Nyquist vs Banchy-Crooks

This Julia Pluto-notebook is part of the supplementary material for the paper

> Dirk Oliver Theis, *"Proper" Shift Rules for Derivatives of Perturbed-Parametric Quantum Evolutions*, arXiv:2207.01587

Copyright and license information

Address inquiries to:

```
University of Tartu
Centre for Entrepreneurship and Innovation
Narva mnt 18
51009 Tartu linn,
Tartu linn, Tartumaa
Estonia
+372 737 4809
eik@ut.ee
https://eik.ut.ee
```

Introduction

This Pluto notebook is concerned with methods to estimate derivatives, with respect to θ of parameterized quantum expectation-values of the form

$$f\colon (t,\theta) \mapsto \mathrm{tr}\big(M e^{it(\theta A+B)/\hbar} \varrho e^{-it(\theta A+B)/\hbar}\big) \qquad (*)$$

The unitary $e^{it(\theta A+B)/\hbar}$ expresses a quantum evolution with Hamiltonian $H := -(\theta A + B)$ for a time t.

It implements, for the sake of comparison, the following methods:

- Banchi-Crooks' *Stochastic Approximate Parameter Shift Rule* arXiv:2005.10299 and
- the *truncated Nyquist shift rule* arXiv:2207.01587.

When implemented on quantum devices, these methods are estimators, i.e., the output is random and the expected output is off from the sought derivative by a (hopefully only) small bias (approximation error).

In this comparison, we are not interested in the stochastic properties (which, on paper, are essentially identical) but in the magnitude of the approximation error.

Both methods require large magnitudes of the θ parameter in order to approximate the derivative well. We will compare the effect of the magnitude of θ on the approximation error in both methods.

Julia & Pluto setup

```
begin
    using PlutoUI ✓
    using Plots ✓
    import PlotlyBase ✓
    plotly()
    md"""

    #### Julia & Pluto setup
    """
end
```

```
PlotlyBase 0.8.18 is not compatible with this version of Plots. The declared
compatibility is 0.7.
```

```
using LinearAlgebra ✓: Hermitian, Diagonal, tr, eigvals, eigvecs, isposdef
```

```
using QuadGK ✓      # for numerical integration
```

```
using Zygote ✓      # for automatic differentiation (to test against)
```

```
using Statistics ✓  # for percentiles and whatnot
```

We use syntactic sugar to make Julia look more like math: "⋅" instead of "∗" etc, "¬" instead of "!"...

Expectation-value function and data

We make available a Julia function $f(t, \theta; \texttt{::Data})$, along with helpers and variants.

We take $h := 1$ for the Planck constant, i.e., $\hbar = 1/2\pi$. You don't like it, suck it.

`Data = @NamedTuple{M,ϱ,A,B}` is a helper data structure to hold the four Hermitian matrices.

Perturbed-parametric unitary function

$$U : (t, \theta) \mapsto e^{2\pi i t(\theta A + B)}$$

U

Function `U(t::ℝ, θ::ℝ ; A::Hermitian{ℂ}, B::Hermitian{ℂ}) ::Matrix{ℂ}`

Expectation-value function

$$f : (t, \theta) \mapsto \mathrm{tr}(M\, U(t,\theta)\, \varrho\, U(t,\theta)^\dagger)$$

f

Function

```
f(t::ℝ, θ::ℝ
  ;     D::Data ) ::ℝ
```

Randomized input data

To compare the methods, in (∗) above, we choose

- M a random Hermitian matrix with eigenvalues in $\{\pm 1\}$;
- ϱ a random positive definite trace-1 matrix
- A a random Hermitian matrix with eigenvalues in $\{\pm 1\}$;
- B a random Hermitian matrix with iid standard-normal complex entries.

This setting corresponds to the most common application in quantum computing: The observable is a Pauli operator, and the 1-qubit drive is a Pauli rotation. (Our setting is negligibly more general.)

`gimme_data`

Function `gimme_data(d ::Int) :: Data`

Let's make some data, just for fun...

The dimension is `5`.

- Give me a new set of matrices, hop, hop!

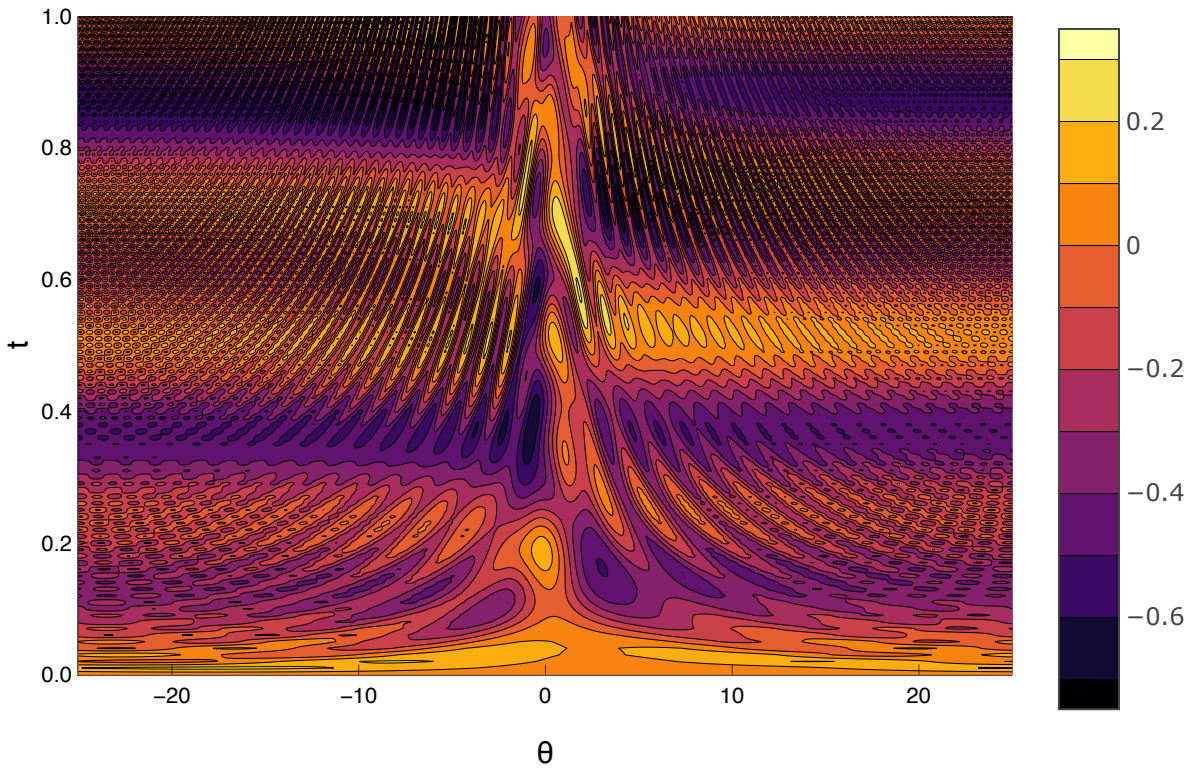
Eigenvalues:

```
M: [-1.0000000000000002, -0.9999999999999997, -0.9999999999999994, 0.9999999999999992,
ϱ: [0.004813290237324075, 0.02503293603676747, 0.1217502520654448, 0.296027079841554, 0
A: [-1.0000000000000007, -0.9999999999999998, -0.9999999999999989, 1.0, 1.0000000000000
B: [-2.690058192248532, -1.4578896192902717, -0.10268151540595387, 0.4736110967284113,
```

... and plot it...

... or not: Plotting takes several seconds. Do you really want to plot? Yes

Plotting:



Analytic derivative by Julia

`j∂`

Function `j∂(θ ; D::Data)` makes available the "true" derivative of the expectation value function $\boldsymbol{\theta} \mapsto \boldsymbol{f(1, \theta)}$ for the given data, using Julia's automatic differentiation based on code reflection (Zygote package).

Implementation of Banchi-Crooks

Expectation-value function for BC ASPSR

`f_asp2()` is the function under the integral in Banchi-Crooks ASPSR rule to approximate ∂f. For the derivative of f as in (∗), in `f_asp2(` s,θ,ε `)`, the unitary $e^{2\pi i(\theta A+B)}$ in (∗) is replaced by

$$e^{2\pi i s(\theta A+B)}e^{2\pi i\varepsilon(\pm\frac{1}{8\varepsilon}A+B)}e^{2\pi i(1-s)(\theta A+B)}$$

With the "±" matching the superscript plus and minus in the following expression, the function `f_asp2()` is used in the BC's ASPSR as follows:

$$\partial f(\theta)\approx 2\pi\int_0^1\left(\,f_{\text{asp}}(s,\theta,\varepsilon)^+-f_{\text{asp}}(s,\theta,\varepsilon)^-\,\right)ds$$

`f_asp2`

Function

```
f_asp2( s::ℝ, θ ::ℝ, ε ::ℝ
            ;
            D ::Data   ) ::@NamedTuple{plus::ℝ,minus::ℝ}
```

returns the results of two evaluations of the "Approximate Stochastic Parameter" function as in Algorithm 3 of Banchi & Crooks; the two values are those for m=±1.

BC deterministic derivative

We implement Banchi-Crooks pseudo-shift rule deterministically, with the "shots" (Monte-Carlo integration) replaced by numerical integration (QuadGK package).

The BC method has a parameter, ε, affecting the accuracy.

`bc_apx`

Function `bc_apx(θ::ℝ ; ε ::ℝ, order ::Int, D ::Data ) ::@NamedTuple{∂::ℝ,err::ℝ}`

Original BC pseudo-shift rule for H with eigenvalues ±1.

Performs the numerical, deterministic approximation of the derivative at `θ`.

Numerical integration adds the parameter `order`: It goes into the QuadGK numerical integration package (the $\log_{10}$ of the maximum number of function evaluations).

Return value: Named tuple w/ 1st entry the derivative, `∂`, 2nd entry the numerical error of the integration, `err`.

Implementation of Nyquist shift rule

Helpers

`f_1 2`

Function

```
f_1 2(θ::ℝ, a::ℝ
                ; T ::ℝ,
                  D ::Data ) ::@NamedTuple{plus::ℝ,minus::ℝ}
```

two `f`-values: $f(1,\theta-a)$, and $f(1,\theta+a)$; each one of them is replaced by 0 if the parameter value falls outside of the interval $[-T,+T]$.

Truncated Nyquist shift rule

We approximate the (analytical!) Nyquist derivative by truncating the sum at the evaluation points of the Banchi-Crooks derivative, $1/\varepsilon$ (where ε is as in `bcapx()`).

`nytrun`

Function `nytrun(θ ; ε , D::Data)::ℝ`

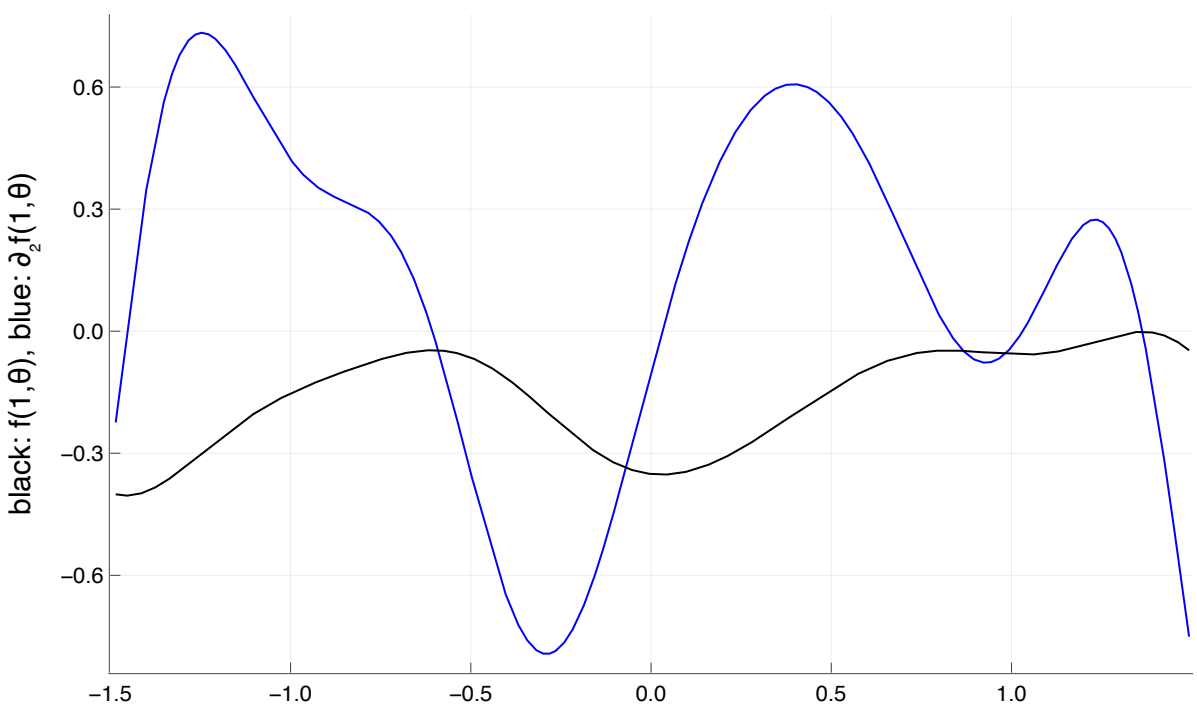
Approximates the Nyquist derivative deterministically by truncating the sum in such a way that the parameter values stay within $[-T, +T]$ for $T := \pi/4\epsilon$. In other way, the parameter values stay within the same window as in Banchi-Crooks.

Visualization

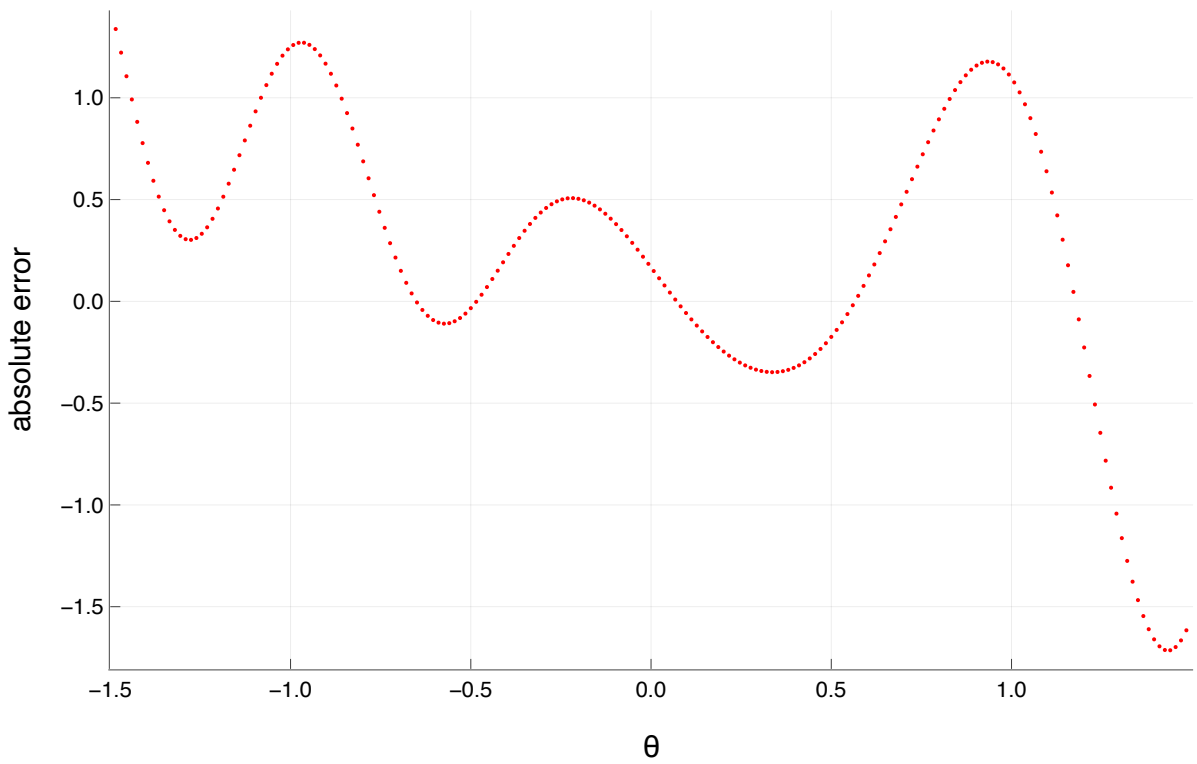
Define the quantities:

- Plot window: [-1.5 , 1.5]
- Number of plot points in the window: 200
- Banchi-Crooks $\varepsilon = e \cdot 10^{-\ell}$ where $e =$ 1.0 and $\ell =$ 1
- Numerical integration order parameter 10

Expectation value function and derivative (error-free automatic differentiation by Julia):

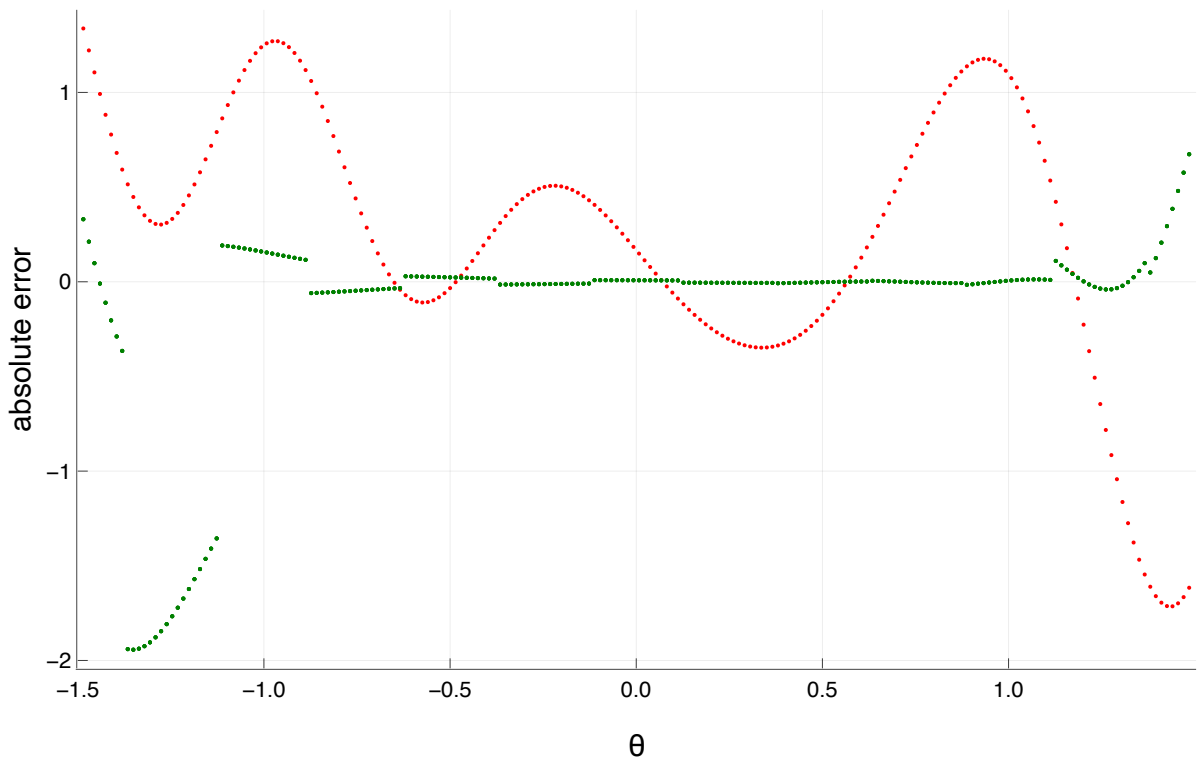


Plot BC error



Plot truncated-Nyquist error (green)

Include BC-error in plot? ☑



Comparison

Data structures and functions for creating the data.

`make_data`

Function `make_data(l; dim=2, RNDDATA_ITER=100, SAMPLE_PTS, order=20)`

Parameters

- `l` determines the range of ε's, it will be $x \cdot 10^{l}$ for $x \in [1,10[$
- `SAMPLE_PTS` is an iterator or iterable or so.

Return value:

- Named tuple `(εs,errors_bc,errors_ny,true_vals)`
- `θs` is a vector of the θ's
- `errors_`xy is a 2-dim array of data points
- `true_vals` is a 2-dim array of the true derivatives

In the arrays, 1st dim is repetition idx, 2nd dim is θ-idx

```
const Stats_t =
NamedTuple{(:mean, :median, :perc01, :perc10, :perc25, :perc90, :perc99, :min, :max), NTup
```

`get_stats`

Function `get_stats(θs,errors_bc,errors_ny,true_vals ; relative=false)`

Input here is the output of `make_data()`

Return value is a named tuple `(bc_err, ny_err, bc_err_minus_ny_err)` of vectors (indexed corresponding to `θs`) of `Stats_t`

Let's get cracking!

How much data should be produced?

- $\varepsilon = 10^{-\ell}$ where $\ell =$ 1
- Dimension: 7
- Number of random expectation-value functions (M, ϱ, A, B as described above) 100
- Number of sample points: 50

Now making data. This will take some time. The terminal will show error messages (such as numerical issues).

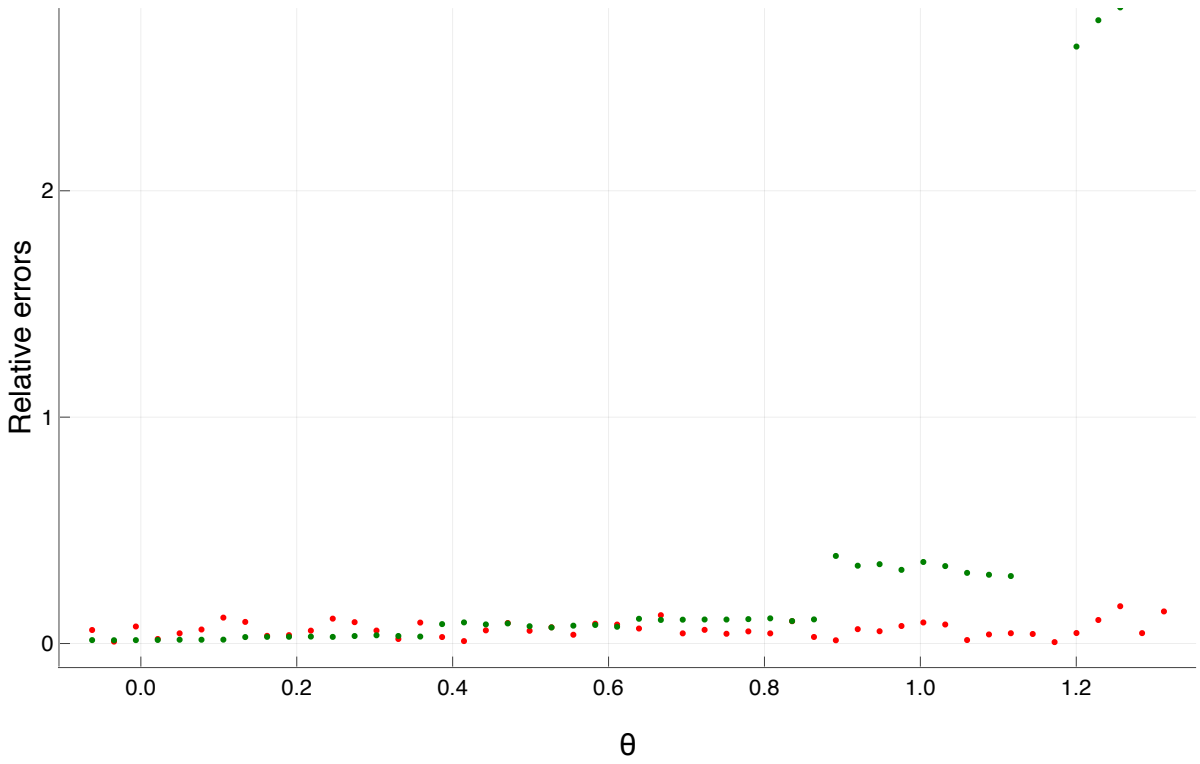
[redacted]

Relative errors: ☑

The next figure shows

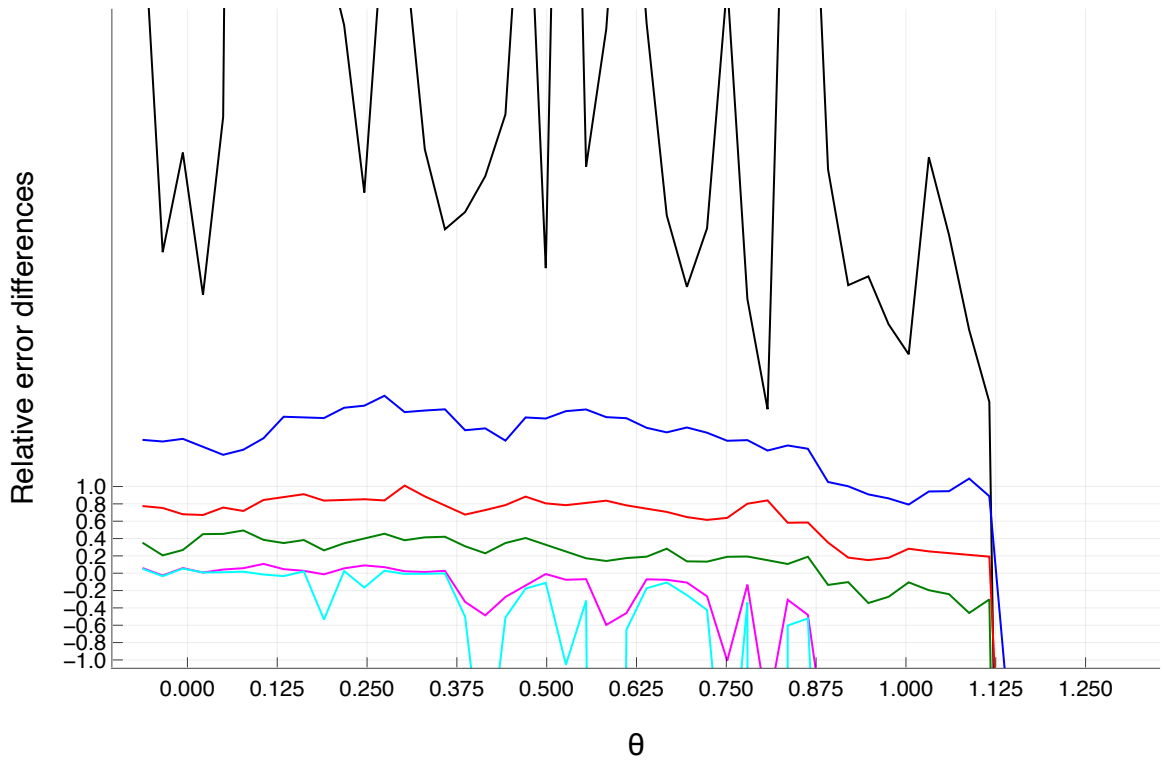
- BC (red): 1st percentile, 10th percentile ☐, median ☐, 90th percentile ☐
- Ny (green): median, 90th percentile ☐, 99th percentile ☐, max ☐

Show it at all? ☑



The next figure shows statistics for the difference of absolute error for each individual data point, $\mathrm{err}_{\mathrm{BC}} - \mathrm{err}_{\mathrm{Ny}}$. The quantity is positive if Nyquist is better than BC, otherwise negative. The statistics derived from that data set which are shown in the figures are, top to bottom:

- mean (black) ☑
- median (blue) ☑
- 25th-percentile (red) ☑
- 10th-percentile (green) ☑
- 1st-percentile (magenta) ☑
- minimum (cyan) ☑.


Relative error differences
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0.000
0.125
0.250
0.375
0.500
0.625
0.750
0.875
1.000
1.125
1.250
θ