ~ 1 year
- Final heliocentric inclination of 90 degrees
- C3 = 0, no gravity assists, all solar electric propulsion

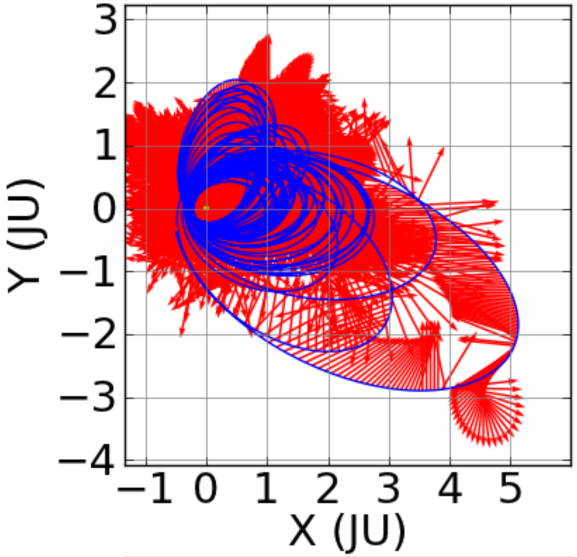


# Personal background

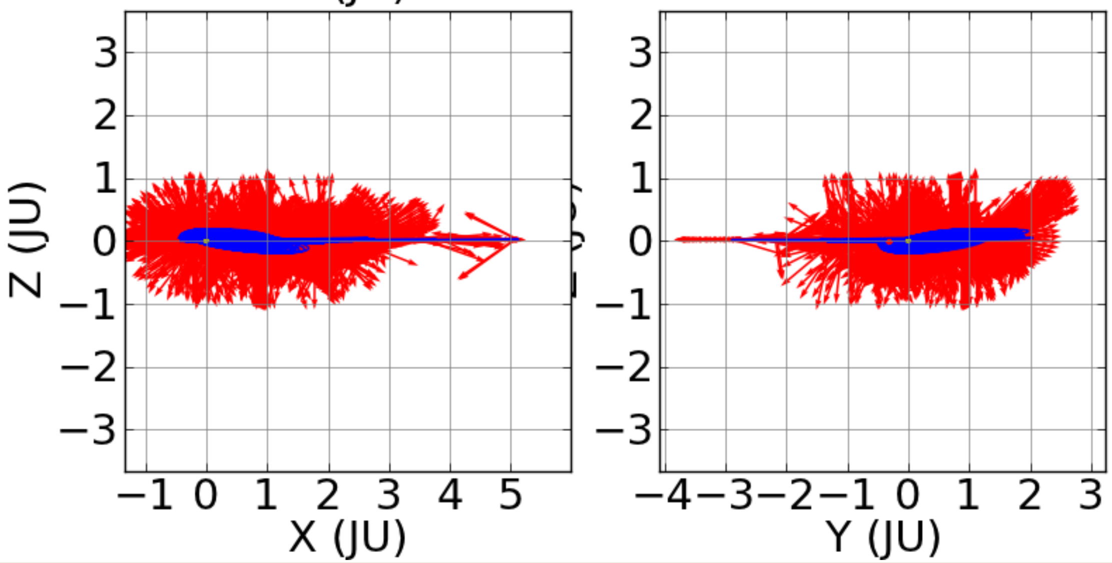
- Realized that even though this code was very specific in its application, it applied very readily to all sorts of problems
  - Human missions to near Earth asteroids
  - Jovian moon tour (GTOC 6)
  - “Warp drive” studies
  - Human fly-by mission of Mars
  - Robotic asteroid tours (GTOC 7)

- Current activities
  - Retired my original low-thrust code
  - Helping out developing a much more powerful implementation of the same method (championed by Stijn de Smet)
  - Working on effective methods for much more complex problems

# Jovian moon tour



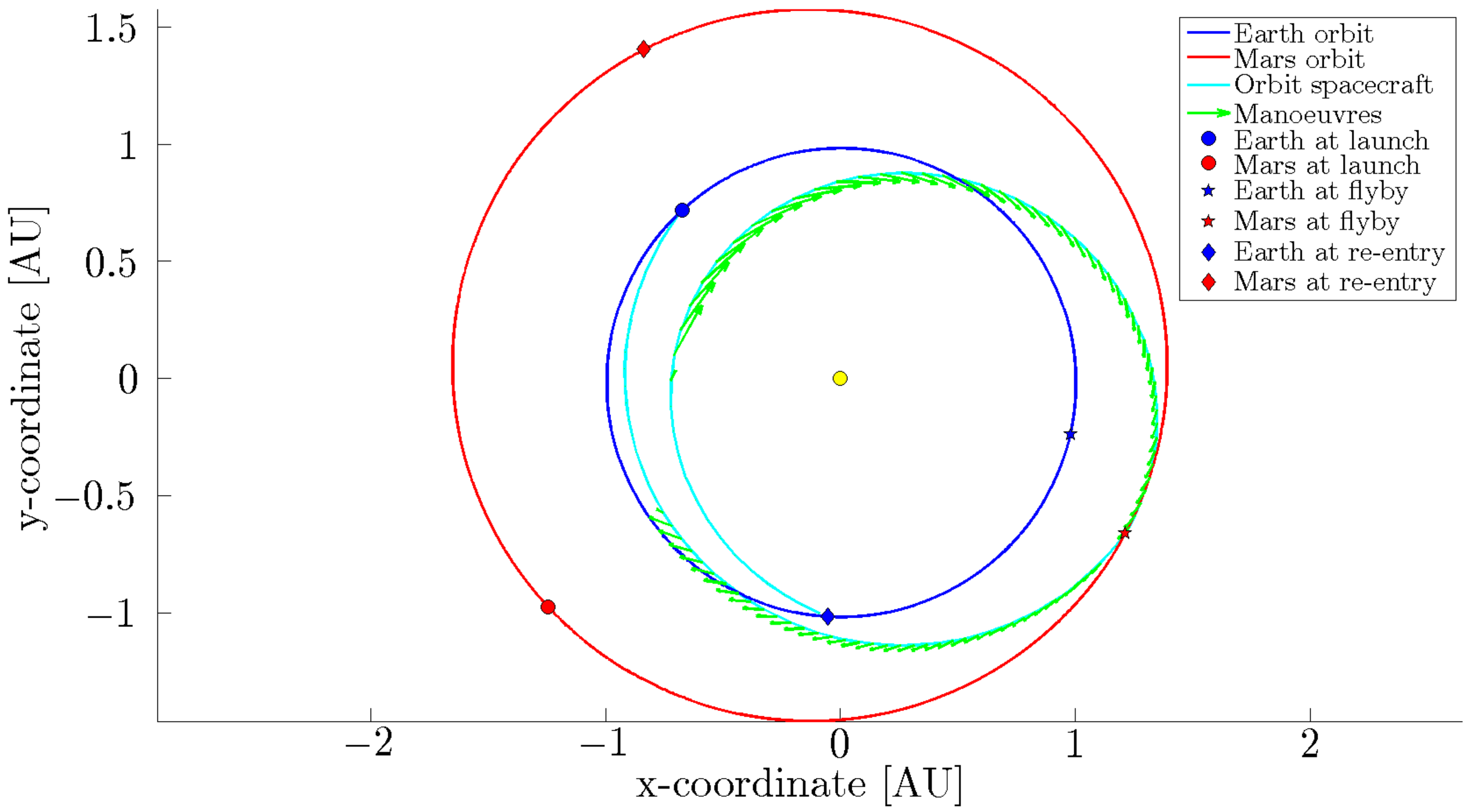

- GTOC6
- Scaling gone wrong…
- But it worked, and got us a very nice ranking!



# Mars fly-by mission

**Colorado Center for Astrodynamics Research**

University of Colorado
Boulder, Colorado



# Personal background



- Spent about 3 years working with SNOPT for a variety of problems & problem formulations
  - And about a month with IPOPT
- Racked up a long list of do's and don'ts over that time
- Hoping to save you all some time by discussing them today
  - Very implementation focused
  - Plenty of published text on the theory behind these tools, easily found through google (or contact me)

# Optimization tools

# Common elements

- At a high level, most optimization libraries are very similar
  - Enormous differences can exist in underlying theory / preferred interface / available options / actual performance
  - Besides those, there are some common features that, if not properly understood, can dominate the effects above
  - The interfaces are ultimately also very similar

- Before we get to some specific differences, its helpful to go over some of these potentially dominant features

# General description

- Parameters (1 x n) $x^L \leq x \leq x^U$
- Cost function (1 x 1) $f(x)$
- Constraints (m x 1) $g^L \leq g \leq g^U$
- Cost derivatives (1 x n) $\nabla f(x) = \frac{\delta f}{dx}$
- Jacobian (m x n) $\nabla g(x) = \frac{\delta g}{dx}$

# General description

- For the purposes of this talk, it is okay to assume that the provided information (constraints, objective, derivatives) are used in a process similar to how a Newton solver is iteratively used to find a root
  - This is obviously an enormous simplification, but there are actually great similarities in these processes

$$X_1 = X_0 - \frac{f(X_0)}{f'(X_0)}$$

- Key take-away:
  - The solver will iteratively improve the optimization state, all the time computing the current value of the objective/constraints, and their derivatives, to determine the state for the next iteration

# General description

- Some solvers also allow the use of second order derivatives
  - Can be a useful feature, but can also be very memory intensive
  - It doesn't really change the aspects discussed today, so will largely be ignored here

- The formulation so far is very generic
  - As expected, smooth functions for the constraints/objective lead to better convergence in this approach
  - Adjoint equations need not be in user formulation
  - Since they are continuous functions, they could be though (and some trajectory optimization tools do this)
  - Typically though, these tools are used for direct optimization

# Toy problem

- To give some concrete examples, we'll work with a toy problem
  - 3 parameters, 2 constraints
  - Designed for instructive/demonstrative value
  - It (probably?) doesn't solve anything useful
  - It should still have a name…

# Toy problem

- To give some concrete examples, we'll work with a toy problem
  - 3 parameters, 2 constraints
  - Designed for instructive/demonstrative value
  - It (probably?) doesn't solve anything useful
  - It should still have a name…

- We will dub this "The POPSICLE-problem"

**The <u>P</u>ointless <u>O</u>ptimization <u>P</u>roblem**
**<u>S</u>o <u>I</u> <u>C</u>an <u>L</u>earn <u>E</u>very-problem**

# POPSICLE-problem formulation

Colorado Center for Astrodynamics Research

University of Colorado
Boulder, Colorado

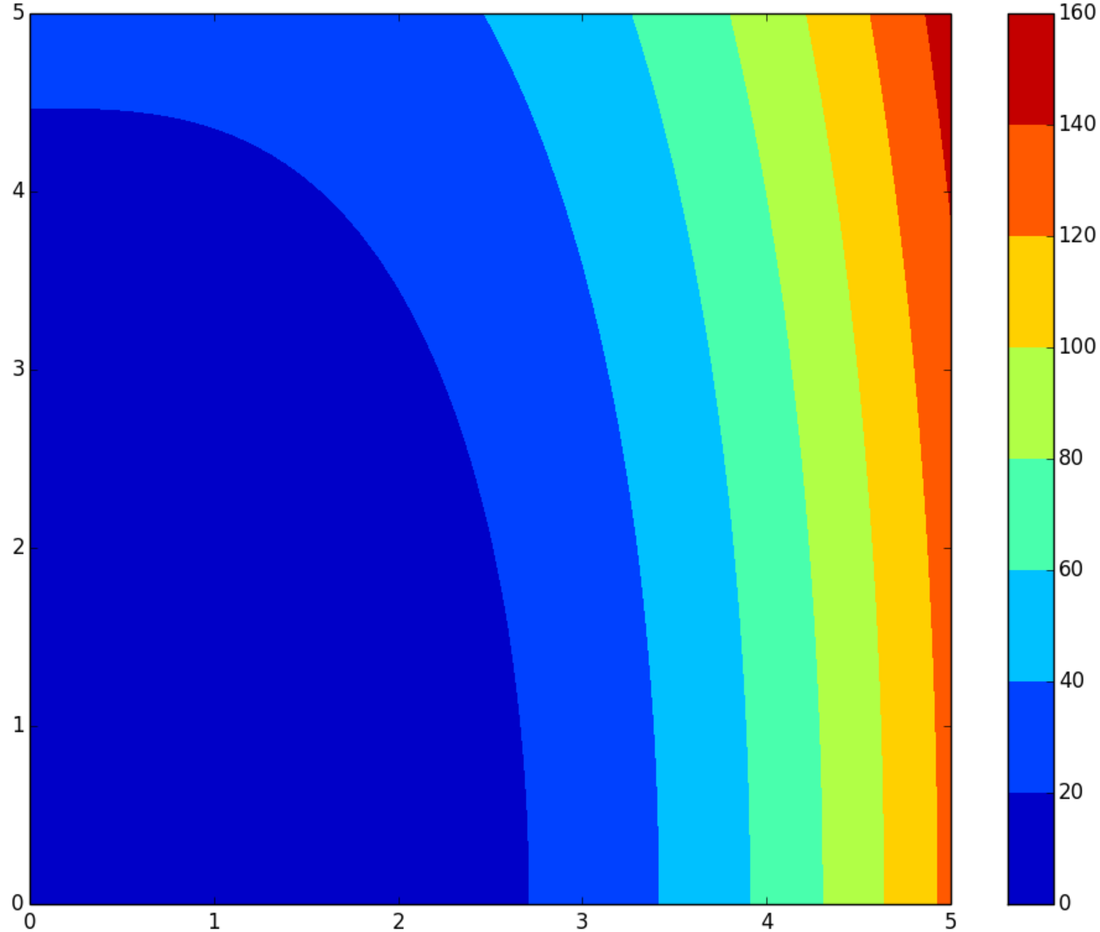
- Parameters (1 x 3) $$\begin{bmatrix}0\\0\\0\end{bmatrix}^T \le \begin{bmatrix}x_0\\x_1\\x_2\end{bmatrix}^T \le \begin{bmatrix}5\\5\\5\end{bmatrix}^T$$
- Cost function (1 x 1) $$f(x) = x_0^2 + x_1^3 + x_2^2$$
- Constraints (2 x 1) $$\begin{bmatrix}0\\30\end{bmatrix} \le \begin{bmatrix}g_1 = x_0^3 + x_2\\g_2 = x_1^2 + x_2^3\end{bmatrix} \le \begin{bmatrix}2.5\\100\end{bmatrix}$$
- Cost derivatives (1 x 3) $$\nabla f(x) = \begin{bmatrix}2x_0 & 3x_1^2 & 2x_2\end{bmatrix}$$
- Jacobian (2 x 3) $$\nabla g(x) = \begin{bmatrix}3x_0^2 & 0 & 1\\0 & 2x_1 & 3x_2^2\end{bmatrix}$$

# Cost function

**Colorado Center for Astrodynamics Research**

University of Colorado
Boulder, Colorado

$$x_0 = 0$$
$$f(x) = x_0^2 + x_1^3 + x_2^2$$

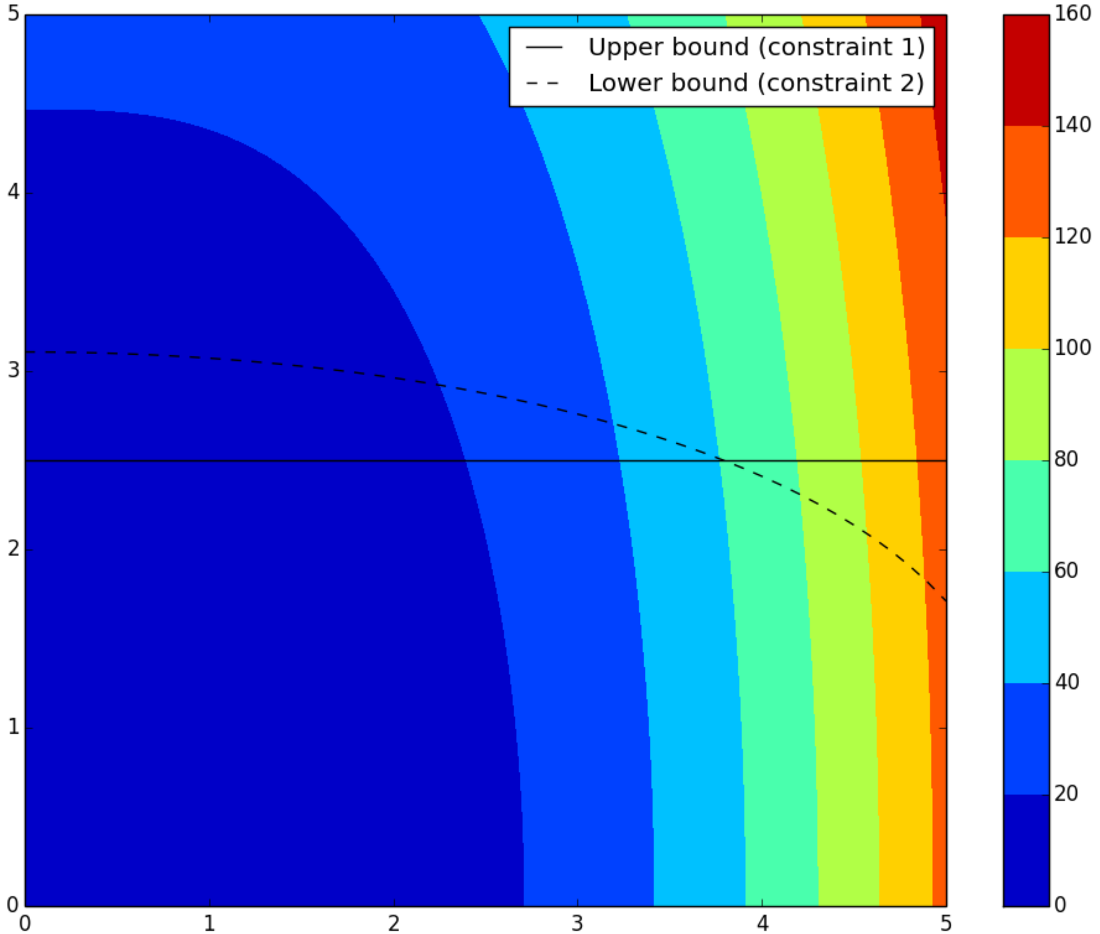


# Cost function & Constraints

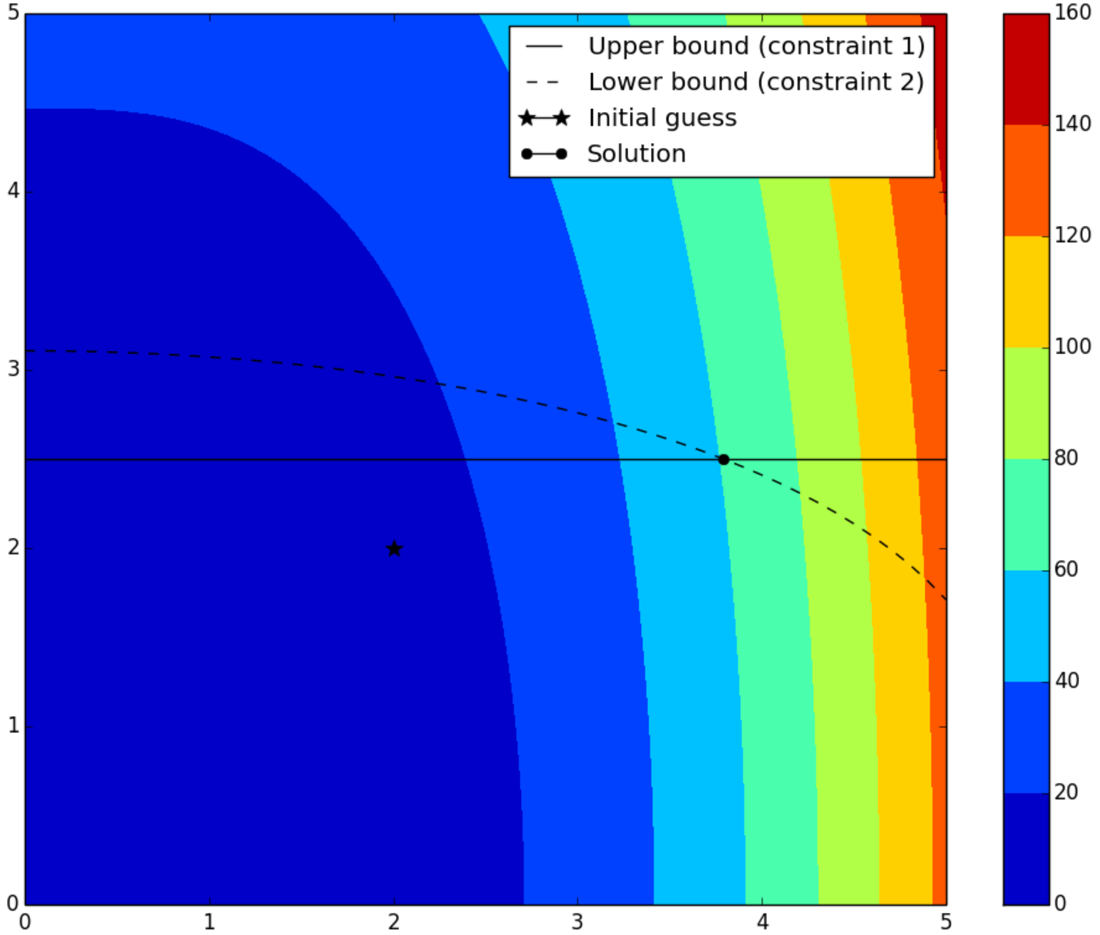
$$x_0 = 0$$

$$\begin{bmatrix} g_1 = x_0^3 + x_2 \\ g_2 = x_1^2 + x_2^3 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 30 \end{bmatrix} \leq \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \leq \begin{bmatrix} 2.5 \\ 100 \end{bmatrix}$$



# Optimal solution



# The importance of scaling!

# POPSICLE-problem formulation

- Parameters (1 x 3)

$$\begin{bmatrix}0\\0\\0\end{bmatrix}^T \le \begin{bmatrix}x_0\\x_1\\x_2\end{bmatrix}^T \le \begin{bmatrix}5\\5\\5\end{bmatrix}^T$$

- Cost function (1 x 1)

$$f(x) = x_0^2 + x_1^3 + x_2^2$$

- Constraints (2 x 1)

$$\begin{bmatrix}0\\30\end{bmatrix} \le \begin{bmatrix}g_1 = x_0^3 + x_2\\g_2 = x_1^2 + x_2^3\end{bmatrix} \le \begin{bmatrix}2.5\\100\end{bmatrix}$$

- Cost derivatives (1 x 3)

$$\nabla f(x) = \begin{bmatrix}2x_0 & 3x_1^2 & 2x_2\end{bmatrix}$$

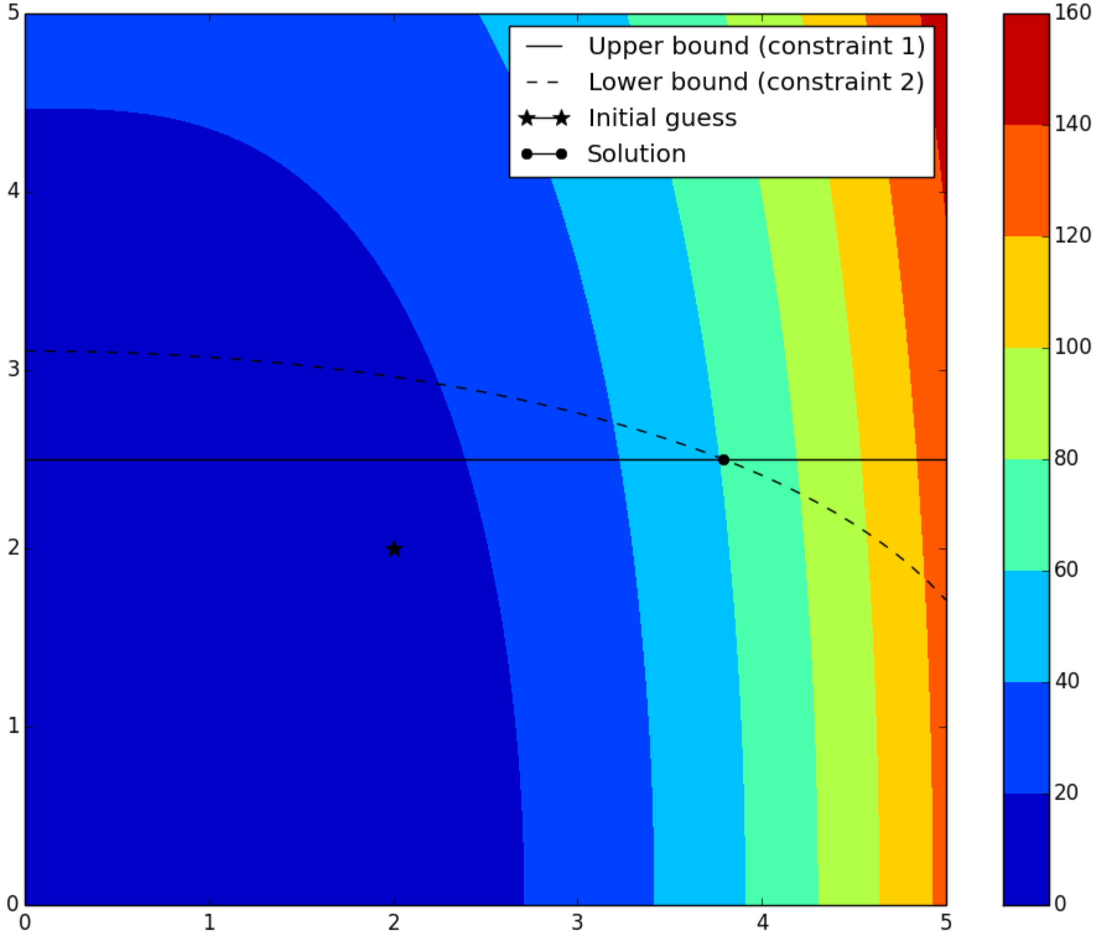
- Jacobian (2 x 3)

$$\nabla g(x) = \begin{bmatrix}3x_0^2 & 0 & 1\\0 & 2x_1 & 3x_2^2\end{bmatrix}$$

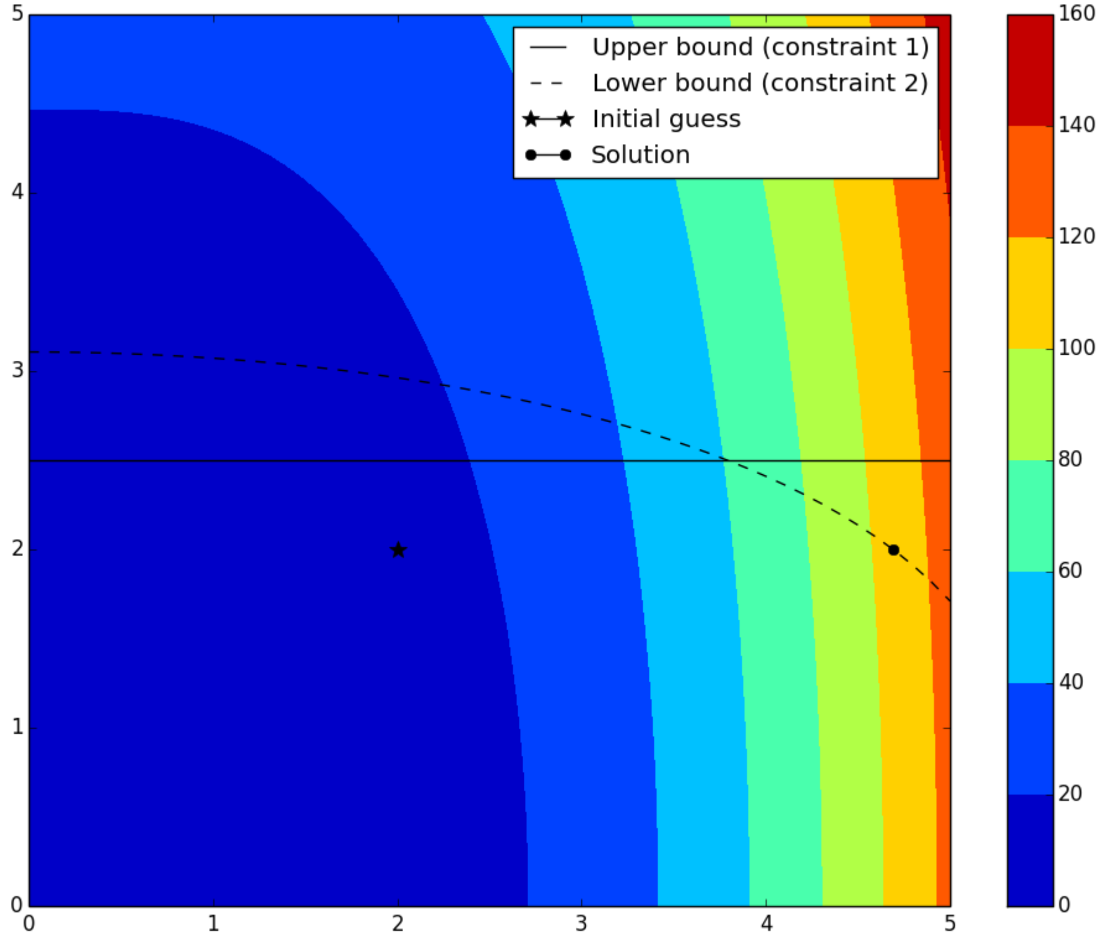
# Scaling example

- Parameters (1 x 3) $$\begin{bmatrix}0\\0\\0\end{bmatrix}^T \leq \begin{bmatrix}x_0\\x_1\\x_2\end{bmatrix}^T \leq \begin{bmatrix}5\\5\cdot 10^7\\5\end{bmatrix}^T$$
- Cost function (1 x 1) $$f(x) = x_0^2 + (x_1 \cdot 10^{-7})^3 + x_2^2$$
- Constraints (2 x 1) $$\begin{bmatrix}0\\30\end{bmatrix} \leq \begin{bmatrix}g_1 = x_0^3 + x_2\\g_2 = (x_1 \cdot 10^{-7})^2 + x_2^3\end{bmatrix} \leq \begin{bmatrix}2.5\\100\end{bmatrix}$$
- Cost derivatives (1 x 3) $$\nabla f(x) = \begin{bmatrix}2x_0 & 3{x_1}^2 \cdot (10^{-7})^3 & 2x_2\end{bmatrix}$$
- Jacobian (2 x 3) $$\nabla g(x) = \begin{bmatrix}3x_0^2 & 0 & 1\\0 & 2x_1 \cdot (10^{-7})^2 & 3x_2^2\end{bmatrix}$$

# Optimal solution



# Poorly scaled solution



# Scaling

- Clearly, this arbitrary scale factor of $10^{-7}$ is absurd, right?
  - Sure, except… $\frac{V_{Earth}}{AU} \approx \frac{30\ km/s}{150 \cdot 10^6\ km} = 2 \cdot 10^{-7}$
- **Scaling. Is. Everything.** It can (or rather; It will)…
  - Make a feasible solution appear infeasible
  - Make an infeasible solution appear feasible
  - Change runtimes by orders of magnitude
  - Determine whether or not the problem converges at all
  - Give you enormous headaches
- Scale appropriately!
  - **Everything** should be in the [0,1] domain (or [-0.5,0.5])
  - That means parameters, constraints, and objective function
  - You can deviate to weight certain aspects, **but tread carefully!**

# Sparsity

- Most optimization problems have many derivatives that are always zero → Sparsity

- An important way to reduce memory usage
  - Which can also lead to improved runtimes

- Easy to show implementation with the POPSICLE-problem
  - Harder to show importance

- Will follow the sparse POPSICLE-problem with a real example

# Sparsity: IPOPT format

- Actual Jacobian

$$\nabla g(x) = \begin{bmatrix} 3x_0^2 & 0 & 1 \\ 0 & 2x_1 & 3x_2^2 \end{bmatrix}$$

# Sparsity: IPOPT format

- Actual Jacobian

$$\nabla g(x) = \begin{bmatrix} 3x_0^2 & 0 & 1 \\ 0 & 2x_1 & 3x_2^2 \end{bmatrix}$$

- Input of dense (full) Jacobian

$$\begin{aligned} \nabla g(x) &= [3x_0^2 \quad 0 \quad 1 \quad 0 \quad 2x_1 \quad 3x_2^2] \\ iRow &= [0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1] \\ jCol &= [0 \quad 1 \quad 2 \quad 0 \quad 1 \quad 2] \end{aligned}$$

# Sparsity: IPOPT format

- Actual Jacobian

$$\nabla g(x) = \begin{bmatrix} 3x_0^2 & 0 & 1 \\ 0 & 2x_1 & 3x_2^2 \end{bmatrix}$$

- Input of dense (full) Jacobian

$$\nabla g(x) = [3x_0^2 \quad 0 \quad 1 \quad 0 \quad 2x_1 \quad 3x_2^2]$$
$$iRow = [0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1]$$
$$jCol = [0 \quad 1 \quad 2 \quad 0 \quad 1 \quad 2]$$

- Input of sparse Jacobian

$$\nabla g(x) = [3x_0^2 \quad 1 \quad 2x_1 \quad 3x_2^2]$$
$$iRow = [0 \quad 0 \quad 1 \quad 1]$$
$$jCol = [0 \quad 2 \quad 1 \quad 2]$$

# Sparsity

- Example from my own research
- Low-thrust trajectory split into segments
  - Each segment contains a number of parameters & constraints
  - Sparsity grows (almost) linearly with problem size

| Segments | 100 | 1000 | 10 000 |
|---|---|---|---|
| Non-zero Jacobian elements (%) | 1.8 | 0.18 | 0.018 |
| Full Jacobian (megabyte) | 7.1 | 700 | 70 000 |
| Sparse Jacobian (megabyte) | 0.1 | 1.3 | 12.6 |

- Sparsity matters!

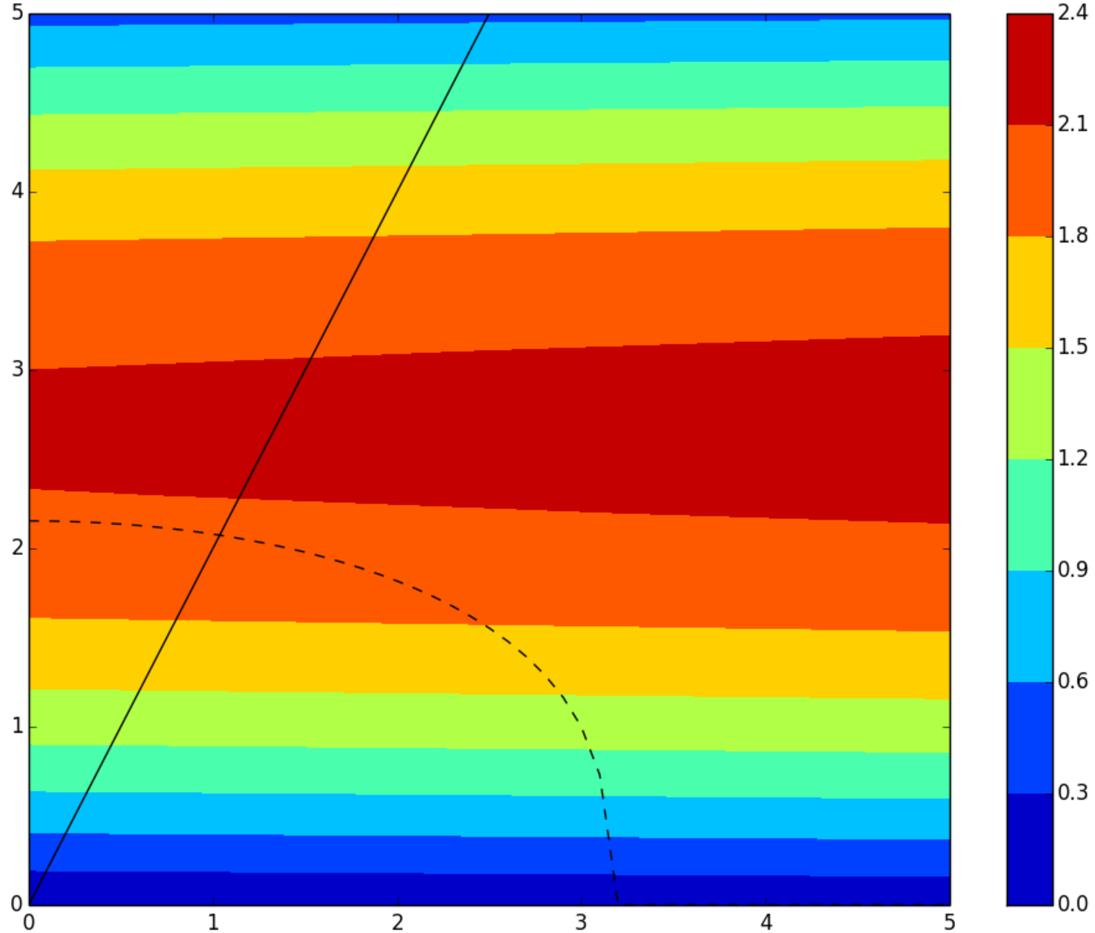
# Local optima

# POPSICLE-2

- Parameters (1 x 3) $$\begin{bmatrix}0\\0\\0\end{bmatrix}^T \leq \begin{bmatrix}x_0\\x_1\\x_2\end{bmatrix}^T \leq \begin{bmatrix}5\\5\\5\end{bmatrix}^T$$
- Cost function (1 x 1) $$f(x) = x_0 + 0.01x_1 + x_2(1.6 - 0.3x_2)$$
- Constraints (2 x 1) $$\begin{bmatrix}-10\\10\end{bmatrix} \leq \begin{bmatrix}g_1 = -2x_1 + x_2\\ g_2 = x_1^2 + x_2^3\end{bmatrix} \leq \begin{bmatrix}0\\100\end{bmatrix}$$
- Cost derivatives (1 x 3) $$\nabla f(x) = \begin{bmatrix}1 & 0.01 & 1.6 - 0.6x_2\end{bmatrix}$$
- Jacobian (2 x 3) $$\nabla g(x) = \begin{bmatrix}0 & -2 & 1\\ 0 & 2x_1 & 3x_2^2\end{bmatrix}$$

# Problem space

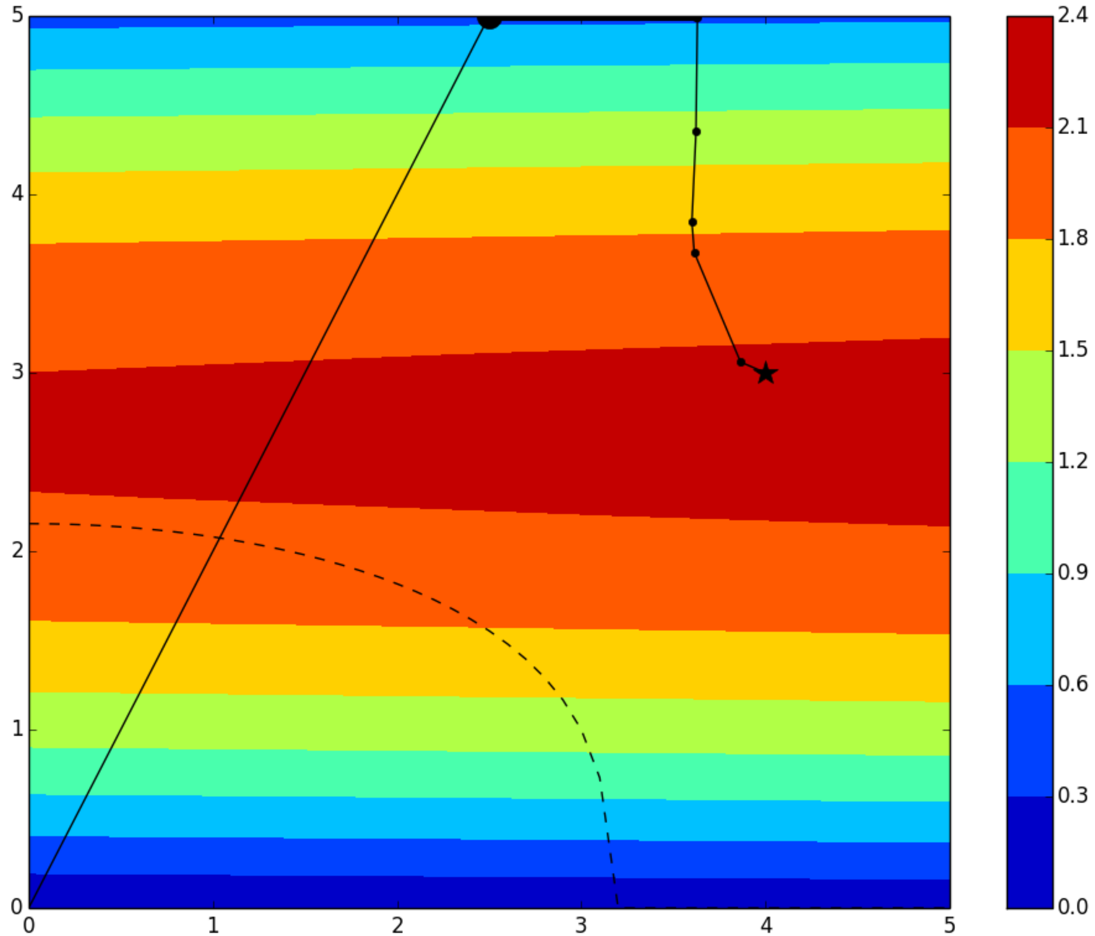
$$f(x) = x_0 + 0.01x_1 + x_2(1.6 - 0.3x_2)$$



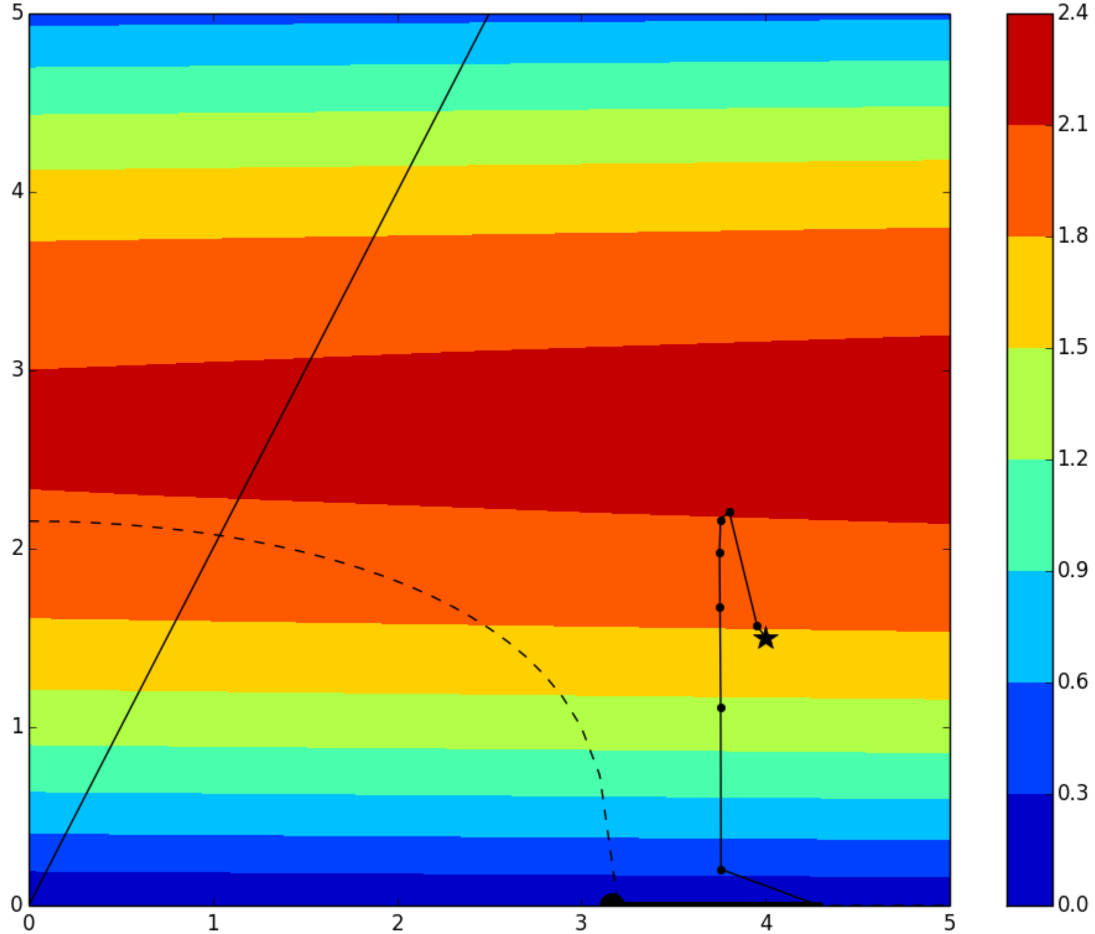

$$\begin{bmatrix} g_1 = -2x_1 + x_2 \\ g_2 = x_1^2 + x_2^3 \end{bmatrix}$$

$$\begin{bmatrix} -10 \\ 10 \end{bmatrix} \leq \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 100 \end{bmatrix}$$

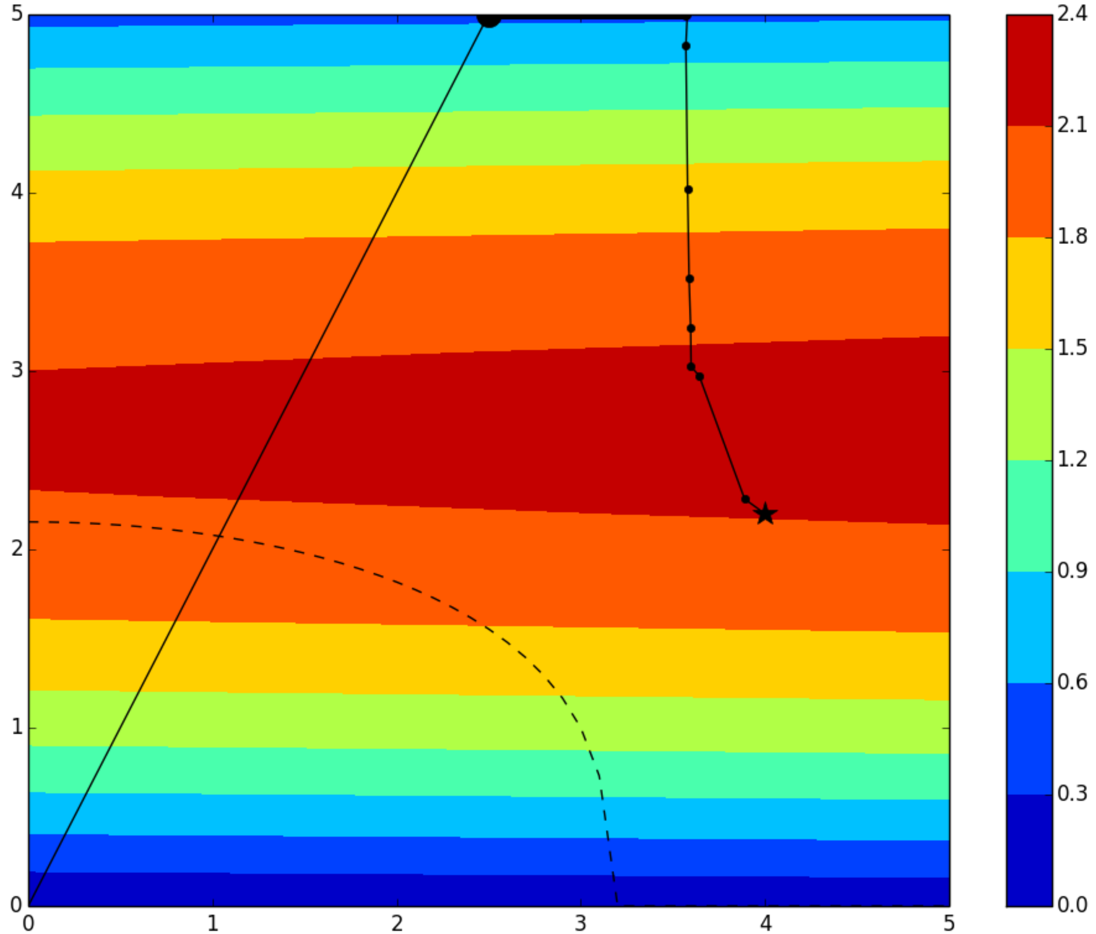
# Local optimum



# Optimal solution



# Crossing to local optimum



# Code examples

**Colorado Center for Astrodynamics Research**

University of Colorado
Boulder, Colorado

(Switch to code)

# Optimizer overview

- SNOPT
- IPOPT
- MATLAB (fmincon, …)
- Many others exist
  - GPU-based
  - …

# SNOPT

- SNOPT (Sparse Nonlinear OPTimizer)
  - Written in Fortran 77 (Fortran 2003 version in development)
  - Interfaces exist for C, C++, Fortran, MATLAB, Python, …
  - CU has an academic license to a fairly recent version
  - Sparsity notation a little more cumbersome than IPOPT
  - No parallel solving, and only first order derivatives
  - Performance generally excellent
  - Has become somewhat of an industry standard in optimization
  - Assumes/prefers you install on Linux…
  - Website: http://ccom.ucsd.edu/~optimizers/

# IPOPT

- IPOPT (Interior Point OPTimizer)
  - Written in C++
  - Interfaces exist for C, C++, Fortran, MATLAB, Python, …
  - Completely functional as open source distribution
  - Additional capability available through (academic) licenses
  - Very modular in design, essentially a wrapper for several solvers
  - Support for parallel solving & second order derivatives
  - Sparsity notation extremely straightforward
  - Seems to perform somewhat-to-significantly better than SNOPT
  - Very widely used (and seems to be growing)
  - EXCELLENT documentation & community support
  - Assumes/prefers you install on Linux…
  - Website: http://www.coin-or.org/Ipopt/

# fmincon



- fmincon (MATLAB)
  - Easy to get access to (it's in MATLAB…done)
  - MATLAB probably also has other solvers
  - No personal experience
  - Similarly in its setup/functionality to SNOPT/IPOPT
  - It does support sparse matrices
  - Might be perfect for a project, probably not for research
  - Website: http://www.mathworks.com/help/optim/ug/fmincon.html

# Finite differencing

**Colorado Center for Astrodynamics Research**

University of Colorado
Boulder, Colorado

- Instead of the user defining the Jacobian, some optimizers offer the functionality of finite differencing the objective & constraint functions
  - Generally worthless, since it comes at tremendous computational cost and is less accurate than analytical derivatives
  - It does allow for very quick implementations of problem formulations, so there is some value during prototyping

- Similarly, most optimizers can check the user-provided derivatives through finite differencing
  - This is **invaluable**, since it allows for (comparatively) easy identification of scaling errors, indexing errors, and 100 other errors you didn't know were even possible…

# The joys of compilation

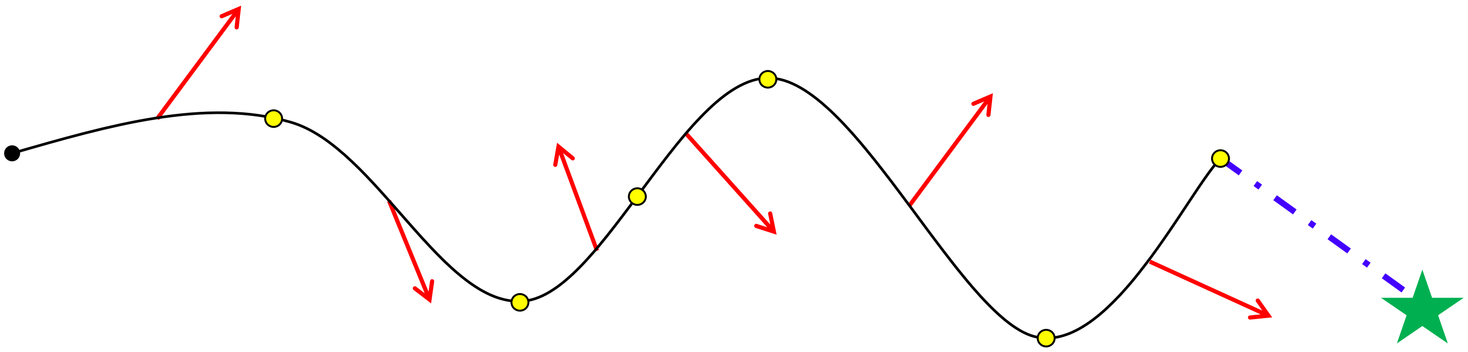
**Colorado Center for Astrodynamics Research**
University of Colorado
Boulder, Colorado

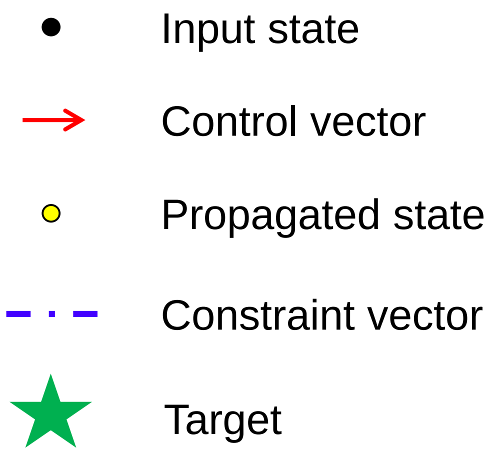
- It can be a chore to get SNOPT/IPOPT to compile
  - Especially if you are not extremely familiar with Fortran/C++
- Be prepared: don't plan to do this when you are highly pressed for time
- Doing this in Linux will typically save a lot of work
- I'd be happy to assist, but my abilities to do so are entirely experience based and include large gaps of knowledge
  - Like you, I'm not a computer scientist, and just want to get it running
- Just for this reason, it might be preferable to work with fmincon for this class

# Research example

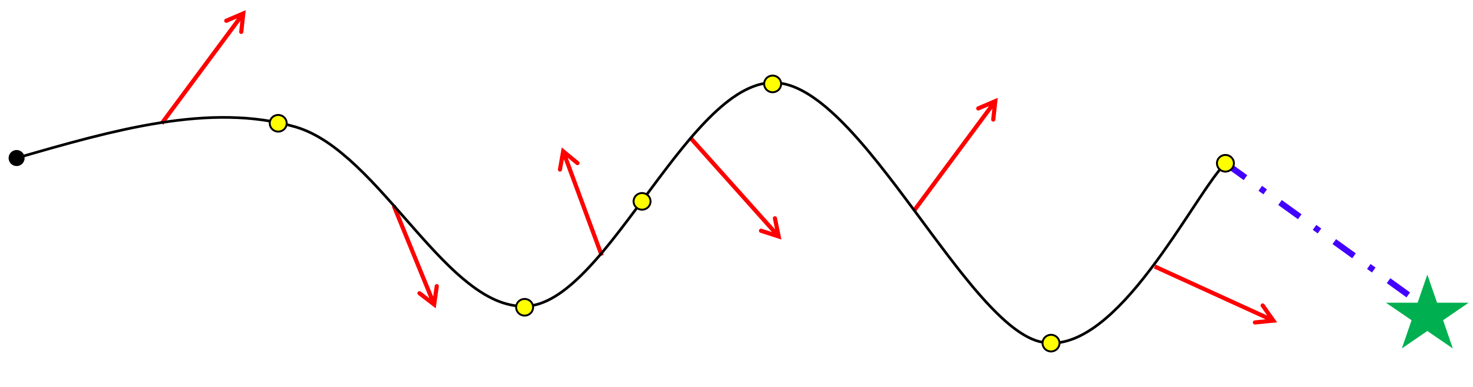
- Low-thrust trajectory optimization
- Direct transcription through Gauss-Lobatto collocation
- Some appealing aspects…
  - Very direct control over problem
  - Extremely sparse Jacobian
  - Embarrassingly parallel formulation
- Has proven to be very effective with large and/or complex problems

# Single shooting





# Single shooting



Input state

Control vector

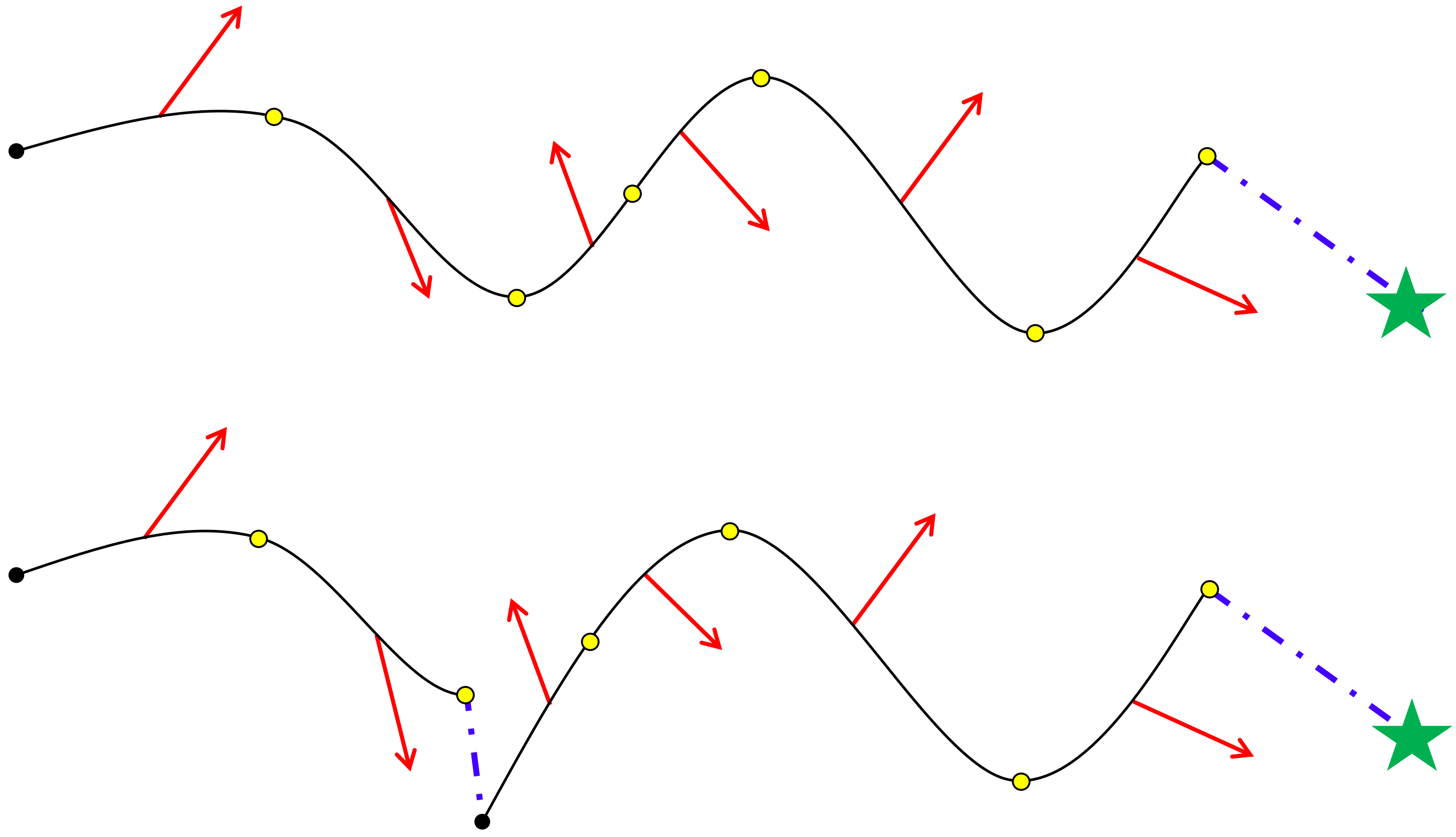
→ **<u>This is all you directly control</u>**

Propagated state

Constraint vector

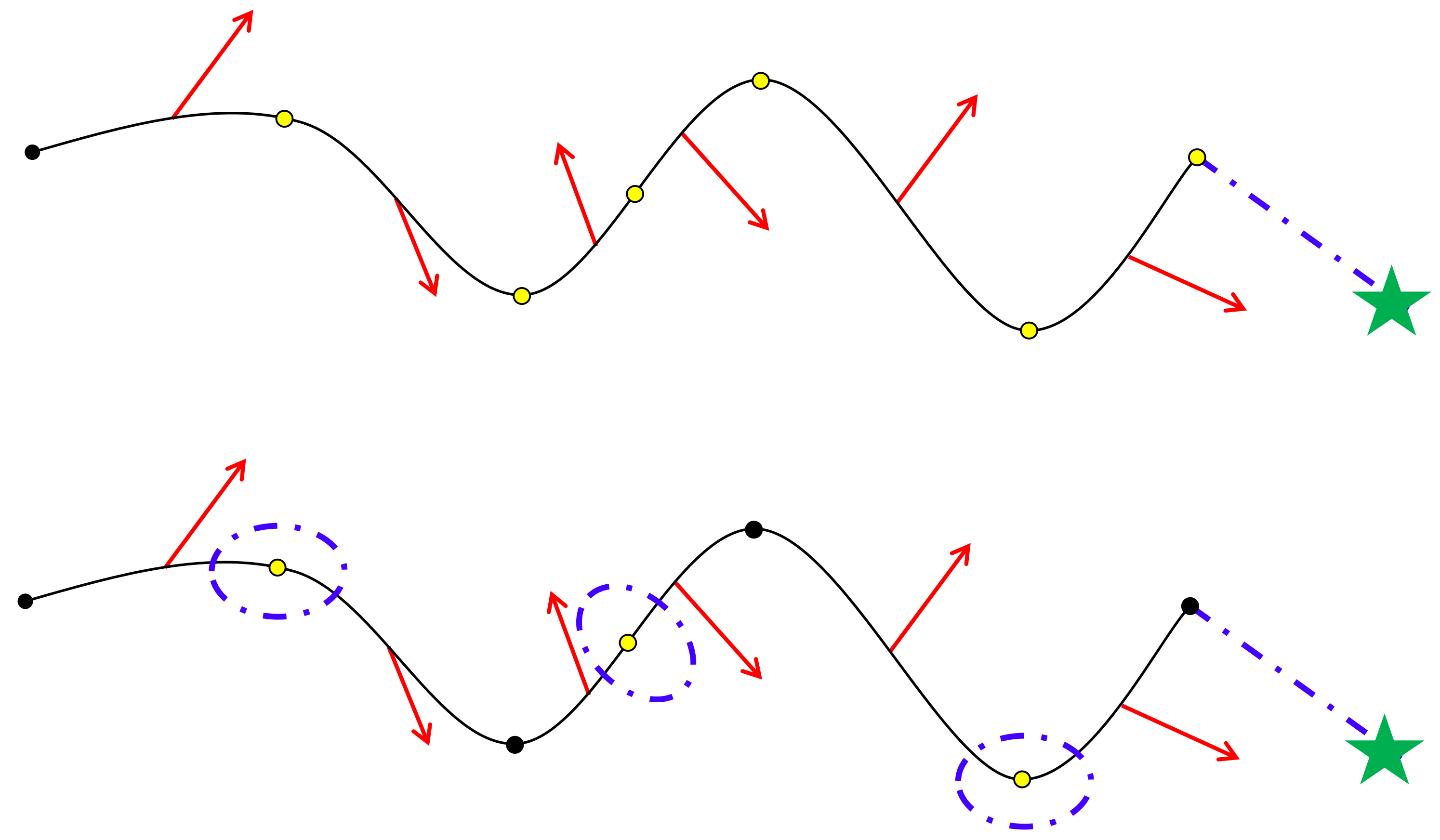
Target

# Multiple shooting



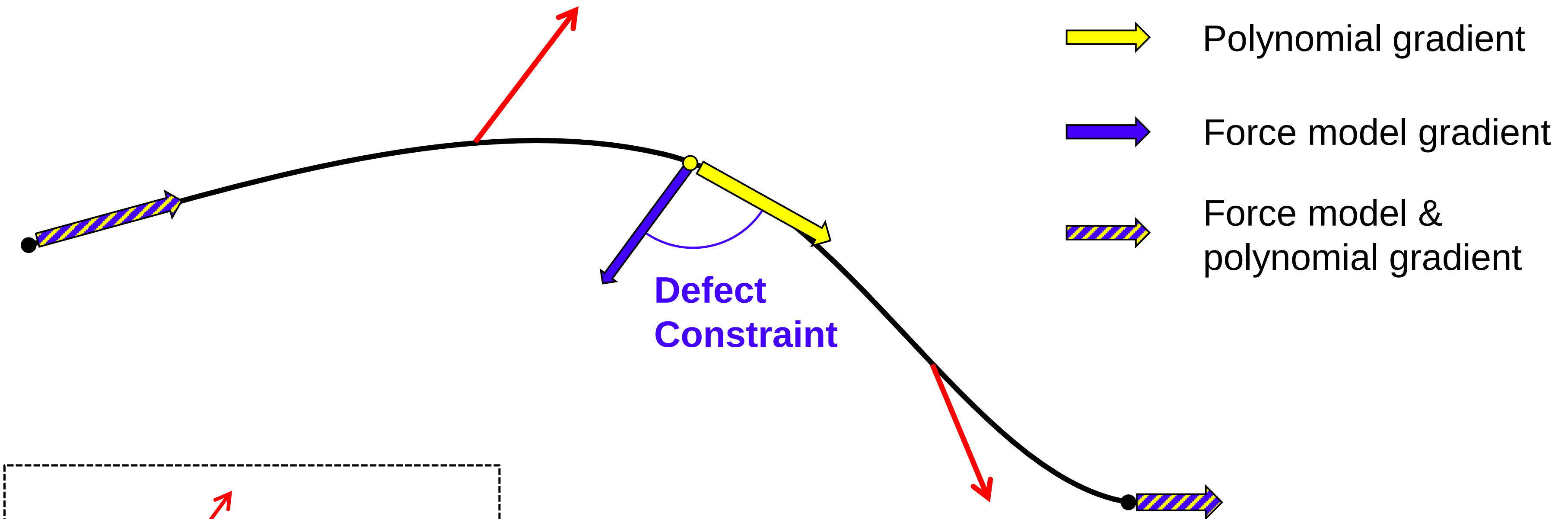
● Input state → Control vector ○ Propagated state — · — Constraint vector ★ Target

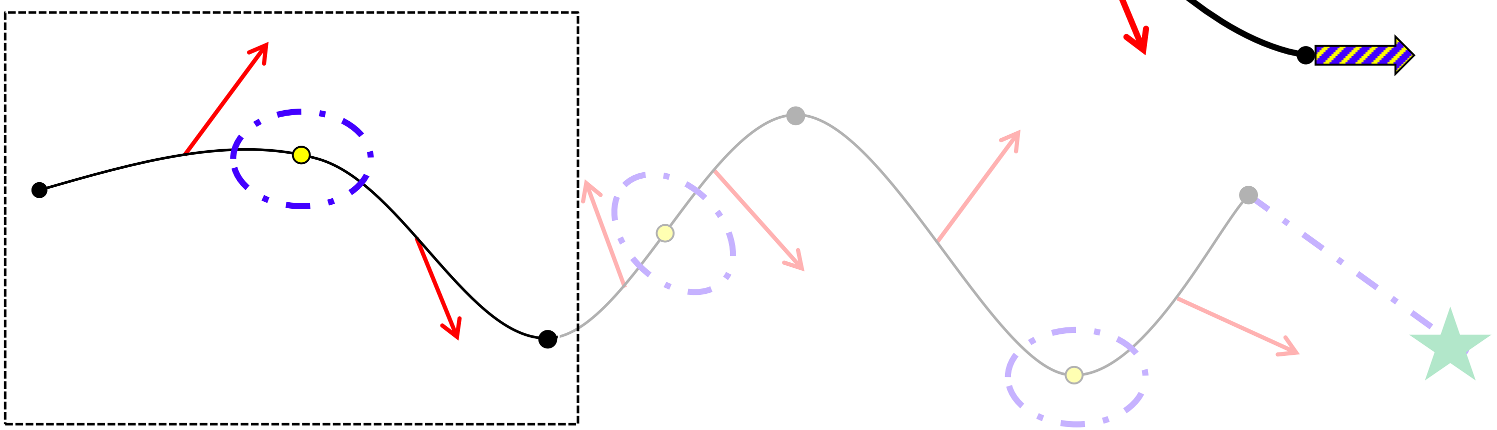
# Collocation



● Input state → Control vector ○ Propagated state — · — Constraint vector ★ Target

# Defect constraints



Input state
Control vector
Propagated state
Constraint vector
Target

# Example problem

**Colorado Center for Astrodynamics Research**
University of Colorado
Boulder, Colorado

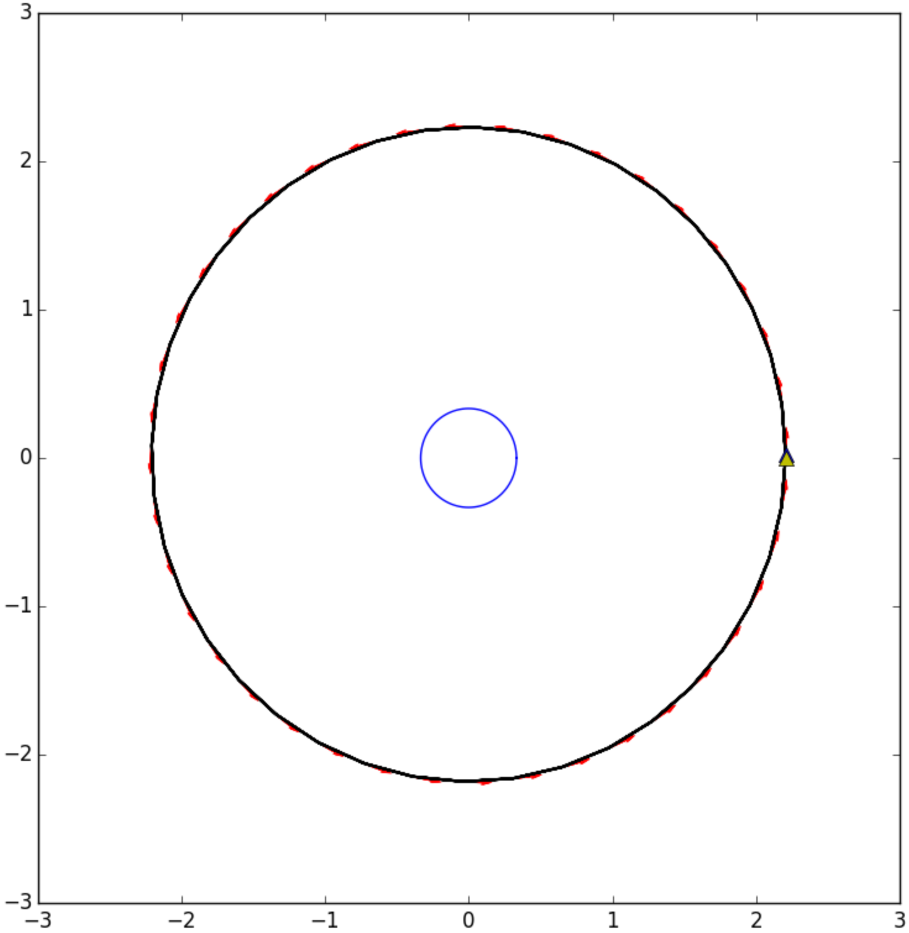
- Outward spiral from GEO
  - ~5 days flight time
  - Two-body motion
  - 3000 kg initial mass
  - 50 kW constant power (60% jet efficiency)
  - 2000 s specific impulse

- Objective: maximize orbital energy

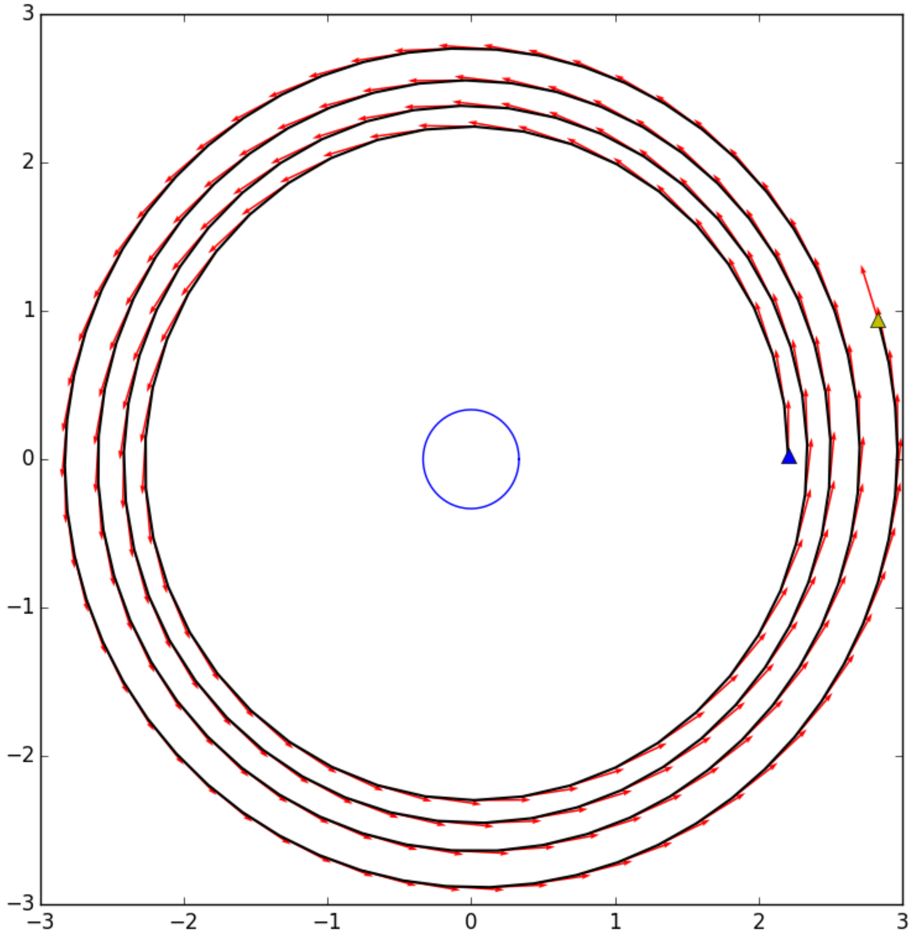
# Example problem



## Initial guess




## Solution



# Convergence behavior



(Video)

# Project suggestion

# Project suggestion

- It is actually quite manageable to write a simple but reasonably powerful low-thrust tool
  - The final product can be about 100-150 lines of code, plus some 50-100 lines to initialize the problem
  - Took me about an hour to write it from scratch (where I count several years of experience with these tools as "scratch")

- What do you need?
  - A ready-to-go optimizer that can compute derivatives through finite differencing (fmincon?)
  - A propagator for two-body motion (MATLAB's rk4?)
  - Perhaps start by building the POPSICLE-problem to get the interface down, after which all you need to do is change the contents of the function

# Project suggestion

- Suggested problem
  - 180 degree Earth-Mars transfer
  - 10 trajectory segments, single shooter
  - 2000 kg initial mass
  - 30 kW of constant power
  - Fixed specific impulse (but can be optimized with the trajectory)
  - Flight time of 250 days

- Variables
  - Hardcode the initial & final state, including the initial mass
  - That leaves 1 variable for power, 1 variable for the specific impulse, and three thrust components for 10 segments: 32 variables

- Constraints
  - 3 for the final position, 3 for the final velocity, and 10 thrust constraints for a total of 16 constraints

# Project suggestion

Colorado Center for Astrodynamics Research

University of Colorado
Boulder, Colorado

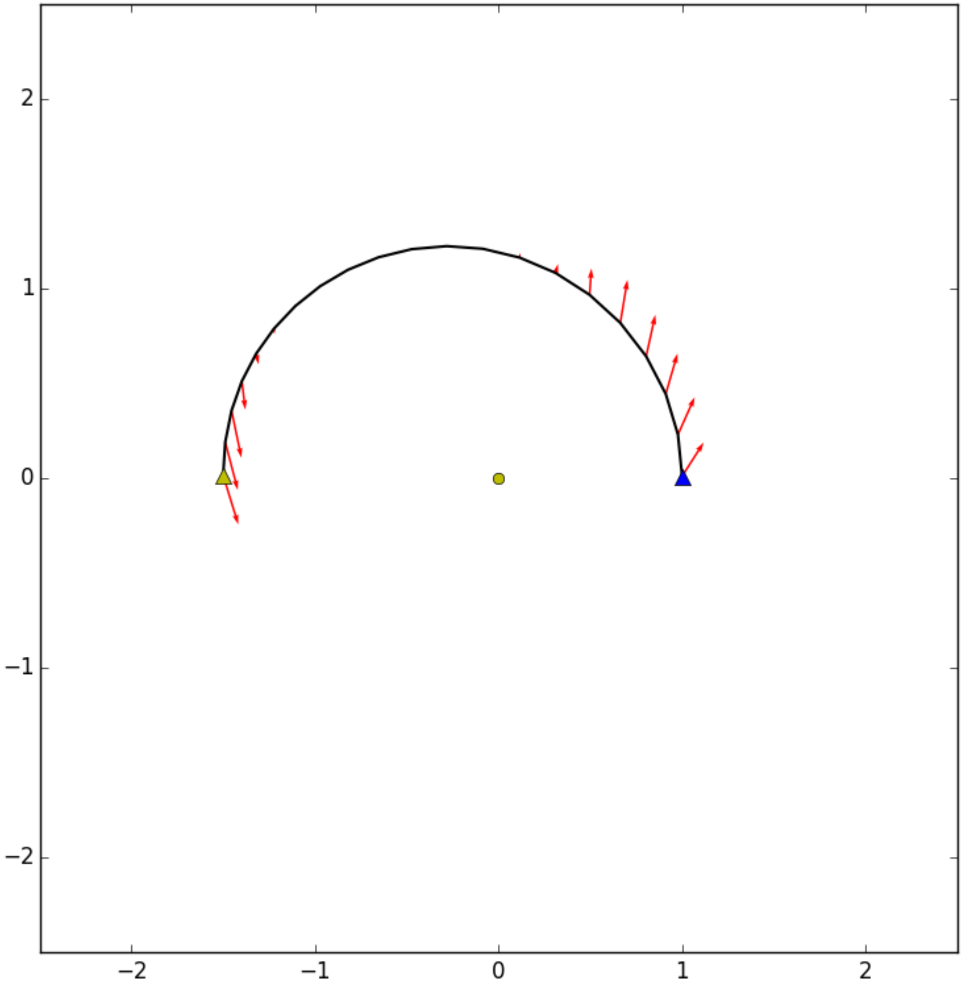
- Propagation
  - Divide total flight time (250 days) by 10
  - Apply the 3 thrust components of the current segment as an impulsive Delta-V to the initial state (just set all of these to 10% of max thrust for the initial guess)
  - Store the thrust magnitude for the correct thrust constraint
  - Propagate for the 25 days of the current segment
  - Repeat, with as initial state the final state of the last 25 day propagation
  - At the end of the 10 segments, compute the error with respect to the target state (Mars) and store these as the appropriate constraint values

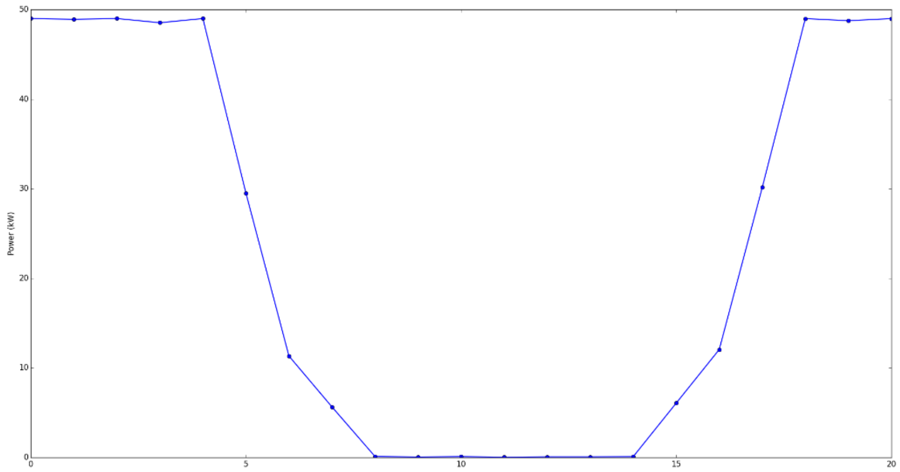
- Cost function
  - Take the mass at the end of the propagation, either maximize its positive value or minimize its negative value (most tools minimize by default)

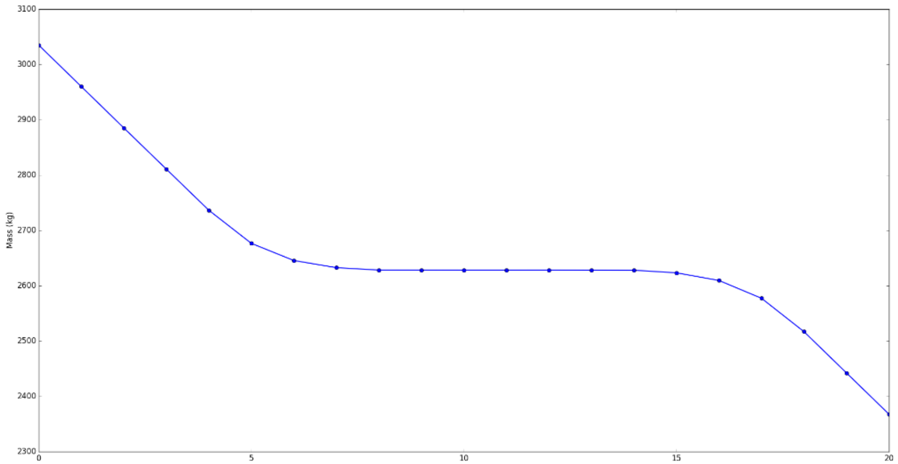
Colorado Center for Astrodynamics Research

University of Colorado
Boulder, Colorado

# Resulting transfer



# Project suggestion



- Perhaps a bit much to take in right now, but a very manageable exercise

- What you'll get out of it…
  - Familiarity with these types of tools
  - Insight into the effects of scaling
  - Optimizer will independently introduce 2 thrust arcs and 1 coast arc
  - Optimizer will independently optimize the value of the specific impulse and/or spacecraft power if you open up those variables
  - Once you've done this once in such a controlled formulation, you'll be very comfortable using these tools for much more advanced problems (you could actually evolve this simple example into a very powerful low-thrust optimization tool…)

# Conclusion

- High-level overview of the key aspects & features of optimization packages
- Hopefully, this is a useful starting point for building your own IPOPT/SNOPT/fmincon/... implementations
  - Try out the POPSICLE-problem!
  - Slides & supporting material will be online
- Contact me if you need a hand with these packages and/or the problem suggestion!
  - jon.herman@colorado.edu