NLOptControl Documentation

Release 0.0.1-rc1

Huckleberry Febbo

January 16, 2017

Contents

CHAPTER 1

Table of Contents

1.1 Background Information

While detailed information on these approaches to discretizing infinite dimensional (or continuous) optimal control problems can be found (and comes from) this Ph.D. dissertation, this related journal publication and this technical report, the Background Information section will cover some basics.

1.1.1 Lagrange Interpolating Polynomials

Definition

- given $(N+1)$ unique data points
  - $(x_0, y_0), (x_1, y_1), ..., (x_N, y_N)$
- we can create an N^{th} order Lagrange interpolating polynomial

$$P_n(x) = \sum_{i=0}^{N} \mathcal{L}_i(x) f(x_i)$$

where,

$$f(x_0) = y_0 \tag{1.1}$$
$$f(x_1) = y_1 \tag{1.2}$$
$$. \tag{1.3}$$
$$. \tag{1.4}$$
$$f(x_i) = y_i \tag{1.5}$$
$$. \tag{1.6}$$
$$f(x_N) = y_N \tag{1.7}$$

So, we are just multiplying by the given y_i values.

Lagrange Basis Polynomials

More information on Lagrange Basis Polynomials is here

$$\mathcal{L}_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{N} \frac{x - x_j}{x_i - x_j}$$

so expanding this,

$$\mathcal{L}_i(x) = \tfrac{x-x_0}{x_i-x_0}\tfrac{x-x_1}{x_i-x_1}... \tag{1.8}$$

$$...\tfrac{x-x_{i-1}}{x_i-x_{i-1}}... \tag{1.9}$$

$$...\tfrac{x-x_{i+1}}{x_i-x_{i+1}}... \tag{1.10}$$

$$...\tfrac{x-x_N}{x_i-x_N} \tag{1.11}$$

Notice that we do not include the term where $i == j$!

Please see *Functionality* for details on implementation.

1.1.2 Direct Transcription of Optimal Control Problems

Let $N_t + 1$ be the total number of discrete time points.

1.1.3 Time Marching Methods

Euler Method

Trapezoidal Method

1.1.4 Pseudospectral Methods

Change of Interval

To can change the limits of the integration (in order to apply Quadrature), we introduce $\tau \in [-1, +1]$ as a new independent variable and perform a change of variable for t in terms of τ, by defining:

$$\tau = \frac{2}{t_{N_t} - t_0} t - \frac{t_{N_t} + t_0}{t_{N_t} - t_0}$$

Polynomial Interpolation

Select a set of $N_t + 1$ node points:

$$\tau = [\tau_0, \tau_1, \tau_2, ..., \tau_{N_t}]$$

- These none points are just numbers
  - Increasing and distinct numbers $\in [-1, +1]$

A *unique* polynomial $P(\tau)$ exists (i.e. $\exists! P(\tau)$) of a maximum degree of N_t where:

$$f(\tau_k) = P(\tau_k), \quad k = 0, 1, 2, ...N_t$$

- So, the function evaluated at τ_k is equivalent the this polynomial evaluated at that point.

But, between the intervals, we must approximate $f(\tau)$ as:

$$f(\tau) \approx P(\tau) = \sum_{k=0}^{N_t} f(\tau_k)\phi_k(\tau)$$

with $\phi_k()$ are basis polynomials that are built by interpolating $f(\tau)$ at the node points.

Approximating Derivatives

We can also approximate the derivative of a function $f(\tau)$ as:

$$\frac{\mathrm{d}f(\tau)}{\mathrm{d}\tau} = \dot{f}(\tau_k) \approx \dot{P}(\tau_k) = \sum_{i=0}^{N_t} D_{ki} f(\tau_i)$$

With $\mathbf{D}$ is a $(N_t + 1) \times (N_t + 1)$ differentiation matrix that depends on:

- values of τ
- type of interpolating polynomial

Now we have an approximation of $\dot{f}(\tau_k)$ that depends only on $f(\tau)$!

Approximating Integrals

The integral we are interested in evaluating is:

$$\int_{t_0}^{t_{N_t}} f(t)\mathrm{d}t = \frac{t_{N_t} - t_0}{2} \int_{-1}^{1} f(\tau_k)\mathrm{d}\tau$$

This can be approximated using quadrature:

$$\int_{-1}^{1} f(\tau_k)\mathrm{d}\tau \sum_{k=0}^{N_t} w_k f(\tau_k)$$

where w_k are quadrature weights and depend only on:

- values of τ
- type of interpolating polynomial

Legendre Pseudospectral Method

- Polynomial

Define an N order Legendre polynomial as:

$$L_N(\tau) = \frac{1}{2^N N!}\frac{\mathrm{d}^n}{\mathrm{d}\tau^N}(\tau^2-1)^N$$

- Nodes

$$\tau_k = \begin{cases} -1, & \text{if } k = 0 \\ \text{kth root } of \dot{L}_{N_t}(\tau), & \text{if } k = 1, 2, 3, ..N_t - 1 \\ +1 & \text{if } k = N_t \end{cases} \tag{1.12}$$

- Differentiation Matrix
- Interpolating Polynomial Function

1.1.5 hp-psuedospectral method

To solve the integral constraints within the optimal control problem we employs the hp-pseudospectral method. The hp-psuedospectral method is an form of Gaussian Quadrature, which uses multi-interval collocation points.

Single Phase Optimal Control

Find:

- The state: $\mathbf{x}(t)$
- The control: $\mathbf{u}(t)$
- The integrals: $\mathbf{q}$
- The initial time: t_0
- The final time: t_f

To Minimize:

$$J = \Phi(\mathbf{x}(t_0), \mathbf{x}(t_f), \mathbf{q}, t_0, t_f)$$

That Satisfy the Following Constraints:

- Dynamic Constraints:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \psi(\mathbf{x}(t), \mathbf{u}(t), t)$$

- Inequality Path Constraints:

$$\mathbf{c}_{min} <= \mathbf{c}(\mathbf{x}(t), \mathbf{u}(t), t) <= \mathbf{c}_{max}$$

- Integral Constraints:

$$q_i = \int_{t_0}^{t_f} \Upsilon_i(\mathbf{x}(t), \mathbf{u}(t), t)\,\mathrm{d}t, \quad (i = 1, ..., n_q)$$

- Event Constraints:

$$\mathbf{b}_{min} <= \mathbf{b}(\mathbf{x}(t_0), \mathbf{x}(t_f), t_f, \mathbf{q}) <= \mathbf{b}_{max}$$

Change of Interval

To can change the limits of the integration (in order to apply Quadrature), we introduce $\tau \in [-1, +1]$ as a new independent variable and perform a change of variable for t in terms of τ, by defining:

$$t = \frac{t_f - t_0}{2}\tau + \frac{t_f + t_0}{2}$$

The optimal control problem defined above (TODO: figure out equation references), is now redefined in terms of τ as:

Find:

- The state: $\mathbf{x}(\tau)$
- The control: $\mathbf{u}(\tau)$
- The integrals: $\mathbf{q}$
- The initial time: t_0
- The final time: t_f

To Minimize:

$$J = \Phi(\mathbf{x}(-1), \mathbf{x}(+1), \mathbf{q}, t_0, t_f)$$

That Satisfy the Following Constraints:

- Dynamic Constraints:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}\tau} = \frac{t_f - t_0}{2}\psi(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau, t_0, t_f)$$

- Inequality Path Constraints:

$$\mathbf{c}_{min} <= \mathbf{c}(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau, t_0, t_f) <= \mathbf{c}_{max}$$

- Integral Constraints:

$$q_i = \frac{t_f - t_0}{2}\int_{-1}^{+1} \Upsilon_i(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau, t_0, t_f)\,\mathrm{d}\tau, \quad (i = 1, ..., n_q)$$

- Event Constraints:

$$\mathbf{b}_{min} <= \mathbf{b}(\mathbf{x}(-1), \mathbf{x}(+1), t_f, \mathbf{q}) <= \mathbf{b}_{max}$$

Divide The Interval $\tau \in [-1, +1]$

The interval $\tau \in [-1, +1]$ is now divided into a mesh of K mesh intervals as:

$$[T_{k-1}, T_k], k = 1, ..., T_K$$

with $(T_0, ..., T_K)$ being the mesh points; which satisfy:

$$-1 = T_0 < T_1 < T_2 < T_3 < ..... < T_{K-1} < T_K = T_f = +1$$

Rewrite the Optimal Control Problem using the Mesh

Find:

- The state : $\mathbf{x}^{(k)}(\tau)$ **in mesh interval k**
- The control: $\mathbf{u}^{(k)}(\tau)$ **in mesh interval k**
- The integrals: $\mathbf{q}$
- The initial time: t_0
- The final time: t_f

To Minimize:

$$J = \Phi(\mathbf{x}^{(1)}(-1), \mathbf{x}^{(K)}(+1), \mathbf{q}, t_0, t_f)$$

That Satisfy the Following Constraints:

- Dynamic Constraints:

$$\frac{\mathrm{d}\mathbf{x}^{(k)}(\tau^{(k)})}{\mathrm{d}\tau^{(k)}} = \frac{t_f - t_0}{2}\psi(\mathbf{x}^{(k)}(\tau^{(k)}), \mathbf{u}^{(k)}(\tau^{(k)}), \tau^{(k)}, t_0, t_f), \quad (k = 1, ..., K)$$

- Inequality Path Constraints:

$$\mathbf{c}_{min} <= \mathbf{c}(\mathbf{x}^{(k)}(\tau^{(k)}), \mathbf{u}^{(k)}(\tau^{(k)}), \tau^{(k)}, t_0, t_f) <= \mathbf{c}_{max}, \quad (k = 1, ..., K)$$

- Integral Constraints:

$$q_i = \frac{t_f - t_0}{2}\sum_{k=1}^{K}\int_{T_{k-1}}^{T_k} \Upsilon_i(\mathbf{x}^{(k)}(\tau^{(k)}), \mathbf{u}^{(k)}(\tau^{(k)}), \tau, t_0, t_f)\,\mathrm{d}\tau, \quad (i = 1, ..., n_q, k = 1, ..., K)$$

- Event Constraints:

$$\mathbf{b}_{min} <= \mathbf{b}(\mathbf{x}^{(1)}(-1), \mathbf{x}^{(K)}(+1), t_f, \mathbf{q}) <= \mathbf{b}_{max}$$

- State Continuity
  - Also, we must **now** constrain the state to be continuous at each interior mesh point $(T_1, ...T_{k-1})$ by enforcing:

$$\mathbf{y}^k(T_k) = \mathbf{y}^{k+1}(T_k)$$

Optimal Control Problem Approximation

The optimal control problem will now be approximated using the Radau Collocation Method as which follows the description provided by *[BGar11]*. In collocation methods, the state and control are discretized at particular points within the selected time interval. Once this is done the problem can be transcribed into a nonlinear programming problem (NLP) and solved using standard solvers for these types of problems, such as IPOPT or KNITRO.

For each mesh interval $k \in [1, .., K]$**:**

$$\mathbf{x}^{(k)}(\tau) \approx \mathbf{X}^{(k)}(\tau) = \sum_{j=1}^{N_k+1} \mathbf{X}_j^{(k)} \frac{\mathrm{d}\mathcal{L}_j^k(\tau)}{\mathrm{d}\tau} \tag{1.13}$$

$$where, \tag{1.14}$$

$$\mathcal{L}_j^k(\tau) = \prod_{\substack{l=1 \\ l \neq j}}^{N_k+1} \frac{\tau - \tau_l^{(k)}}{\tau_j^{(k)} - \tau_l^{(k)}} \tag{1.15}$$

$$and, \tag{1.16}$$

$$D_{ki} = \dot{\mathcal{L}}_i(\tau_k) = \frac{\mathrm{d}\mathcal{L}_j^k(\tau)}{\mathrm{d}\tau} \tag{1.17}$$

also,

- $\mathcal{L}_j^{(k)}(\tau), \quad (j = 1, ..., N_k + 1)$ is a basis of Lagrange polynomials
- $(\tau_1^k, ..., \tau_{N_k}^{(k)})$ are the Legendre-Gauss-Radau collocation points in mesh interval k
  - defined on the subinterval $\tau^{(k)} \in [T_{k-1}, T_k]$
  - $\tau_{N_k+1}^{(k)} = T_k$ is a noncollocated point

A basic description of Lagrange Polynomials is presented in *Lagrange Interpolating Polynomials*

The D matrix:

- **Has a size** $= [N_c] \times [N_c + 1]$
  - with $(1 <= k <= N_c), (1 <= i <= N_c + 1)$
  - **this non-square shape because the state approximation uses the** $N_c + 1$ **points:** $(\tau_1, ...\tau_{N_c+1})$
  - **but collocation is only done at the** N_c **LGR points:** $(\tau_1, ...\tau_{N_c})$

If we define the state matrix as:

$$\mathbf{X}^{LGR} = \begin{bmatrix} \mathbf{X}_1 \\ . \\ . \\ . \\ \mathbf{X}_{N_c+1} \end{bmatrix} \tag{1.18}$$

The dynamics are collocated at the N_c LGR points using:

$\mathbf{D}_k \mathbf{X}^{LGR} = \frac{(t_f - t_0)}{2} \mathbf{f}(\mathbf{X}_k, \mathbf{U}_k, \tau, t_0, t_f) \quad for \quad k = 1, ..., Nc$

with,

- $\mathbf{D}_k$ being the k^{th} row of the $\mathbf{D}$ matrix.

References

1.2 Package Functionality

1.2.1 Code Development

Approximation of Optimal Control Problem

Completed Functionality

Lagrange Basis Polynomials

Functionality The basic description of this functionality is detailed here *Lagrange Interpolating Polynomials*

lagrange_basis_poly() The Lagrange basis polynomial equations where turned into a function.

interpolate_lagrange() The interpolation functionality was pushed to a lower level. This allows the user to easily use code to interpolate a polynomial.

The **development** of these function can be:

- Viewed remotely on using the jupyter nbviewer.
- Viewed locally and interacted using IJulia

  To do this in julia type:

```
using IJulia
notebook(dir=Pkg.dir("NLOptControl/examples/LIP/lagrange_basis_poly_dev"))
```

Examples

Simple Interpolation -> ex#1 In this first example, we demonstrate the functionality using the interpolating functionality.

where:

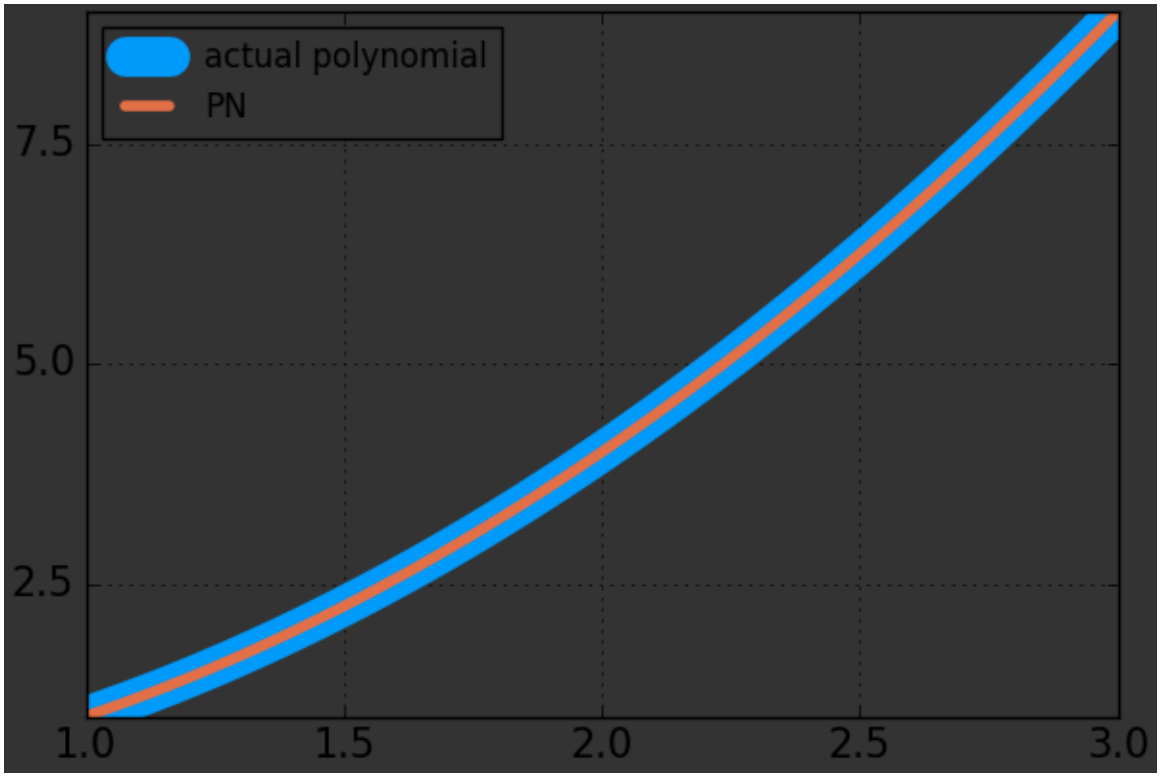
$$y(x) = x^2$$

and the interval from `x=1` to `x=3`

with:

```
N = 2; # number of collocation points
```

Test 1a



- Conclusions
  - Working as expected.

Scaled Lagrange Polynomials-> ex#2

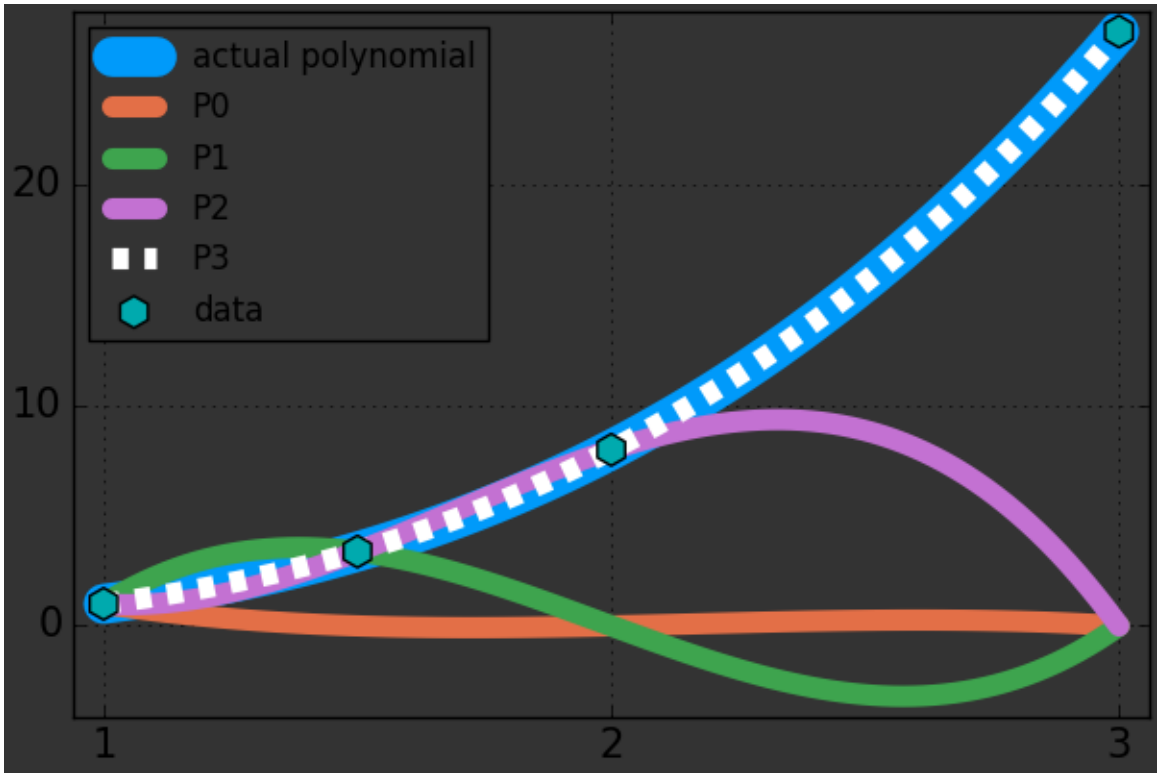
where:

$$y(x) = x^3$$

and the interval from `x=1` to `x=3`

with:

```
N = 2; # number of collocation points
```



Test 2a

- Conclusions
  - Each scaled Lagrange polynomial passes through a control point
  - The final interpolating polynomial passes through each control point and is exact

Runge's Phenomena-> ex#3 This example investigates Runge's phenomena.

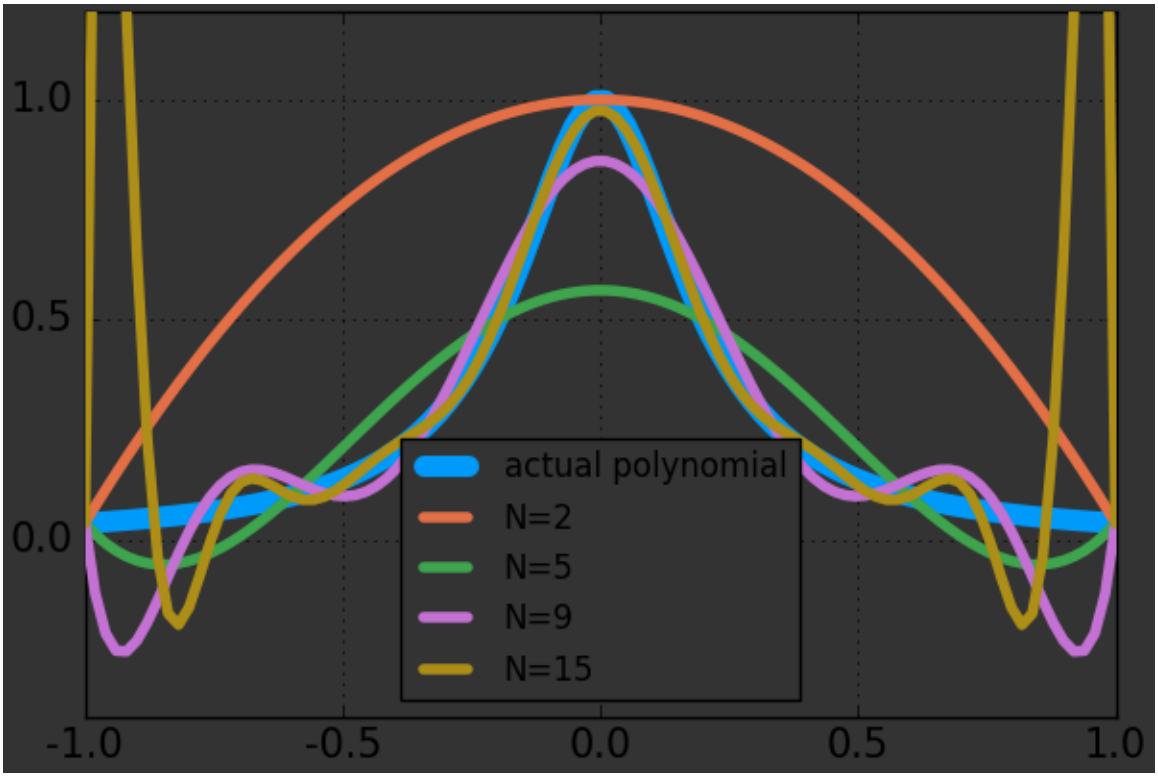
where:

$$y(x) = \frac{1}{1 + 25x^2}$$

and the interval from `x0=-1` to `xf=1`

with:

```
    N = order of Lagrange Polynomial
    x_data = linspace(x0,xf,N+1);
```



- Conclusions
  - Be careful not to use too high of an `N`

To Mitigate Runge's Phenomenon

- Could sample more near the end points
- Could use Chebyshev nodes

These examples can be:

- Viewed remotely on using the jupyter nbviewer.
- Viewed locally and interacted using IJulia

To do this in julia type:

```
using IJulia
notebook(dir=Pkg.dir("NLOptControl/examples/LIP/"))
```

Legendre Gaussian Method

LGL Single Interval

Basic Problem Definition The code developed in this package uses the Legendre-Pseudospectral Method with Lagrange-Gauss-Lobatto (LGL) nodes. A basic description of this implementation presented in this documentation at *Pseudospectral Methods* and more much more detailed information can be found in both *[ASTW11]* and *[AHer15]*.

Examples

In these examples we use:

- Legendre-Gauss-Lobatto (LGL) nodes
- Single interval approximations
- Approximate integrals in the range of `[-1,1]`
- Approximate derivatives in the range of `[-1,1]`

These examples can be:

- Viewed remotely on using the jupyter nbviewer.
- Viewed locally and interacted using IJulia

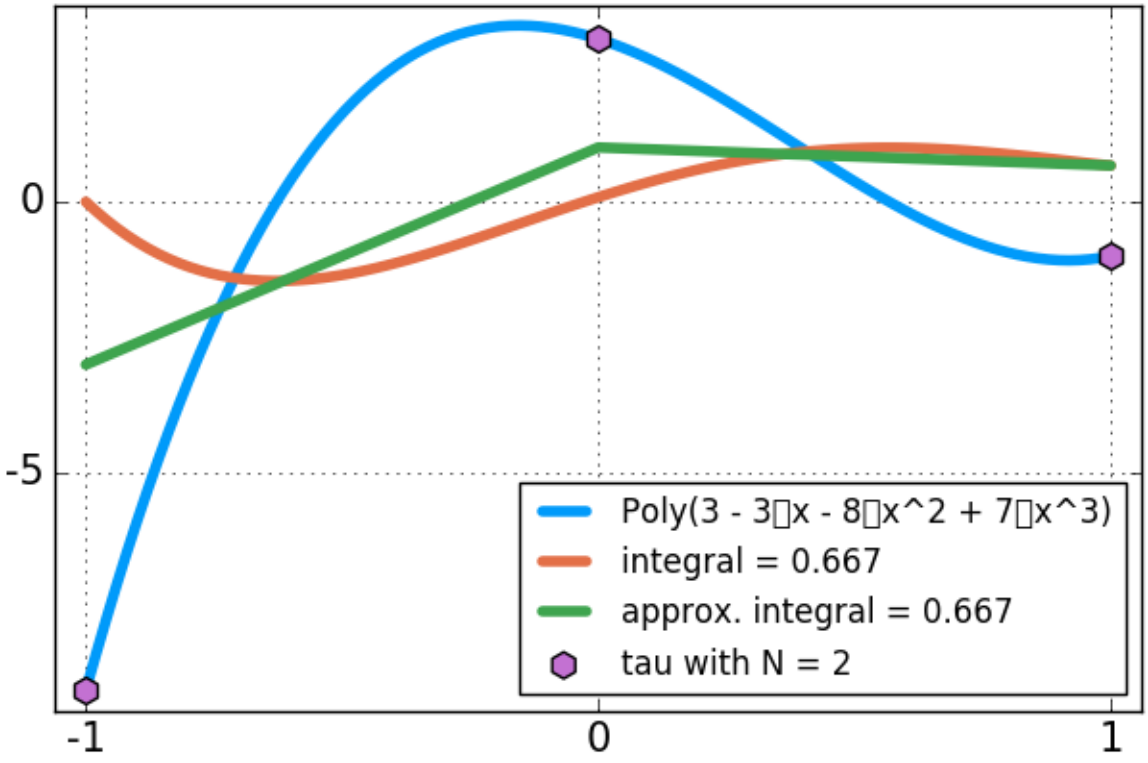
  To do this in julia type:

```
using IJulia
notebook(dir=Pkg.dir("NLOptControl/examples/LGL_SI/"))
```

Example 1 In the first example, we borrow a problem from Wikipedia.

where:

$$y(x) = 7x^3 - 8x^2 - 3x + 3$$



Difference between the Wikipedia Example and this Example

The difference between Wikipedia example and this one is that the Wikipedia example uses Gauss-Legendre Quadrature while the code developed in this package uses Legendre-Pseudospectral Method with Lagrange-Gauss-Lobatto (LGL) nodes. Information on the difference between these methods and many more can be found in both *[BSTW11]* and *[BHer15]*.

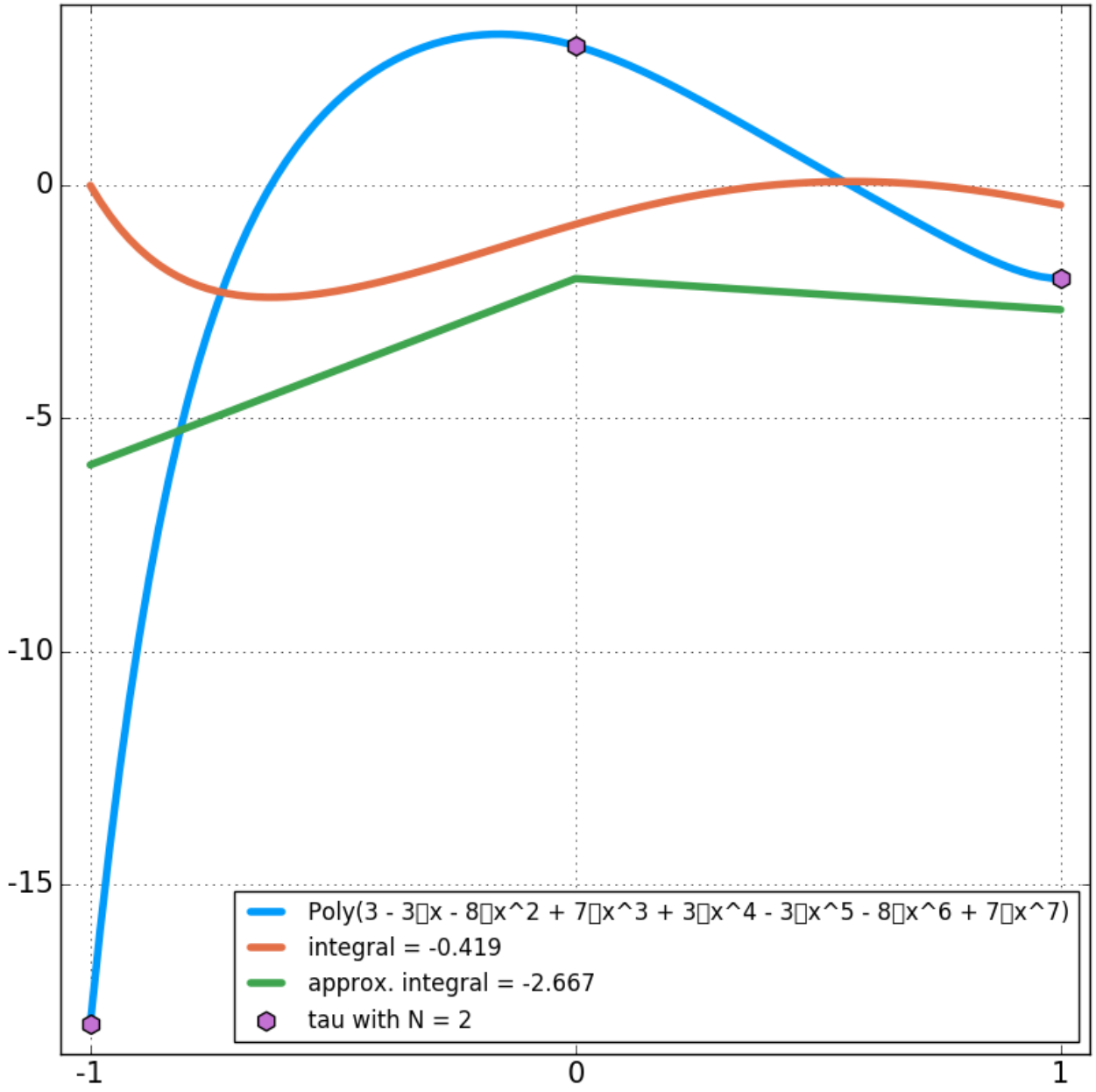
- Conclusions
  - We are able to exactly determine the integral

References

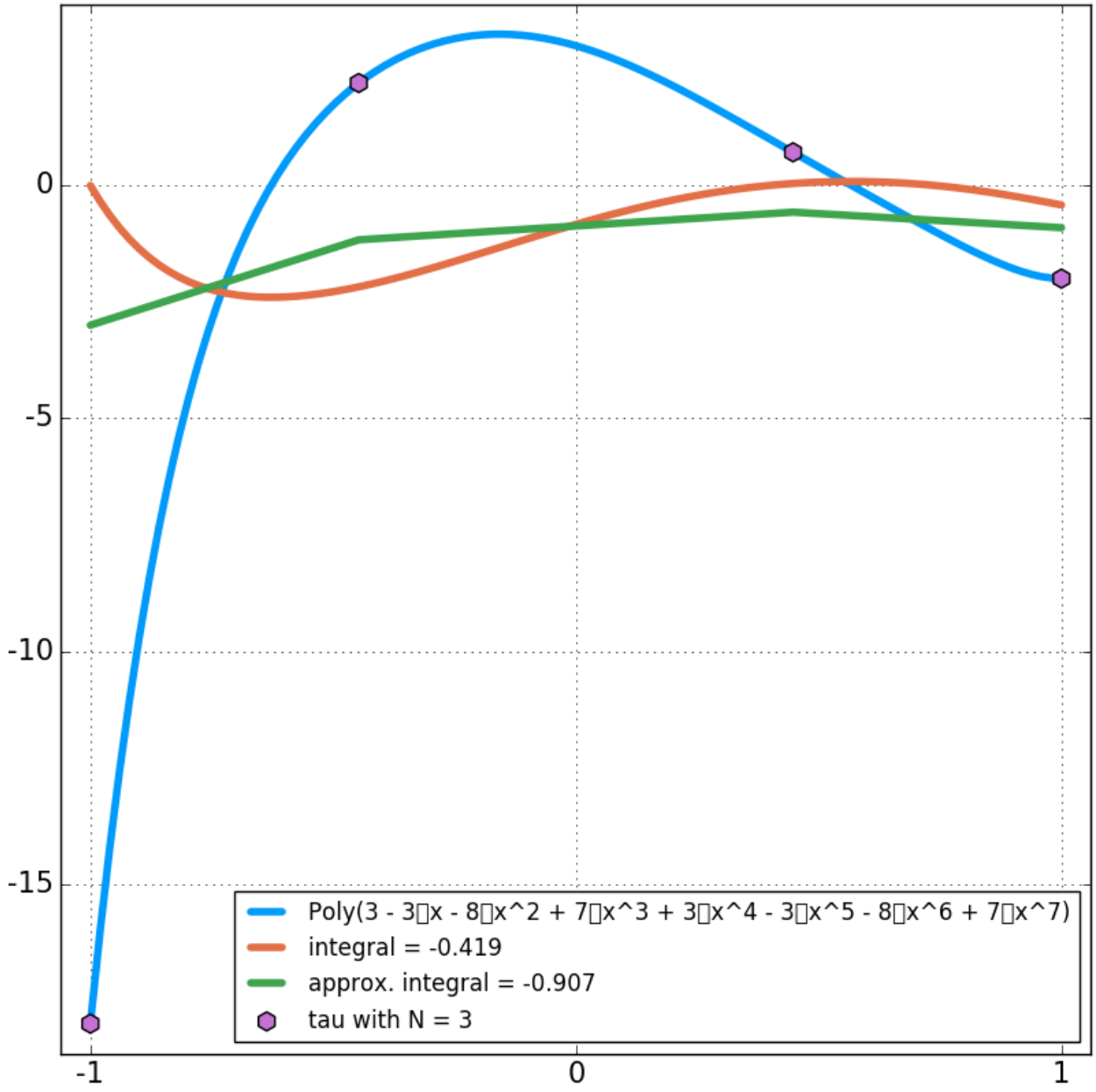
Example 2 In the second example, we increase the order of the polynomial from `3` to `7`. Then we increase `N` until we achieve accurate enough results.

where:

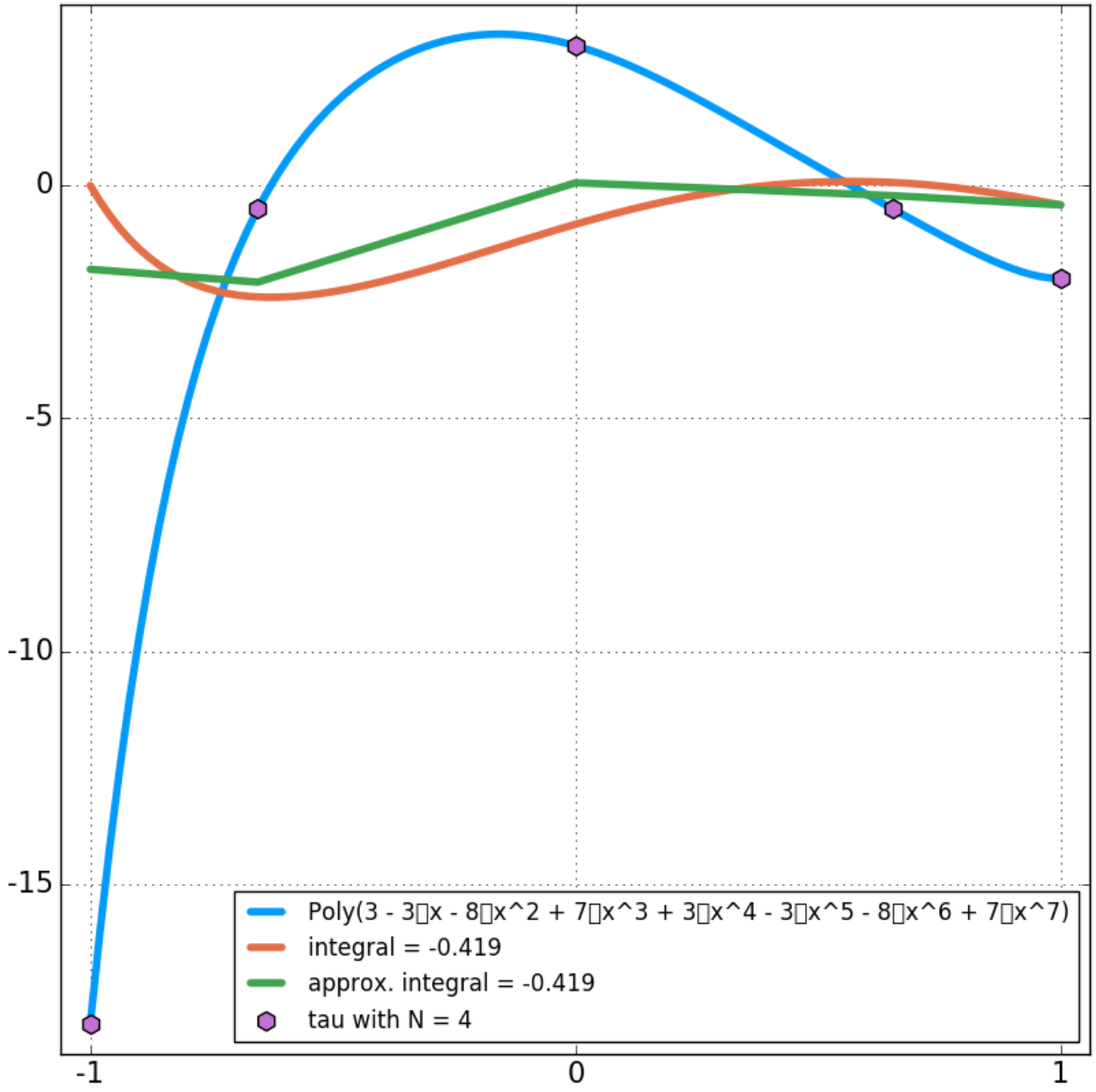
$$y(x) = 7x^7 - 8x^6 - 3x^5 + 3x^4 + 7x^3 - 8x^2 - 3x + 3$$


0
-5
-10
-15
-1
0
1
Poly(3 - 3□x - 8□x^2 + 7□x^3 + 3□x^4 - 3□x^5 - 8□x^6 + 7□x^7)
integral = -0.419
approx. integral = -2.667
tau with N = 2

Test 2a


0
-5
-10
-15
-1
0
1
Poly(3 - 3x - 8x^2 + 7x^3 + 3x^4 - 3x^5 - 8x^6 + 7x^7)
integral = -0.419
approx. integral = -0.907
tau with N = 3

Test 2b



Test 2c

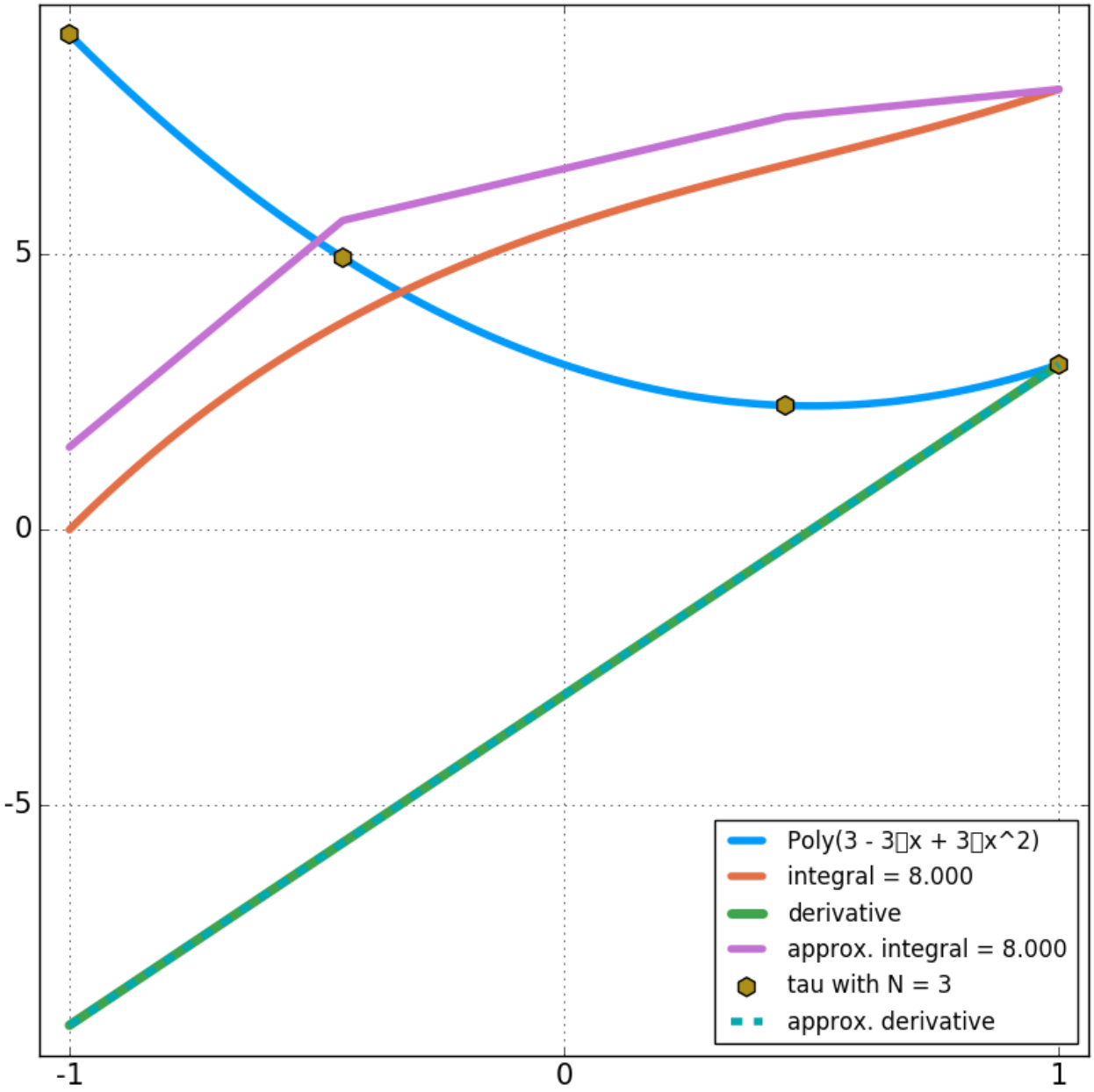
Conclusions

- We are able to determine the integral very accurately when we increase N to 4

Example 3 In the third example, we approximate the derivative of a function by *Approximating Derivatives*.

Test 3a In this test:

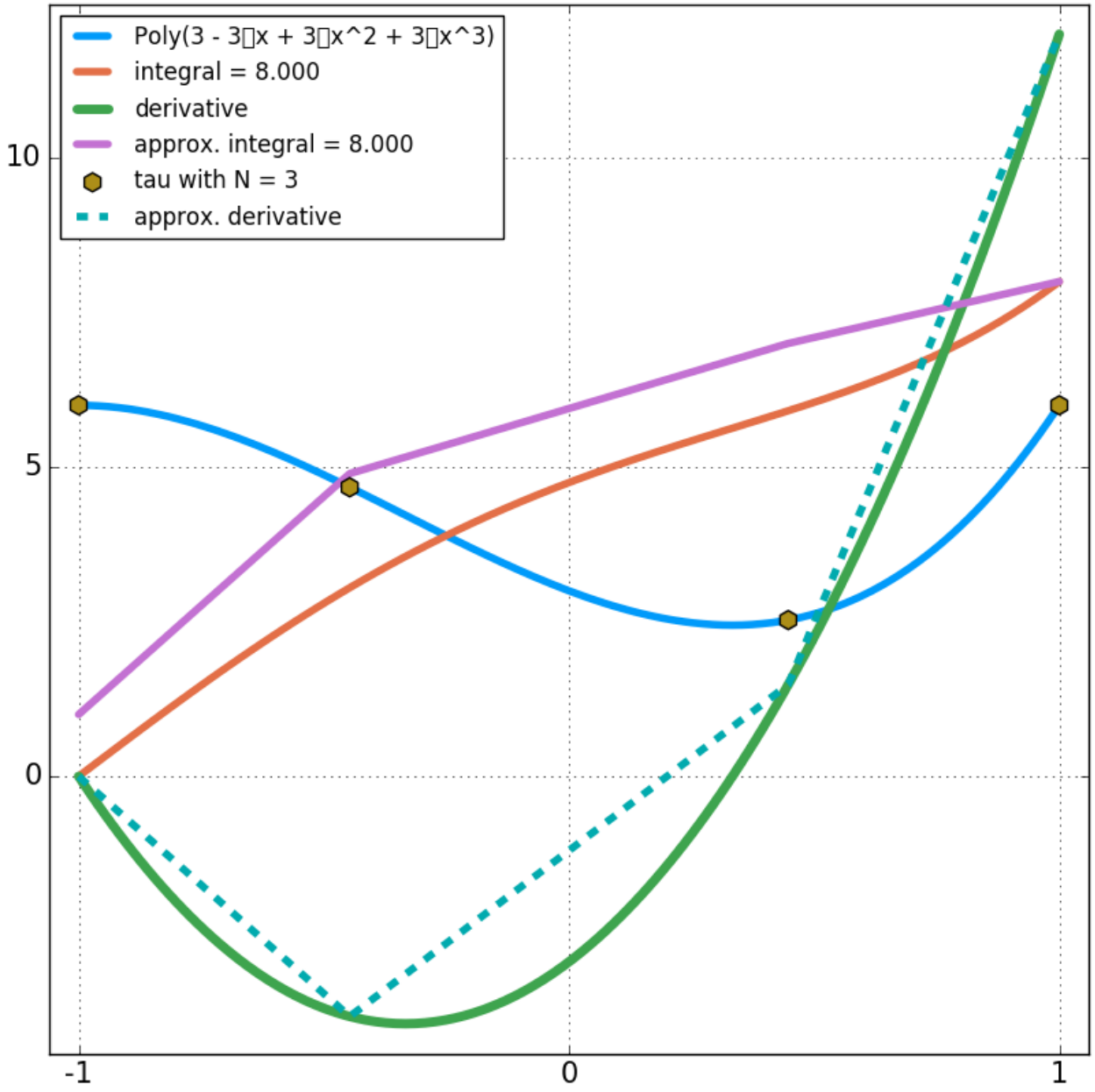
$$y(x) = 3x^2 - 3x + 3$$



- We are able to determine the derivative exactly when they are linear functions is $N = 3$

Test 3b In this test we increase the order of `y(x)` to:

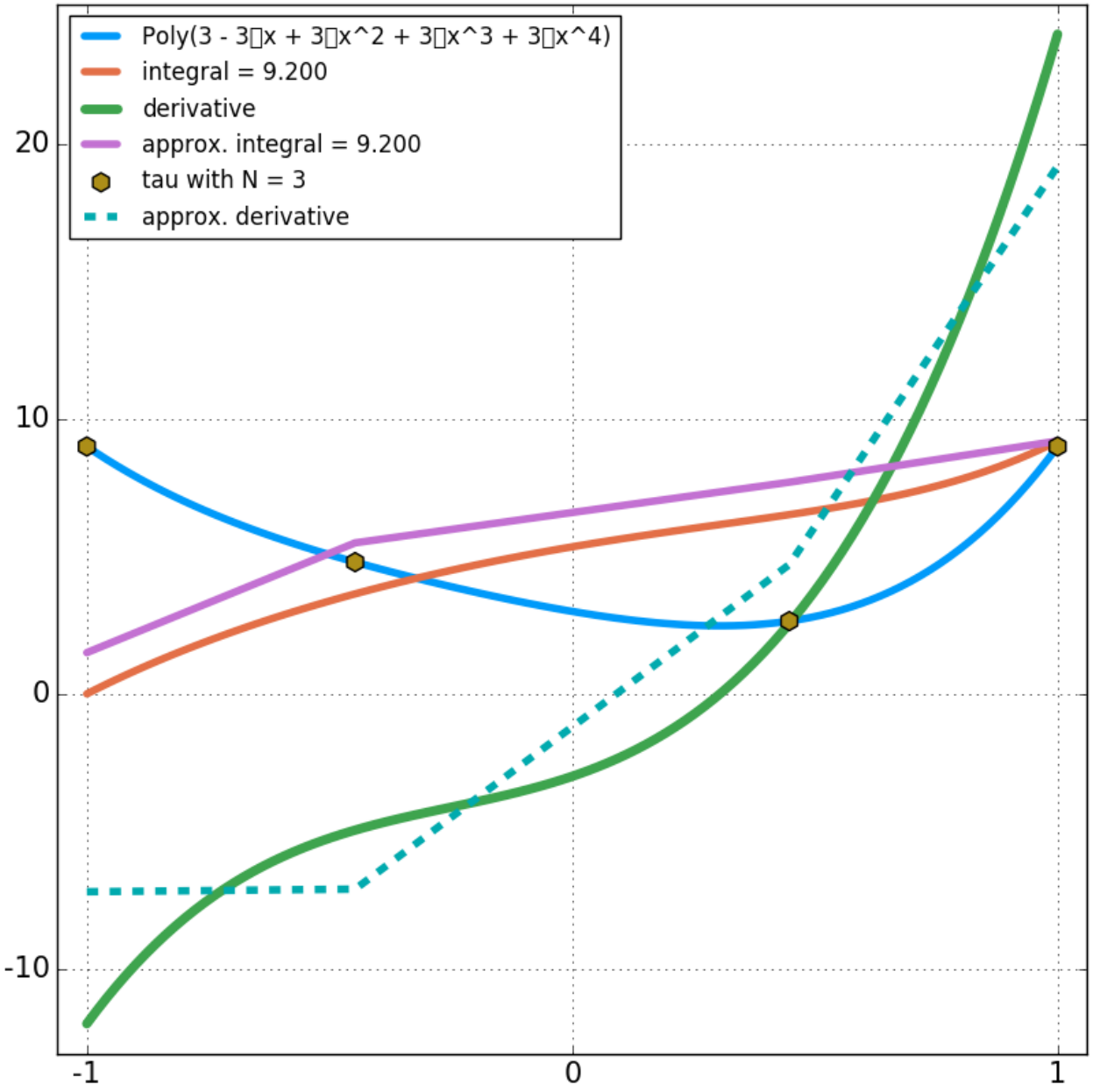
$$y(x) = 3x^3 + 3x^2 - 3x + 3$$



- When the derivative function becomes nonlinear
  - We can no longer calculate it exactly everywhere
  - There are only $N_{t+1} = 4$ node points
  - To calculate the derivative exactly we would need an amount of N_{t+1}
- We are still calculating the integral exactly and should be able to with $N = 3$ until x^5

Test 3c In this test we increase the order of `y(x)` to:

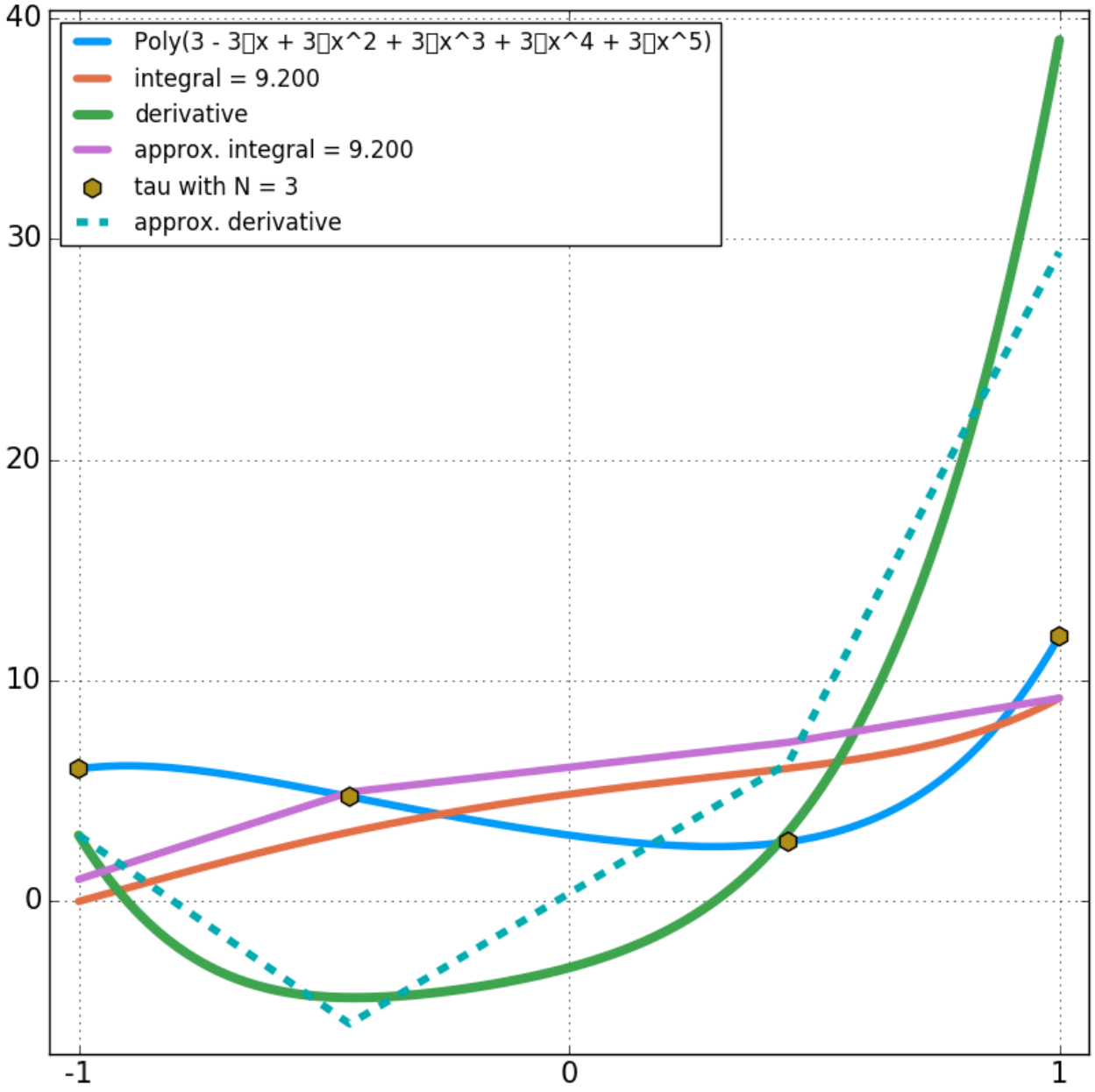
$$y(x) = 3x^4 + 3x^3 + 3x^2 - 3x + 3$$



- We are still calculating the integral exactly and should be able to with $N = 3$ until x^5

Test 3d In this test we increase the order of `y(x)` to:

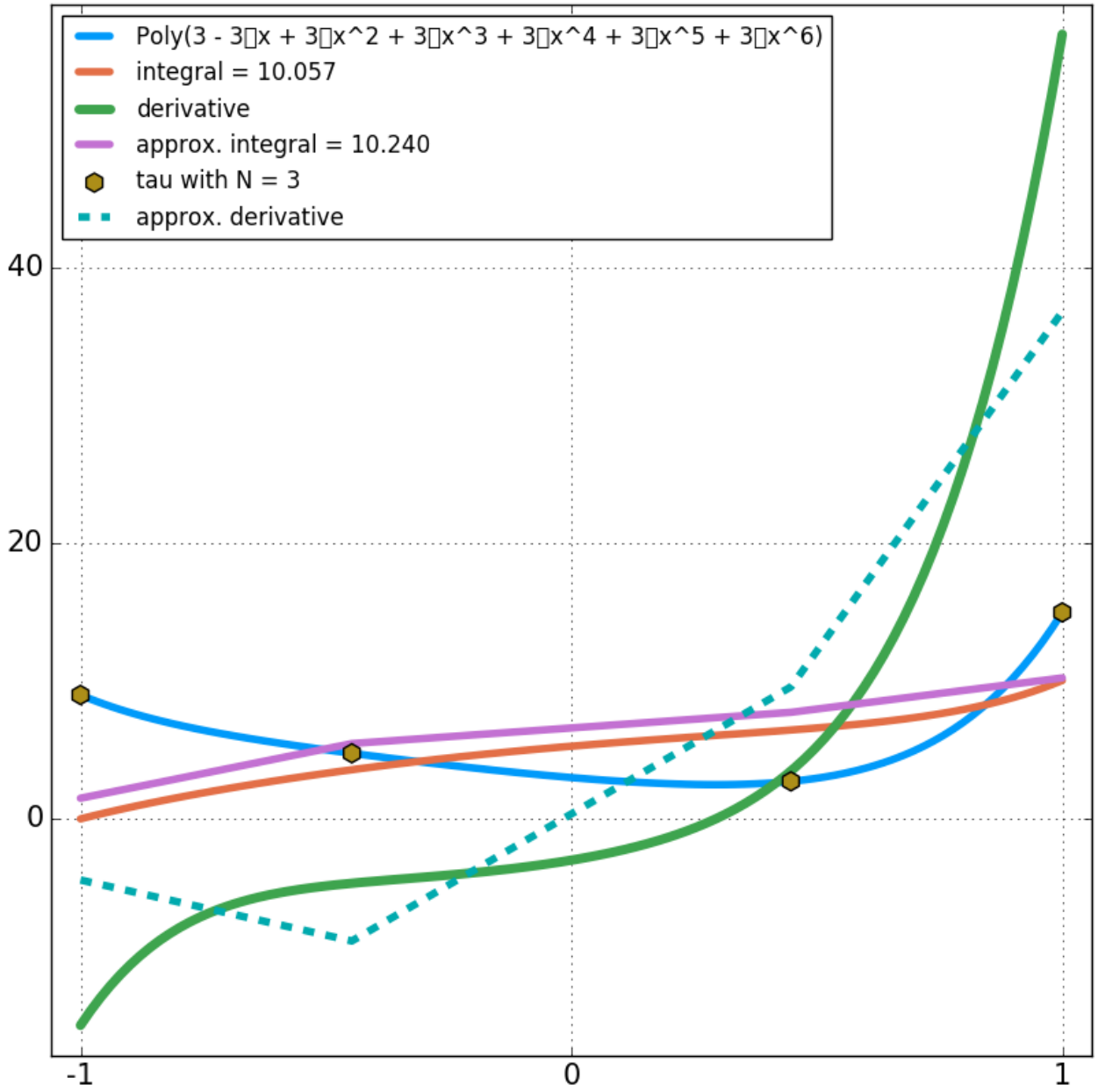
$$y(x) = 3x^5 + 3x^4 + 3x^3 + 3x^2 - 3x + 3$$



- We are still calculating the integral exactly with $N = 3$!!
- The percent error is = 0.000000000000000000 %

Test 3e In this test we increase the order of `y(x)` to:

$$y(x) = 3x^6 + 3x^5 + 3x^4 + 3x^3 + 3x^2 - 3x + 3$$



- As expected, we are not still calculating the integral exactly with $N = 3$!!
- The percent error is = -1.818181818181822340 %

References

LGR Single Interval

Basic Problem Definition The code developed here uses the Legendre-Pseudospectral Method with Legendre-Gauss-Radau (LGR) nodes. This example demonstrates using the LGR points to calculate the integral and the derivative of a known polynomial function. It can be seen, that it behaves as expected. One, major difference between LGR and LGL is that the LGR method does **NOT** use both endpoints, in fact the LGR method omits the final end point. Researchers at the University of Florida describe this method in many papers including *[A1][A2][A3][A4]*.

Examples

In these examples we use:

- Legendre-Gauss-Lobatto (LGR) nodes
- Single interval approximations
- Approximate integrals in the range of `[x0,xf]`
- Approximate derivatives in the range of `[x0,xf]`

These examples can be:

- Viewed remotely on using the jupyter nbviewer.
- Viewed locally and interacted using IJulia

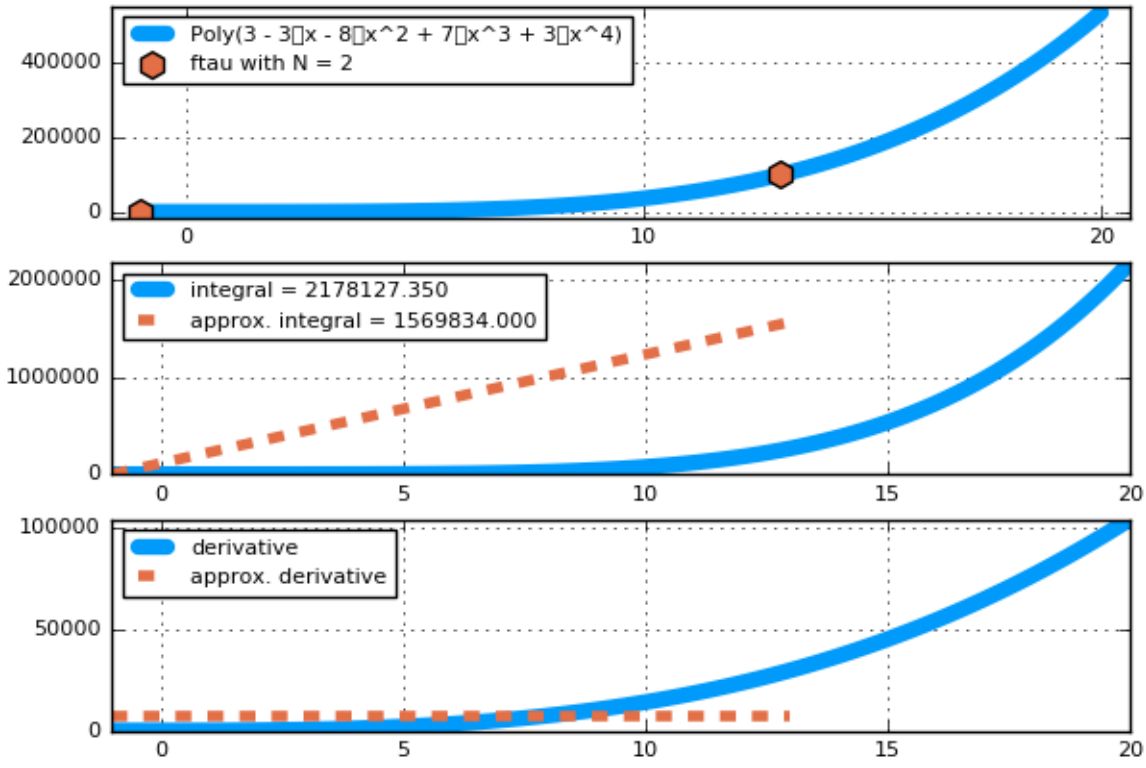
  To do this in julia type:

```
using IJulia
notebook(dir=Pkg.dir("NLOptControl/examples/LGR_SI/"))
```

Example 1 In the first example, we demonstrate the functionality using LGR nodes using `N=2`.

where:

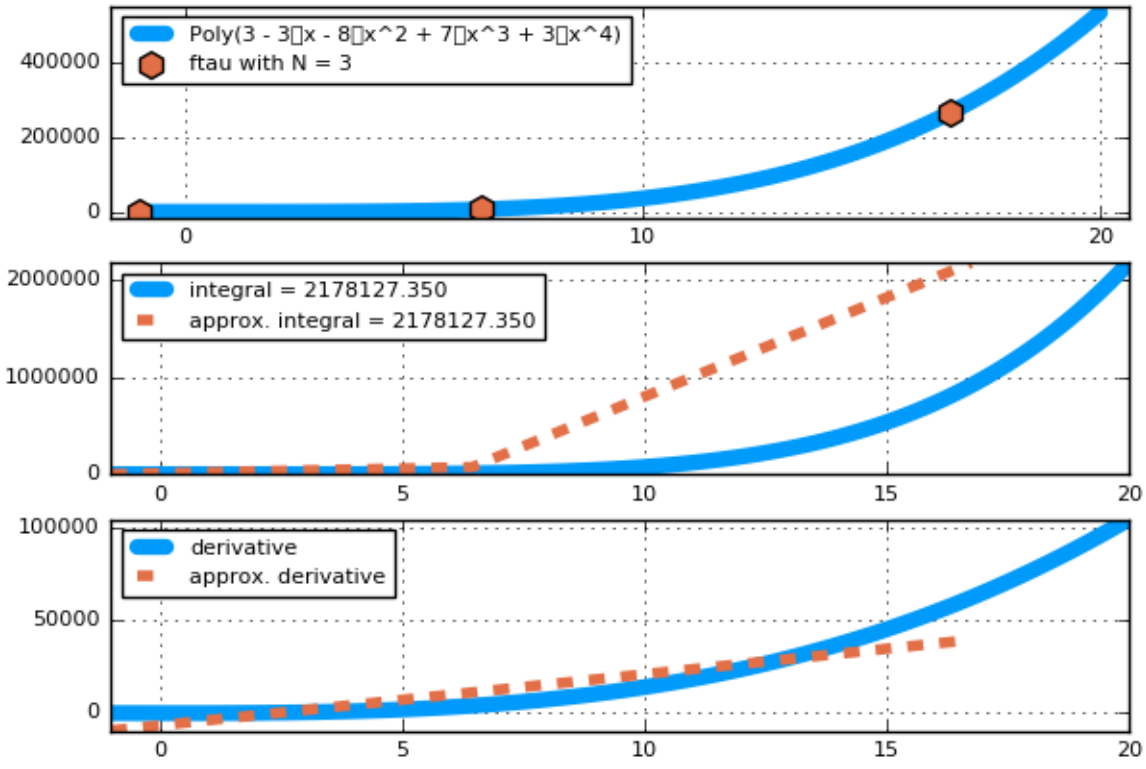
$$y(x) = 3x^4 + 7x^3 - 8x^2 - 3x + 3$$



Test 1a

Notice for all cases, that the end points are NOT included.

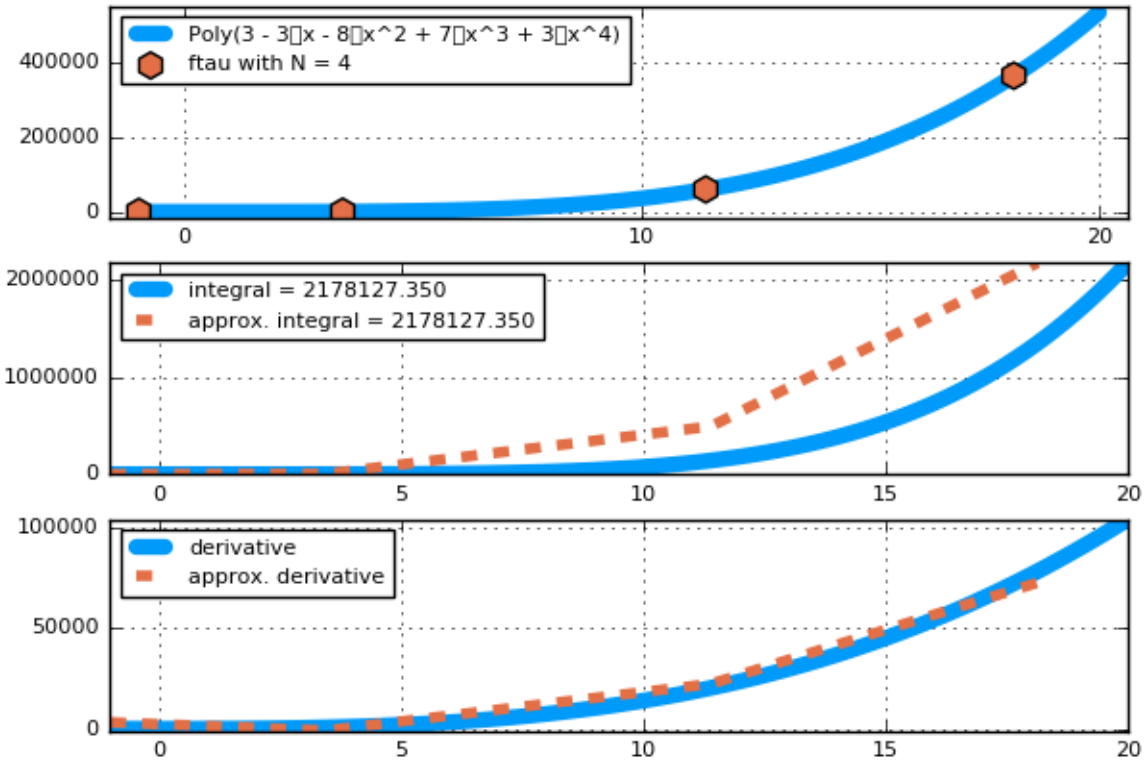
This is one of the differences between the LGL and LGR methods.



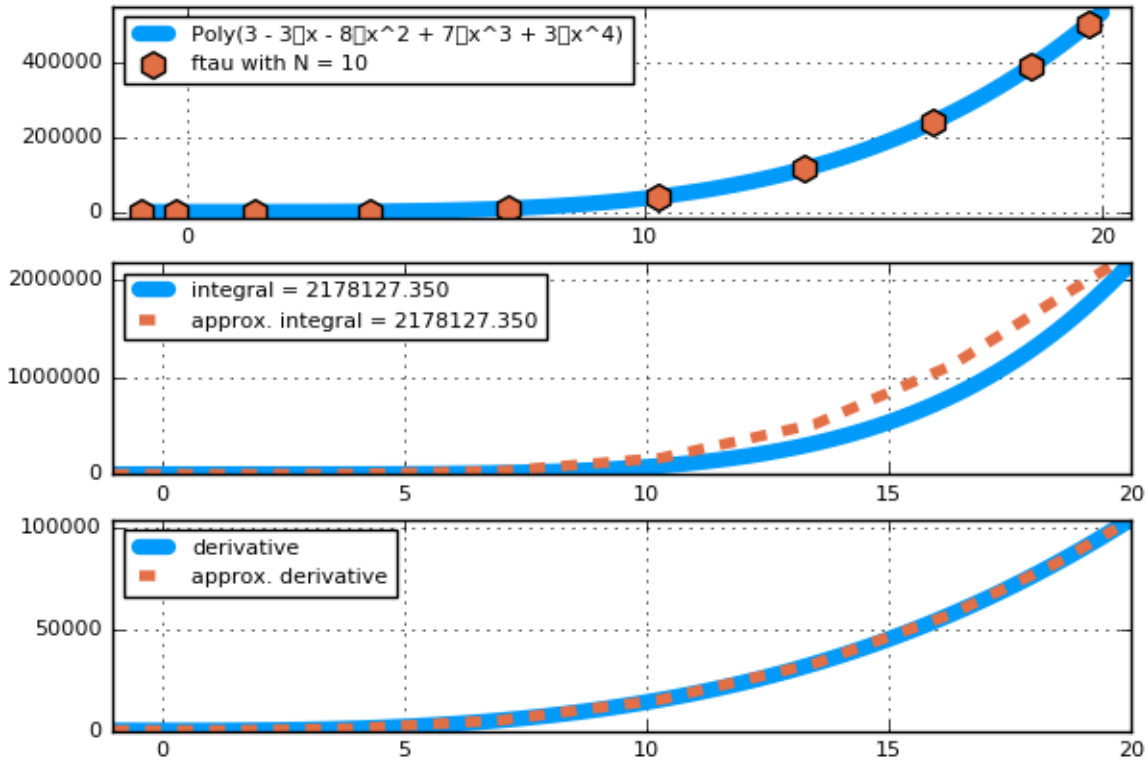
Test 1b

In Test 1b `N=4`

So we can calculate the integral of a 4th order polynomial exactly. Because `2*N-2 = 4`.



Test 1c



Test 1d

- Conclusions
  - Working as expected.

References

LGR Multiple Interval Now, using Legendre-Gauss-Radau (LGR) points with multiple intervals to calculate the integral and the derivative of a known polynomial function. This example demonstrates using the Legendre-Gauss-Radau (LGR) points to calculate the integral and the derivative of a known polynomial function using a **multiple interval approach**.

Researchers at the University of Florida

describe this method in many papers including *[CDar11][CGar11][CGPH+10][CGPH+09]*.

Functionality **NOTE:** this is not an exhaustive list and there are help notes for most functions, that can be easily seen by typing:

```
julia>?  # then press enter and type
         # the name of the function of interest
```

LGR_matrices(ps,nlp)

- Calculate LGR matrices
  - IMatrix
  - DMatrix

Notes:

- Make sure that you calculate these matrices before you try and use them in either *integrate_states()* or *differentiate_state()*.

Examples

In these examples we use:

- Legendre-Gauss-Radau (LGR) nodes
- Multiple interval approximations
- Approximate integrals in the range of `[x0,xf]`
- Approximate derivatives in the range of `[x0,xf]`

Neglecting Non-Collocated Point $Y^{(k)}()$ -> ex#1 In this first example, we demonstrate the functionality using LGR nodes

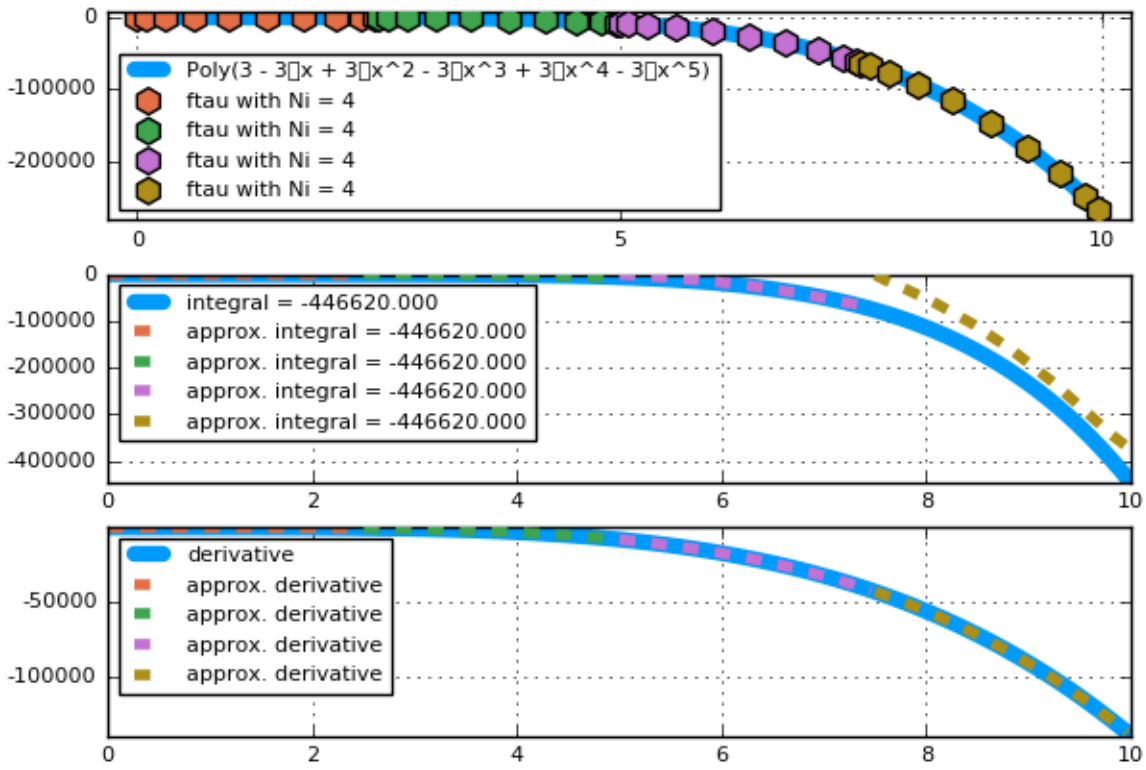
where:

$$y(x) = -3x^5 + 3x^4 + 7x^3 - 8x^2 - 3x + 3$$

Test 1a

with:

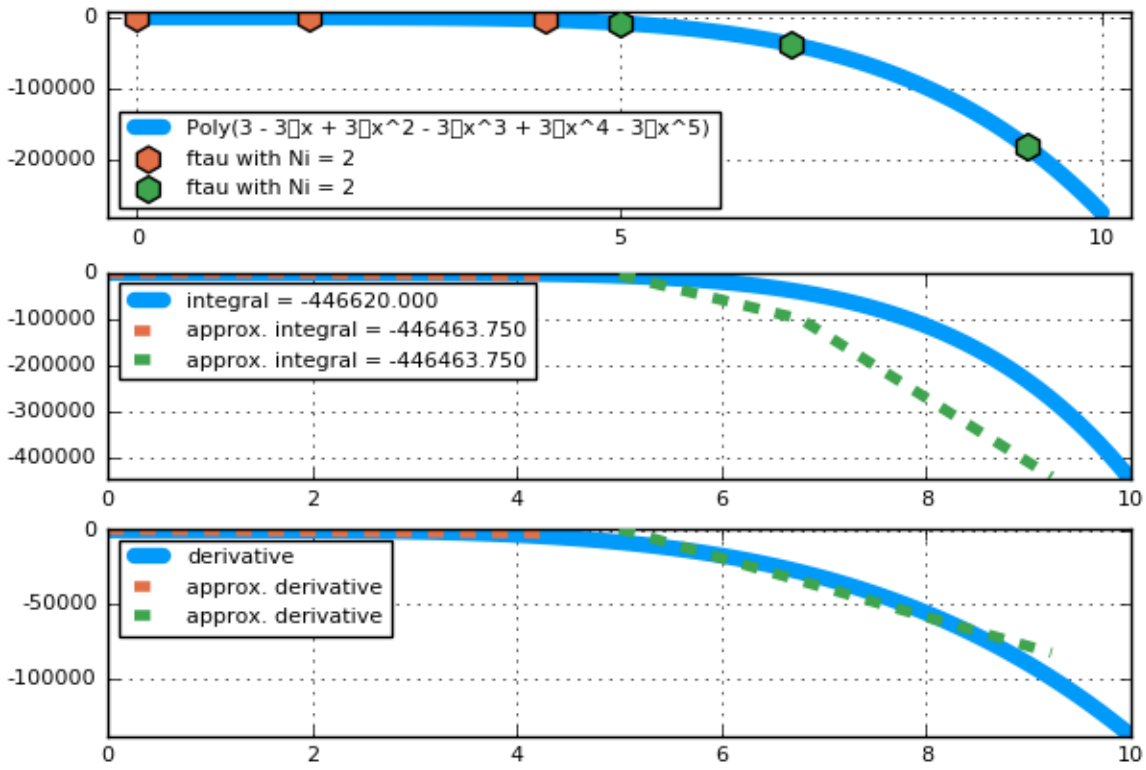
```
    Nc = Int64(10); # number of collocation points in each interval
    Ni = Int64(4);  # number of intervals
```



Test 1b

with:

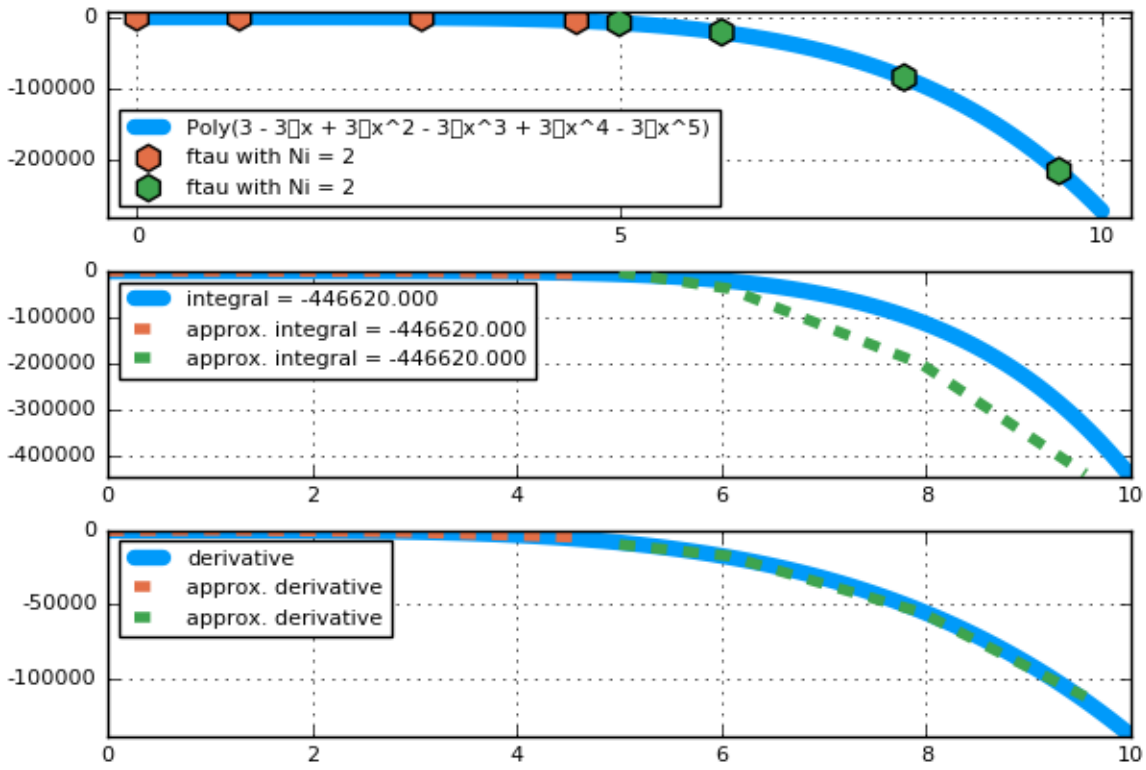
```
    Nc = Int64(3); # number of collocation points in each interval
    Ni = Int64(2);  # number of intervals
```



Test 1c

with:

```
    Nc = Int64(4); # number of collocation points in each interval
    Ni = Int64(2);  # number of intervals
```



- Conclusions
  - It seems, that using the multiple interval formulation sacrifices the property where
    - We can calculate a `Pth` order polynomial exactly with $2 * N - 2$ collocation points
    - We do not approximate a `5th` order polynomial with `6` total collocation points
    - Looking at Test 1c, we can see that when we use `N=4` we calculate the intergral exactly
      - So the property applies only to each interval
  - Test 1b and Test 1c both show that we are not calculating the end point

Approximation of $Y^{(k)}()$ -> ex#2 In the previous example, we neglected the approximation of the final state in each interval $Y^(k)()$.

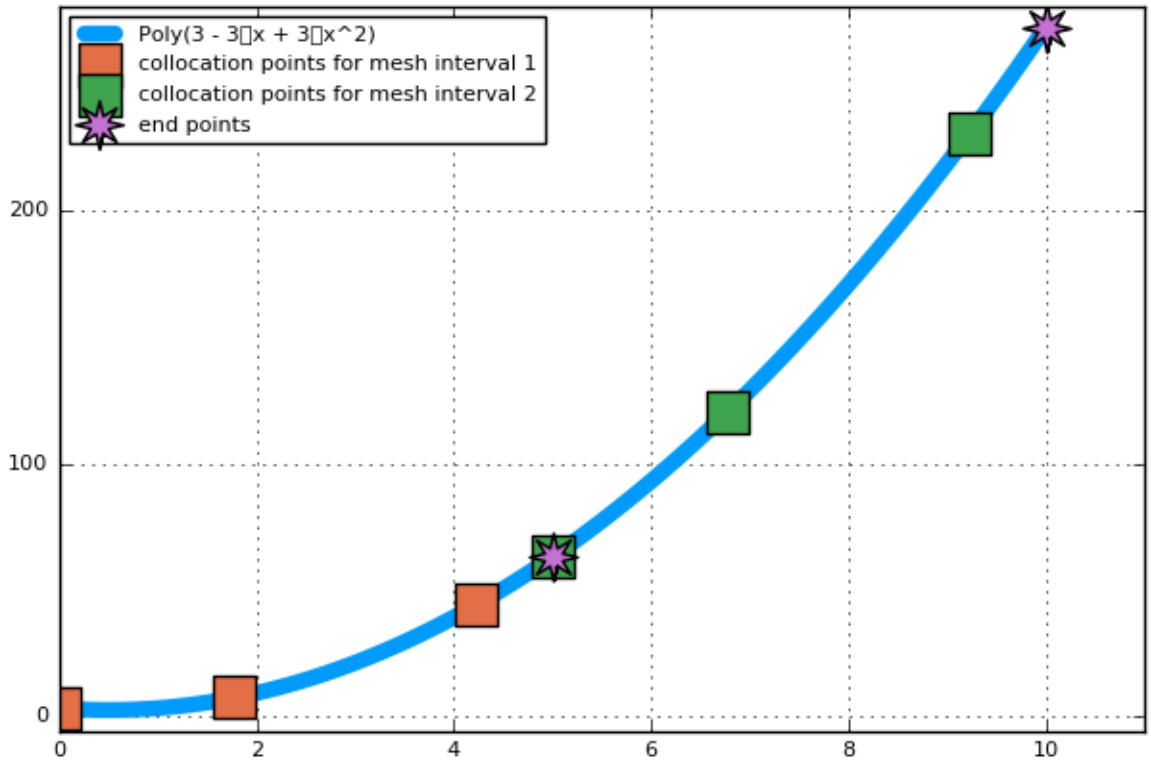
In this example, we will demonstrate calculation of this state.

where:

$$y(x) = 3x^2 - 3x + 3$$

with:

```
    Nc = Int64(3); # number of collocation points in each interval
    Ni = Int64(2);  # number of intervals
```



Why Do We Need This State?

It is needed to make the constraint that the states at the end of each mesh grid are equal.

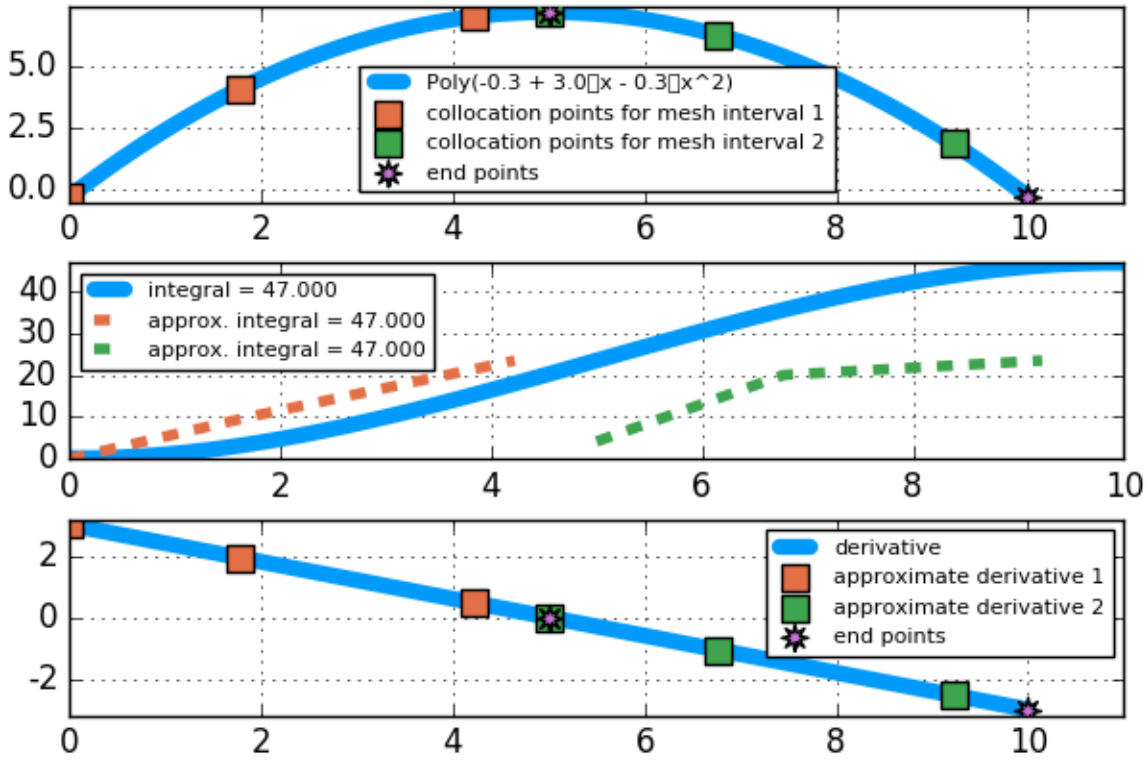
Approximation of State Derivative at of Mesh Grids -> ex#3 In this example, the state derivative at the end of each mesh.

where:

$$y(x) = -0.3x^2 + 3x - 0.3$$

with:

```
    Nc = Int64(3); # number of collocation points in each interval
    Ni = Int64(2);  # number of intervals
```



Why Do We Need This Derivative At the End Point?

Actually, we do not need this. There is no constraint on state dynamics at the end of each mesh grid using the method discussed in *[DGar11]*. This is an important point, that is described here *hp-psuedospectral method*.

Now, we will look at the $\mathbf{D}$ matrix used to calculate the derivatives above:

```
D =
4×4×2 Array{Float64,3}:
[:, :, 1] =
 -1.0         1.50639   -1.10639    0.6
 -0.210639   -0.155051   0.713568  -0.347878
  0.0506395  -0.233568  -0.644949   0.827878
 -0.0666667   0.276429  -2.00976    1.8

[:, :, 2] =
 -1.0         1.50639   -1.10639    0.6
 -0.210639   -0.155051   0.713568  -0.347878
  0.0506395  -0.233568  -0.644949   0.827878
 -0.0666667   0.276429  -2.00976    1.8
```

Notice that for each interval the $\mathbf{D}$ matrix is actually identical. This is quite an interesting observation indeed, because different inputs where used to calculate it, these are the nodes.

For the first interval:

```
0.0
1.77526
4.22474
5.0
```

For the second interval:

```
5.0
6.77526
9.22474
10.0
```

These nodes depend on the interval $t_0 -> t_f$ as well as the τ:

```
-1.0
-0.289898
 0.689898
```

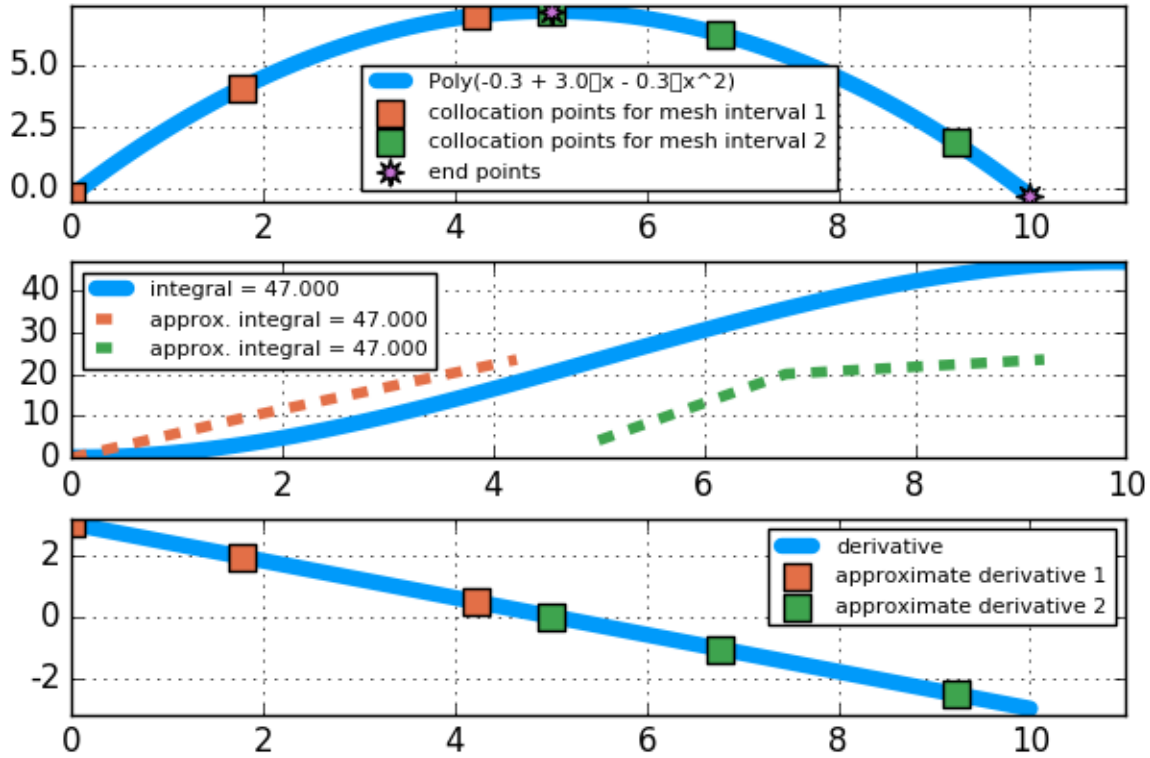
Which are the LGR nodes when $N_c = 3$

So, it seems that maybe we can calculate the weights beforehand as well as the $\mathbf{D}$ matrix and cache the result.

Neglecting Derivative At End Of Mesh For the purposes of using this method for control, again we do not need to calculate the derivative of the state at the ends of each mesh. So, we can remove the bottom row of the $\mathbf{D}$ matrix as:

```
D =
[:, :, 1] =
 -1.0          1.50639   -1.10639    0.6
 -0.210639    -0.155051   0.713568  -0.347878
  0.0506395   -0.233568  -0.644949   0.827878

[:, :, 2] =
 -1.0          1.50639   -1.10639    0.6
 -0.210639    -0.155051   0.713568  -0.347878
  0.0506395   -0.233568  -0.644949   0.827878
```



So, at the end of each mesh grid, we still approximate the state, but neglect it's derivative.

References

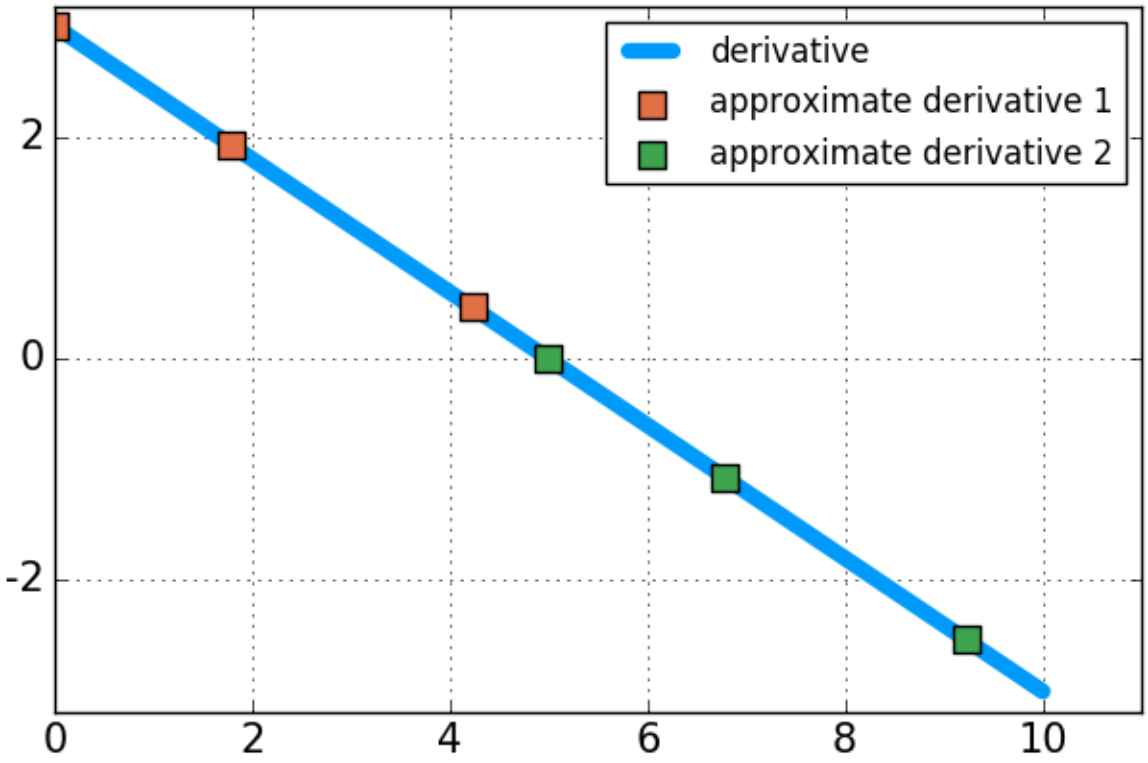
Higher Level Functionality -> ex#4 In this example, we are working on preparing the code for use with optimization by creating higher level functionality. Examine the IJulia notebook to see differences in code.

where:

$$y(x) = -0.3x^2 + 3x - 0.3$$

with:

```
Nc = Int64(3); # number of collocation points in each interval
Ni = Int64(2);  # number of intervals
```



These examples can be:

- Viewed remotely on using the jupyter nbviewer.
- Viewed locally and interacted using IJulia

  To do this in julia type:

```
using IJulia
notebook(dir=Pkg.dir("NLOptControl/examples/LGR_MI/"))
```

References

Optimal Control Problem Formulation

NLP Problem Initialization Here we are developing software to set up and keep track of all of the variables in the problem.

Functionality

ps, nlp = initialize_NLP(numStates=1,numControls=1,Ni=1,Nck=[2])

- To initialize the problem

To use:

```
using NLOptControl
ps, nlp = initialize_NLP(numStates=1,numControls=1,Ni=2,Nck=[2,3]);
```

ps = pseudospectral method related data:

```
NLOptControl.PS_data
  Nck: [2,3]
  Ni: 2
  τ: Array{Float64,1}[[-1.0,0.333333],[-1.0,-0.289898,0.689898]]
  ts: Array{Float64,1}[[-0.0,0.0],[-0.0,-0.0,0.0]]
  ω: Array{Float64,1}[[0.5,1.5],[0.222222,1.02497,0.752806]]
  ω: Array{Float64,1}[[0.0,0.0],[0.0,0.0,0.0]]
  t0: 0.0
  tf: 0.0
  DMatrix: Array{Float64,2}[[0.0 0.0 0.0; 0.0 0.0 0.0],[0.0 0.0 0.0 0.0; 0.0 0.0 0.0 0.0; 0.0 0.0 0.0
  IMatrix: Array{Float64,2}[[0.0 0.0; 0.0 0.0],[0.0 0.0 0.0; 0.0 0.0 0.0; 0.0 0.0 0.0]]
  stateMatrix: Array{Float64,2}[[0.0; 0.0; 0.0],[0.0; 0.0; 0.0; 0.0]]
  controlMatrix: Array{Float64,2}[[0.0; 0.0],[0.0; 0.0; 0.0]]
```

nlp = nonlinear programming problem related data:

```
NLOptControl.NLP_data
numStates: 1
numControls: 1
numStatePoints: [3,4]
numControlPoints: [2,3]
lengthStateVector: 7
lengthControlVector: 5
lengthDecVector: 14
timeStartIdx: 13
timeStopIdx: 14
stateIdx: Tuple{Int64,Int64}[(1,3),(4,7)]
controlIdx: Tuple{Int64,Int64}[(8,9),(10,12)]
stateIdx_all: Tuple{Int64,Int64}[(-99,-99)]
controlIdx_all: Tuple{Int64,Int64}[(-99,-99)]
stateIdx_st: Tuple{Int64,Int64}[(-99,-99)]
controlIdx_ctr: Tuple{Int64,Int64}[(-99,-99)]
decisionVector: [0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
```

Comments

- Lots of zeros right now because we still have a lot of initializing to do.
- *numControls* and *numStates* can both be greater than *1*.
- The value of *Ni* must match the length of *Nck*

generate_Fake_data(nlp,ps,γ)

- Generating some data to play with is useful:

nlp2ocp(nlp,ps)

- Turning the nonlinear-programming problem into an optimal control problem
  - This function basically takes a large design variable an sorts it back into the original variables

ζ, approx_int_st = integrate_state(ps,nlp) Approximates integral.

To use this function with Gaussian Quadrature (the default method):

```
ζ, approx_int_st = integrate_state(ps,nlp)
```

To use this function with the LGRIM:

```
integrate_state(ps,nlp;(:mode=>:LGRIM))
```

dζ = differentiate_state(ps,nlp) Approximate derivatives.

- Currently only using LGRDM as a method.

Examples

In these examples we use:

- Demonstrate functionality to setup optimal control problem
- Also include the development scripts of these functions
- There is not a webpage for all examples, but the interested user can check out them out using IJulia

NLP and OCP Functionality -> ex#2 In this example, we are continuing to work on preparing the code for use with optimization by creating higher level functionality. Examine the IJulia notebook to see code.

Aside from using the new data structures we demonstrate:

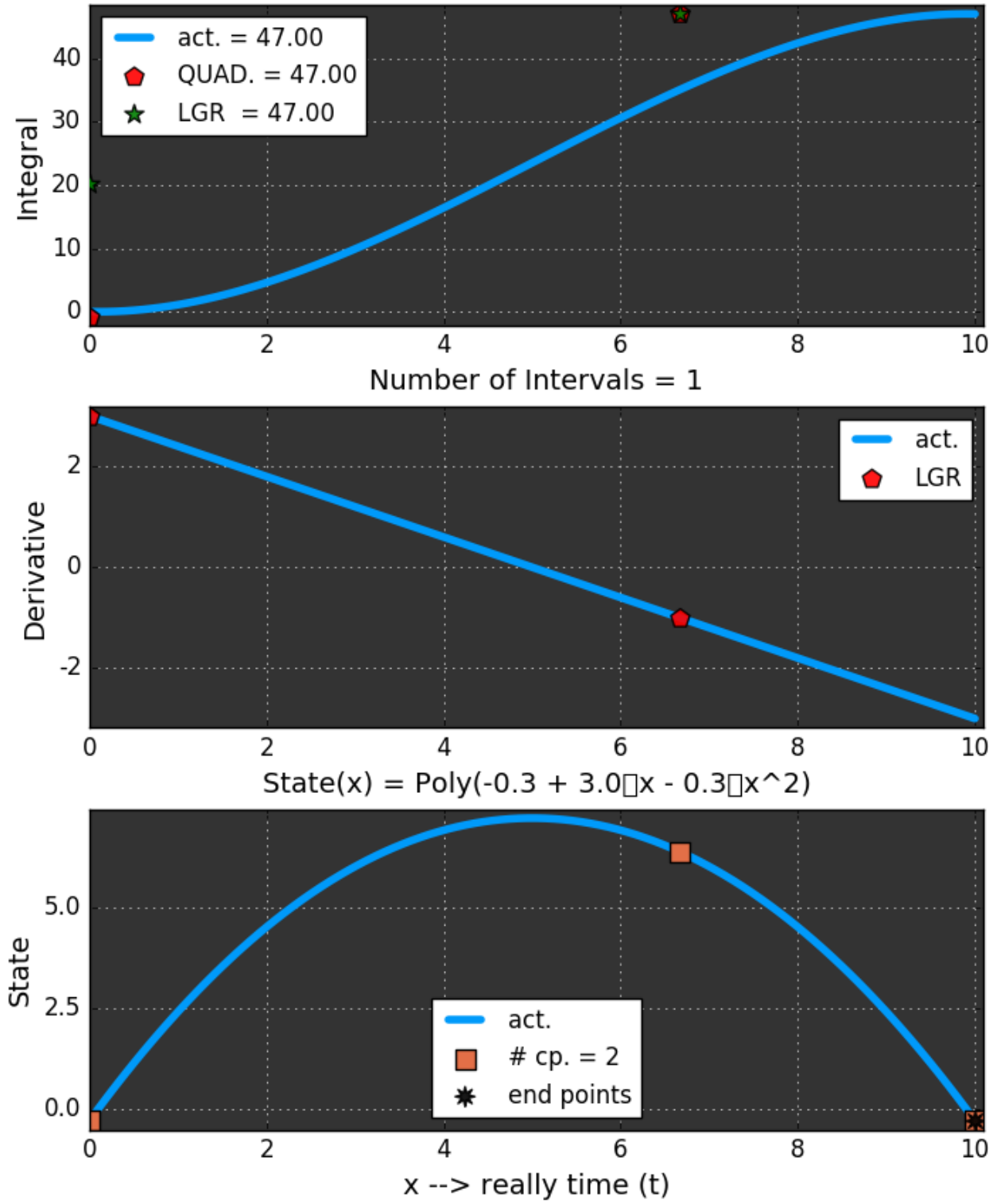
- Using the integration matrix for the first time
- Using the higher level functionality

where:

$$y(x) = -0.3x^2 + 3x - 0.3$$

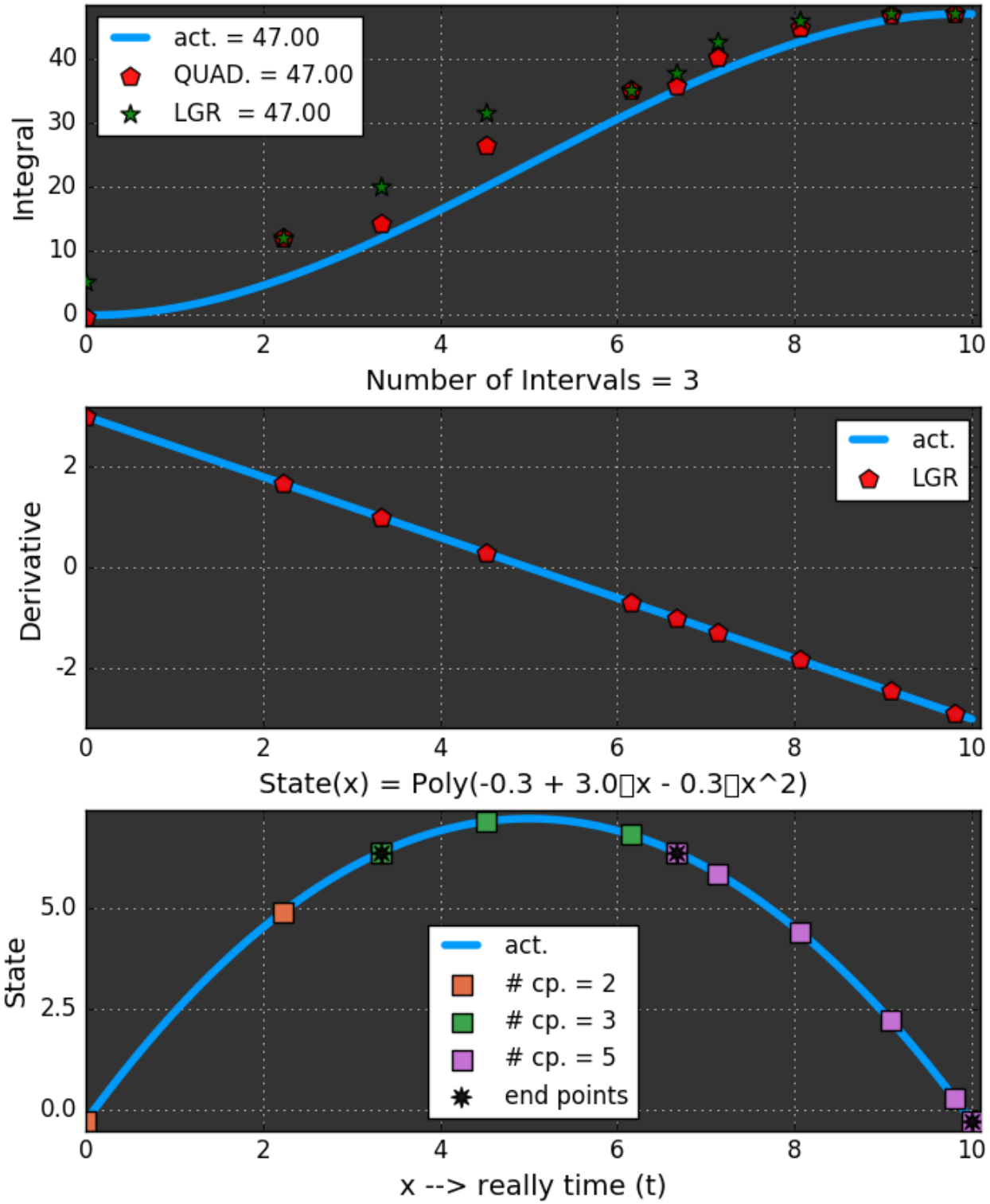
Single Interval Problem with:

```
ps, nlp = initialize_NLP(numStates=1,numControls=1,Ni=1,Nck=[2]);
```



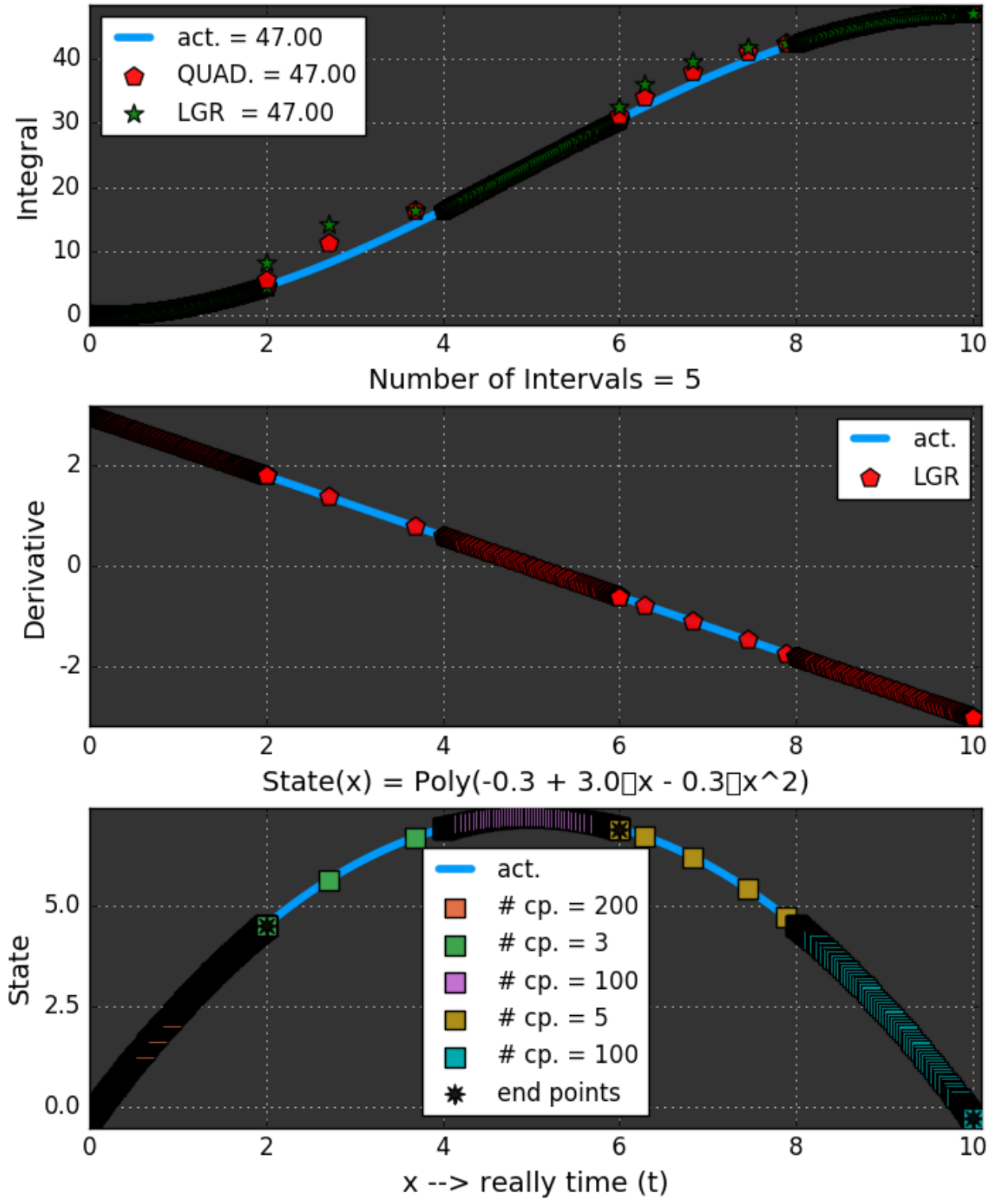
Multiple Interval Problem A with:

```
ps, nlp = initialize_NLP(numStates=1,numControls=1,Ni=3,Nck=[2,3,5]);
```



Multiple Interval Problem B with:

```
ps, nlp = initialize_NLP(numStates=1,numControls=1,Ni=5,Nck=[200,3,100,5,100]);
```



Multiple States -> ex#3 In this example, we are going to approximate the *5th* order Taylor series polynomial for *sin()* and *cos()* expanded about *x=0*.

where:

$$sin(x)P_5(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!}$$

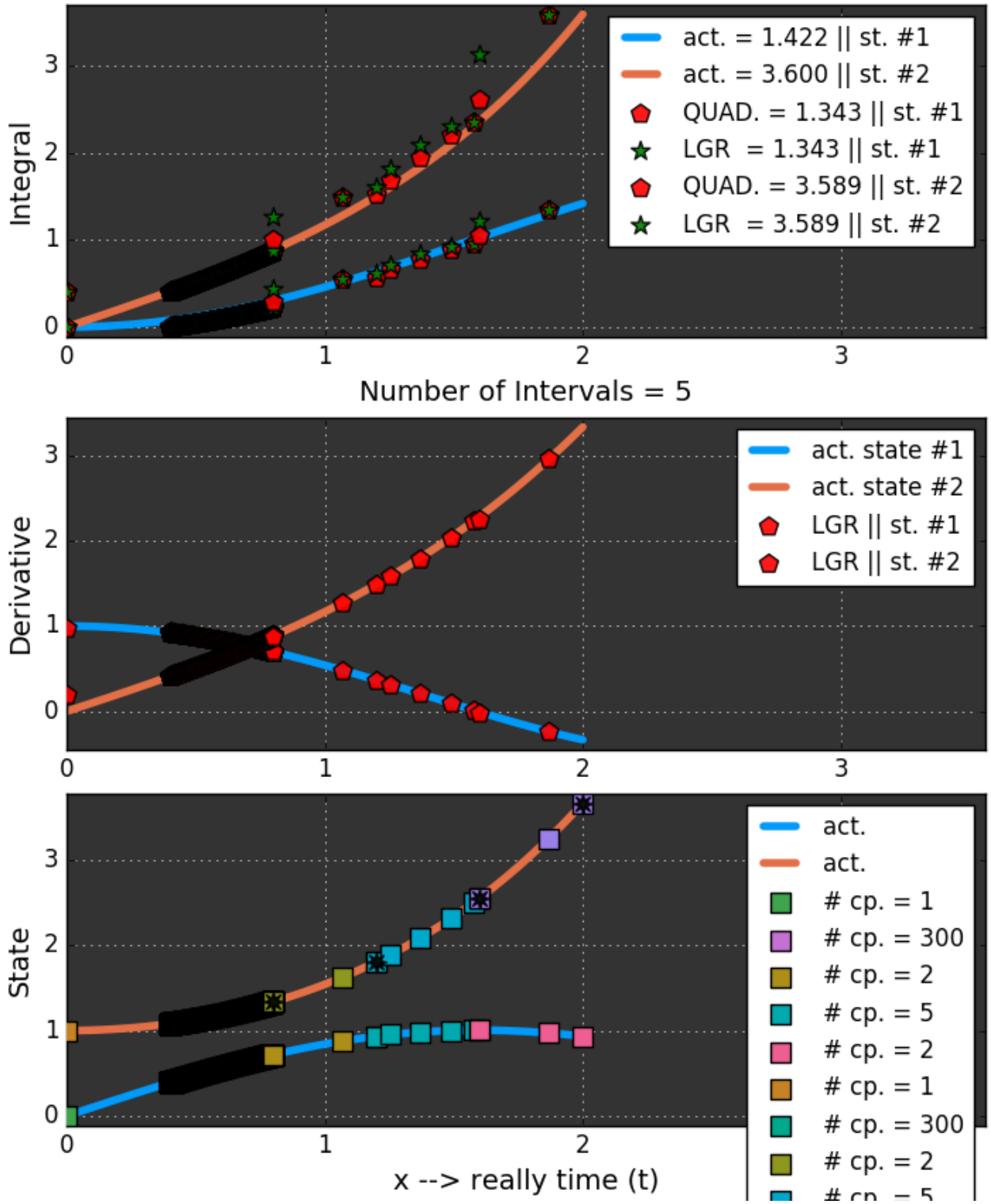
$$cos(x)P_5(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!}$$

and:

```
t0 = Float64(0); tf = Float64(2);
```

Problem A with:

```
ps, nlp = initialize_NLP(numStates=1,numControls=1,Ni=5,Nck=[200,3,100,5,100]);
```

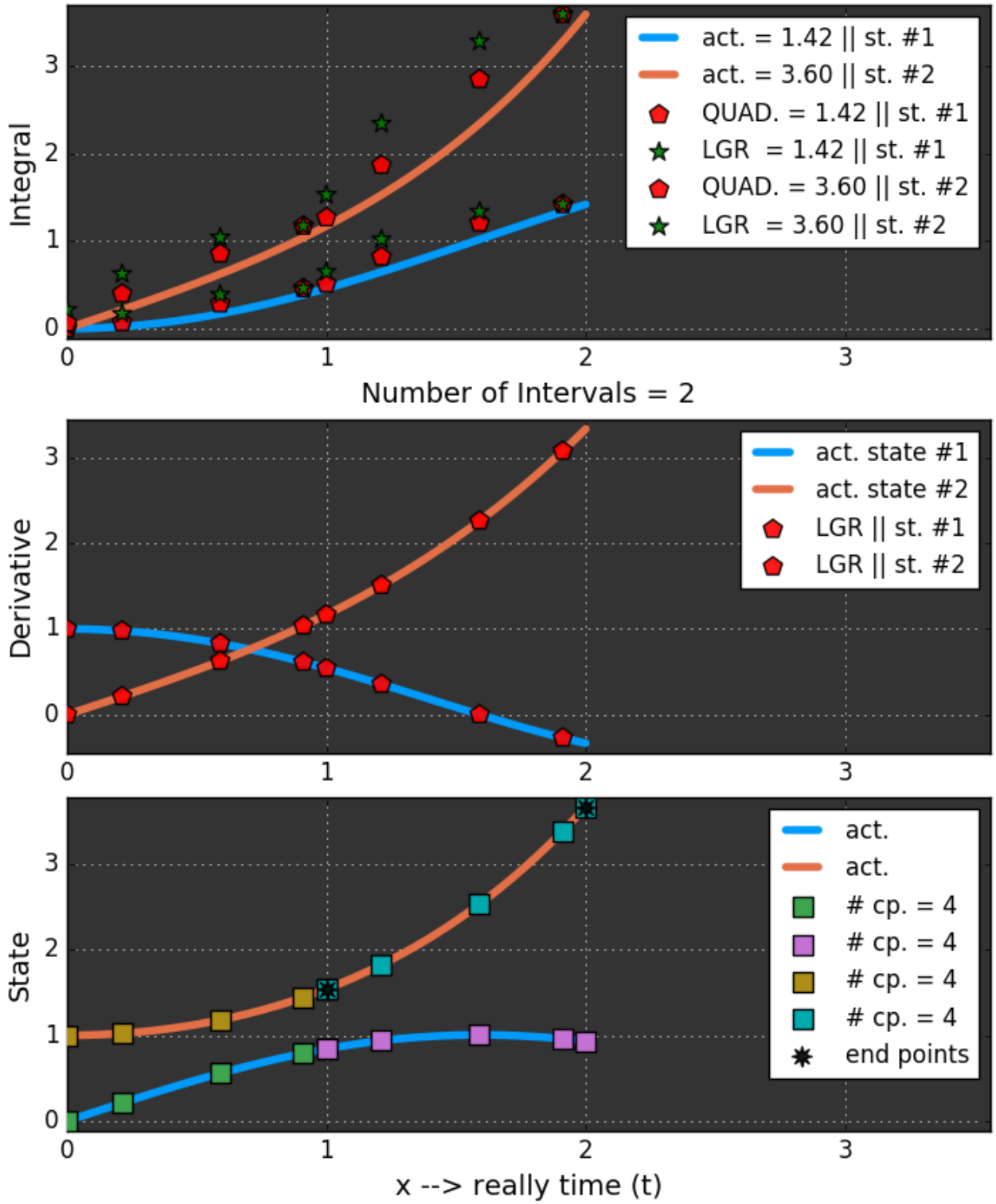


Comments on failing to calculate integral in Problem A

This is expected. Looking at the smallest mesh grid size *Nck=1*, we can only expect to calculate a the integral for a *2*1-2=0th* order polynomial exactly.

Problem B with:

```
ps, nlp = initialize_NLP(numStates=numStates,numControls=1,Ni=2,Nck=[4,4]);
```



These examples can be:

- Viewed remotely on using the jupyter nbviewer.

- Viewed locally and interacted using IJulia

  To do this in julia type:

```
using IJulia
notebook(dir=Pkg.dir("NLOptControl/examples/NLP2OCP/"))
```

Researchers at the University of Florida

Describe this method in many papers including *[EDar11][EGar11][EGPH+10][EGPH+09]*.

References

Developing Code

This site documents current progress and functionality that is being developed.

Current Focus

Old Records This is for documentation that was created where something was still being worked on where:

1. Everything was not working as expected, but has now been fixed or is obsolete.
2. Removed Functionality

1.3 Bibliography

http://docs.juliadiffeq.org/latest/tutorials/ode_example.html#In-Place-Updates-1

https://github.com/JuliaDiffEq/ParameterizedFunctions.jl#ode-macros

http://docs.juliadiffeq.org/latest/tutorials/ode_example.html#Defining-Systems-of-Equations-Using-ParameterizedFunctions.jl-1 @ChrisRackauckas after looking at ParameterizedFunctions.jl, I realized that I have many many parameters and I it would be too much to write them all after the ''end" is there a way that I can include a "parameters.jl" file to execute this? Also, I am currently automatic differentiation, and I noticed that currently ParameterizedFunctions.jl uses symbolic differentiation. I remember you mentioning that you use automatic differentiation, can this also

Bibliography

[BGar11] Divya Garg. *Advances in global pseudospectral methods for optimal control*. PhD thesis, University of Florida, 2011.

[BHer15] Daniel Ronald Herber. Basic implementation of multiple-interval pseudospectral methods to solve optimal control problems. *UIUC-ESDL-2015-01*, 2015.

[BSTW11] Jie Shen, Tao Tang, and Li-Lian Wang. *Spectral methods: algorithms, analysis and applications*. volume 41. Springer Science & Business Media, 2011.

[AHer15] Daniel Ronald Herber. Basic implementation of multiple-interval pseudospectral methods to solve optimal control problems. *UIUC-ESDL-2015-01*, 2015.

[ASTW11] Jie Shen, Tao Tang, and Li-Lian Wang. *Spectral methods: algorithms, analysis and applications*. volume 41. Springer Science & Business Media, 2011.

[A1] Christopher L Darby. *hp–Pseudospectral Method for Solving Continuous-Time Nonlinear Optimal Control Problems*. PhD thesis, University of Florida, 2011.

[A2] Divya Garg. *Advances in global pseudospectral methods for optimal control*. PhD thesis, University of Florida, 2011.

[A3] Divya Garg, Michael Patterson, William W Hager, Anil V Rao, David A Benson, and Geoffrey T Huntington. A unified framework for the numerical solution of optimal control problems using pseudospectral methods. *Automatica*, 46(11):1843–1851, 2010.

[A4] Divya Garg, Michael A Patterson, William W Hager, Anil V Rao, David A Benson, and Geoffrey T Huntington. An overview of three pseudospectral methods for the numerical solution of optimal control problems. *Advances in the Astronautical Sciences*, 135(1):475–487, 2009.

[DGar11] Divya Garg. *Advances in global pseudospectral methods for optimal control*. PhD thesis, University of Florida, 2011.

[CDar11] Christopher L Darby. *hp–Pseudospectral Method for Solving Continuous-Time Nonlinear Optimal Control Problems*. PhD thesis, University of Florida, 2011.

[CGar11] Divya Garg. *Advances in global pseudospectral methods for optimal control*. PhD thesis, University of Florida, 2011.

[CGPH+10] Divya Garg, Michael Patterson, William W Hager, Anil V Rao, David A Benson, and Geoffrey T Huntington. A unified framework for the numerical solution of optimal control problems using pseudospectral methods. *Automatica*, 46(11):1843–1851, 2010.

[CGPH+09] Divya Garg, Michael A Patterson, William W Hager, Anil V Rao, David A Benson, and Geoffrey T Huntington. An overview of three pseudospectral methods for the numerical solution of optimal control problems. *Advances in the Astronautical Sciences*, 135(1):475–487, 2009.

[EDar11] Christopher L Darby. *hp–Pseudospectral Method for Solving Continuous-Time Nonlinear Optimal Control Problems*. PhD thesis, University of Florida, 2011.

[EGar11] Divya Garg. *Advances in global pseudospectral methods for optimal control*. PhD thesis, University of Florida, 2011.

[EGPH+10] Divya Garg, Michael Patterson, William W Hager, Anil V Rao, David A Benson, and Geoffrey T Huntington. A unified framework for the numerical solution of optimal control problems using pseudospectral methods. *Automatica*, 46(11):1843–1851, 2010.

[EGPH+09] Divya Garg, Michael A Patterson, William W Hager, Anil V Rao, David A Benson, and Geoffrey T Huntington. An overview of three pseudospectral methods for the numerical solution of optimal control problems. *Advances in the Astronautical Sciences*, 135(1):475–487, 2009.