A Python package for time scales

Tom Cuchta (Fairmont State University)

Dynamical equations on time scales workshop
Będlewo, Poland, 12-16 June 2019

History

- Matthias Baur took independent study in time scales
- Research courses
- Summer work

Basics

Hosted on github:
`https://github.com/tomcuchta/timescalecalculus`
This talk assumes the following commit:
`139f56545ccc6ee3f3acbc65e0b1fc34520e79b7`
Loading the package: `import timescalecalculus as tsc`

Defining time scales

Current limitation: time scales that are finite unions of singletons and intervals of the form $[a, b]$
Built-in time scales: time scale of integers

```
>>> tsintegers=tsc.integers(-3,10)
Timescale successfully constructed:
Timescale: [-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Timescale name: integers from -3 to 10
```

quantum time scale

```
>>> tsc.quantum(2,0,10)
Timescale successfully constructed:
Timescale: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
Timescale name: quantum numbers 2^0 to 2^10
```

Defining any supported time scale

Suppose we want $\mathbb{T} = \{0, 1\} \bigcup \left[\frac{3}{4}, \frac{9}{7}\right] \bigcup \{5\}$.
Can use standard `Fraction` package to get perfect answers.

```
>>> from fractions import Fraction
>>> tsEXAMPLE=tsc.timescale([0,[Fraction(3,4),Fraction(9,7)],5])
Timescale successfully constructed:
Timescale: [0, [Fraction(3, 4), Fraction(9, 7)], 5]
Timescale name: none
```

Built-in functions for every time scale (σ)

Continue using $\mathbb{T} = \{0\} \bigcup \left[\frac{3}{4}, \frac{9}{7}\right] \bigcup \{5\}$ defined by

```
>>> tsEXAMPLE=tsc.timescale([0,[Fraction(3,4),Fraction(9,7)],5])
```

Forward jump:

```
>>> tsEXAMPLE.sigma(1)
1
>>> tsEXAMPLE.sigma(1.123)
1.123
>>> tsEXAMPLE.sigma(0)
Fraction(3, 4)
>>> tsEXAMPLE.sigma(Fraction(9,7))
5
```

Must be **careful** with fraction syntax:

```
>>> tsEXAMPLE.sigma(9/7)
Fraction(3, 4)
```

Built-in functions for every time scale (μ)

Continue using $\mathbb{T} = \{0\} \bigcup \left[\frac{3}{4}, \frac{9}{7}\right] \bigcup \{5\}$ defined by

```
>>> tsEXAMPLE=tsc.timescale([0,[Fraction(3,4),Fraction(9,7)],5])
```

Graininess:

```
>>> tsEXAMPLE.mu(1)
0
>>> tsEXAMPLE.mu(1.123)
0
>>> tsEXAMPLE.mu(0)
Fraction(3, 4)
>>> tsEXAMPLE.mu(Fraction(9,7))
Fraction(26, 7)
```

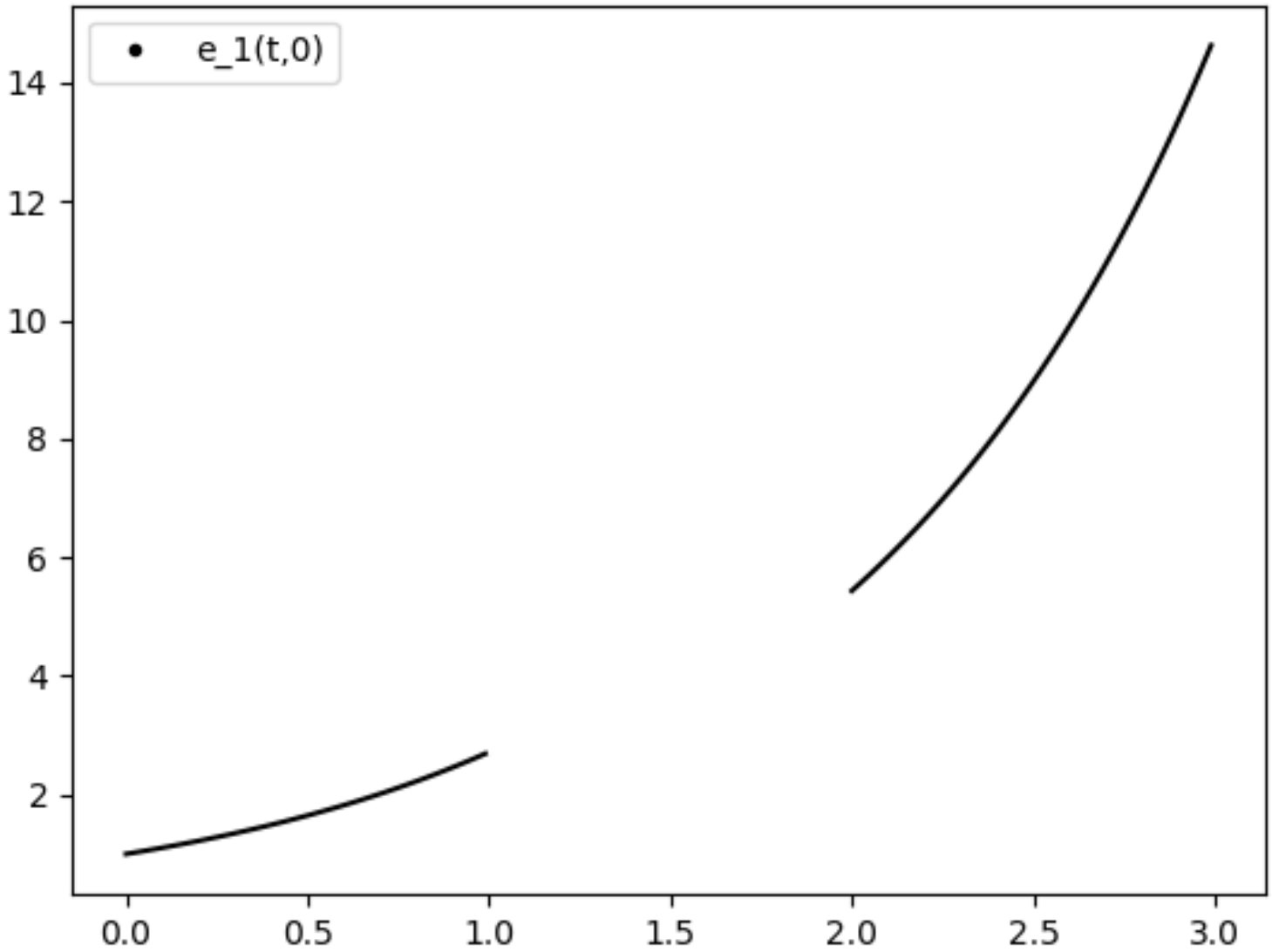
Built-in functions on time scales

e_p, $\sin_p$, $\cos_p$, ξ_h, $\ominus$, $\oplus$, σ, μ

Exponential function

In [**?**, Example 2.58], the time scale $\mathbb{P}_{1,1} = [0,1] \cup [2,3] \cup [4,5] \cup [6,7] \cup \dots$ is studied alongside the exponential equation $N^{\Delta} = N, \quad N(0) = 1$. Easily we get $N(t) = e_1(t,0)$.

Exponential function



Solving a second order dynamic equation

Well-known: the general solution of

$$y^{\Delta\Delta} = -\alpha^2 y$$

is [**?**, Theorem 3.31]

$$y(t) = c_1 \cos_\alpha(t, s) + c_2 \sin_\alpha(t, s)$$

Setting $\alpha = 2$ and applying initial conditions $y(0) = 0, \quad y^\Delta(0) = 2$ gives solution

$$y(t) = \sin_2(t, 0).$$

Does it work?

$y^{\Delta\Delta} = -4y, y(0) = 0, \quad y^{\Delta}(0) = 2$

Convert the equation into a system of first order equations:

$$\begin{cases} y_1 = y \\ y_2 = y^{\Delta}, \end{cases}$$

hence

$$\begin{cases} y_1^{\Delta} = y_2 \\ y_2^{\Delta} = -4y_1 \end{cases}$$

with initial conditions $y_1(0) = 0$, $y_2(0) = 2$.

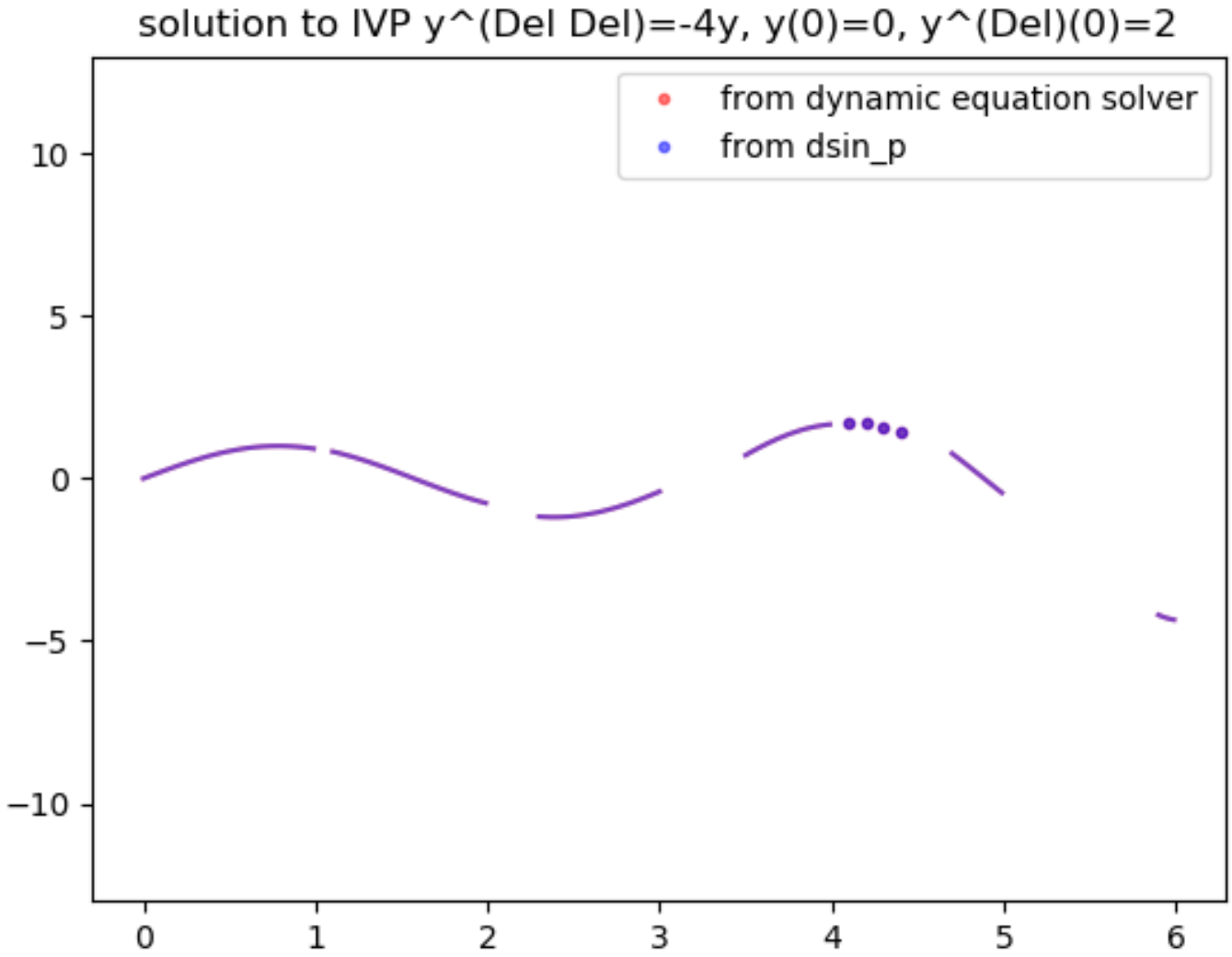
$y^{\Delta\Delta} = -4y, y(0) = 0, \quad y^{\Delta}(0) = 2$

To solve the dynamic IVP on a time scale named `ts`:

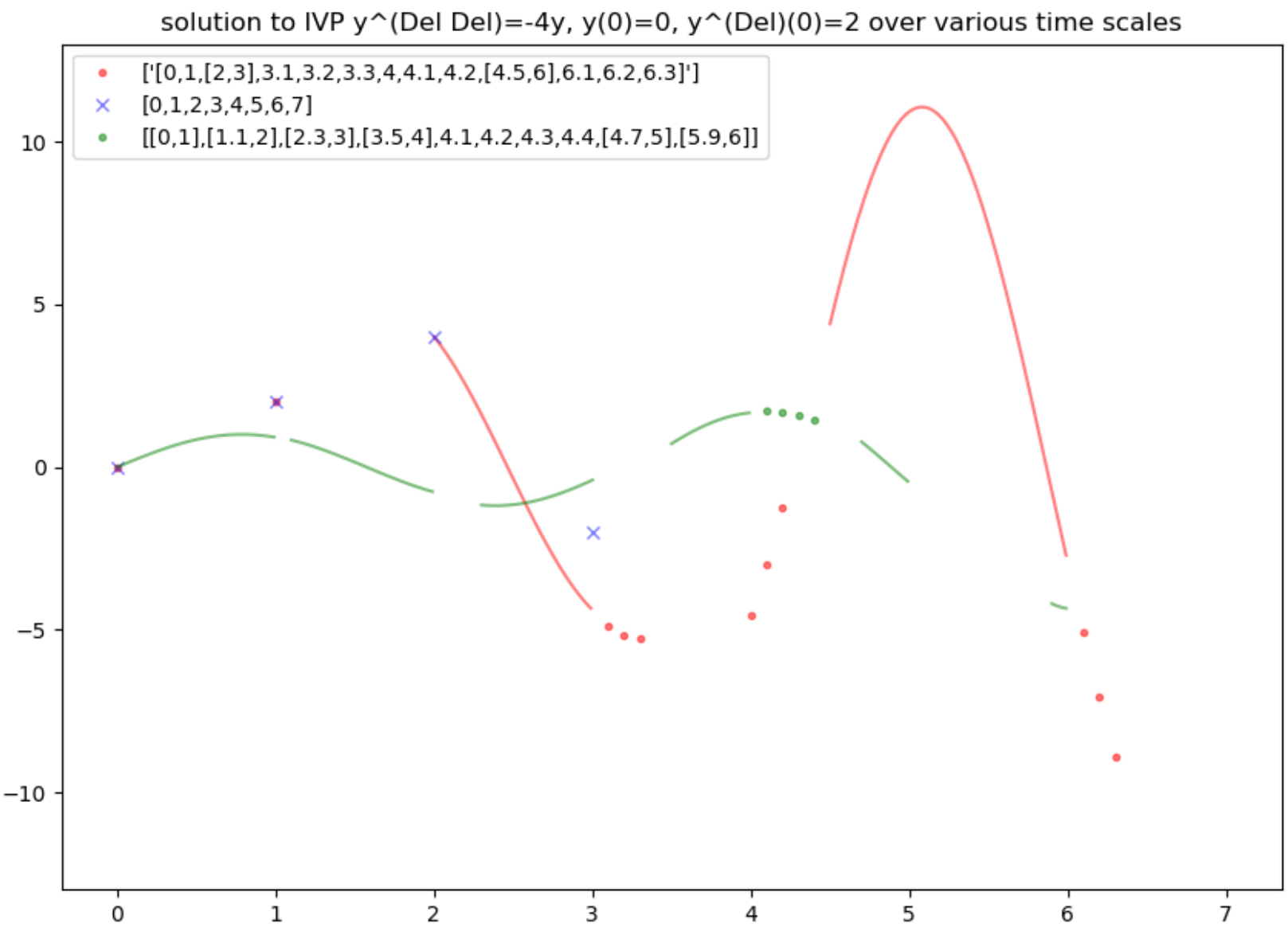
```
def y_prime_vector(vector, t):
    y1, y2 = vector
    dt_vector=[y2,-4*y1]
    return dt_vector

def soln(t):
    return ts.solve_ode_system_for_t([0,2], 0, t, y_prime_vector)
```

Compare this to the function `ts.dsin_p` for various time scales.

$y^{\Delta\Delta} = -4y, y(0) = 0, \quad y^{\Delta}(0) = 2$



$y^{\Delta\Delta} = -4y, y(0) = 0, \quad y^{\Delta}(0) = 2$



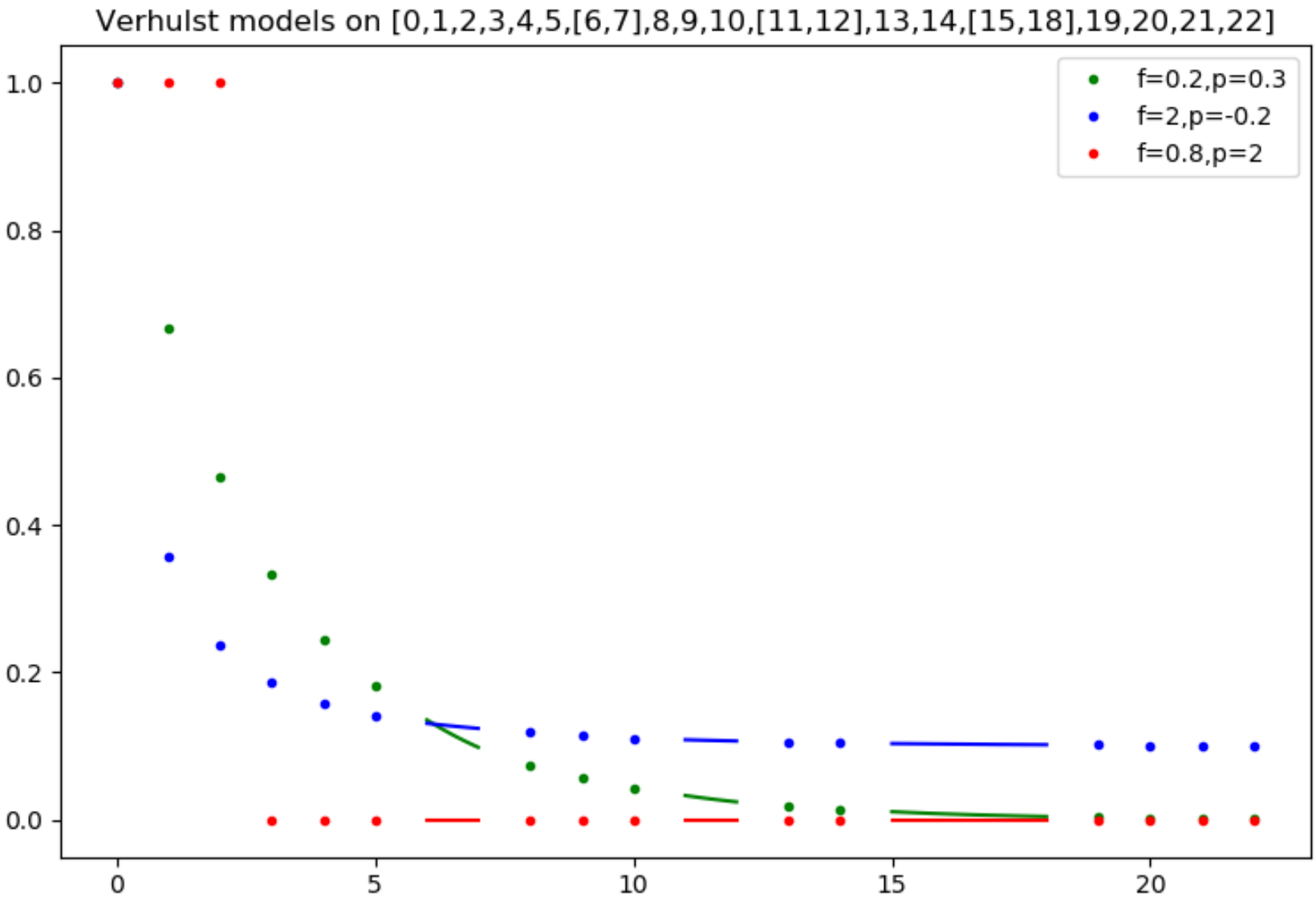
A Verhulst model

From [**?**, (2.24)], a Verhulst equation on time scales may be defined by

$$y^{\Delta} = [\ominus(p + fy)]y$$

```
def y_prime_vector(vector, t):
    y1 = vector[0]
    def verhulst(y,t):
        return p(t)+f(t)*y
    dt_vector=[ts.mucircleminus(lambda x: verhulst(y1,x),t)*y1]
    return dt_vector

def soln(t):
    return ts.solve_ode_system_for_t([1], 0, t, y_prime_vector)
```

A Verhulst model



SIS model

From [**?**, (3.1)],

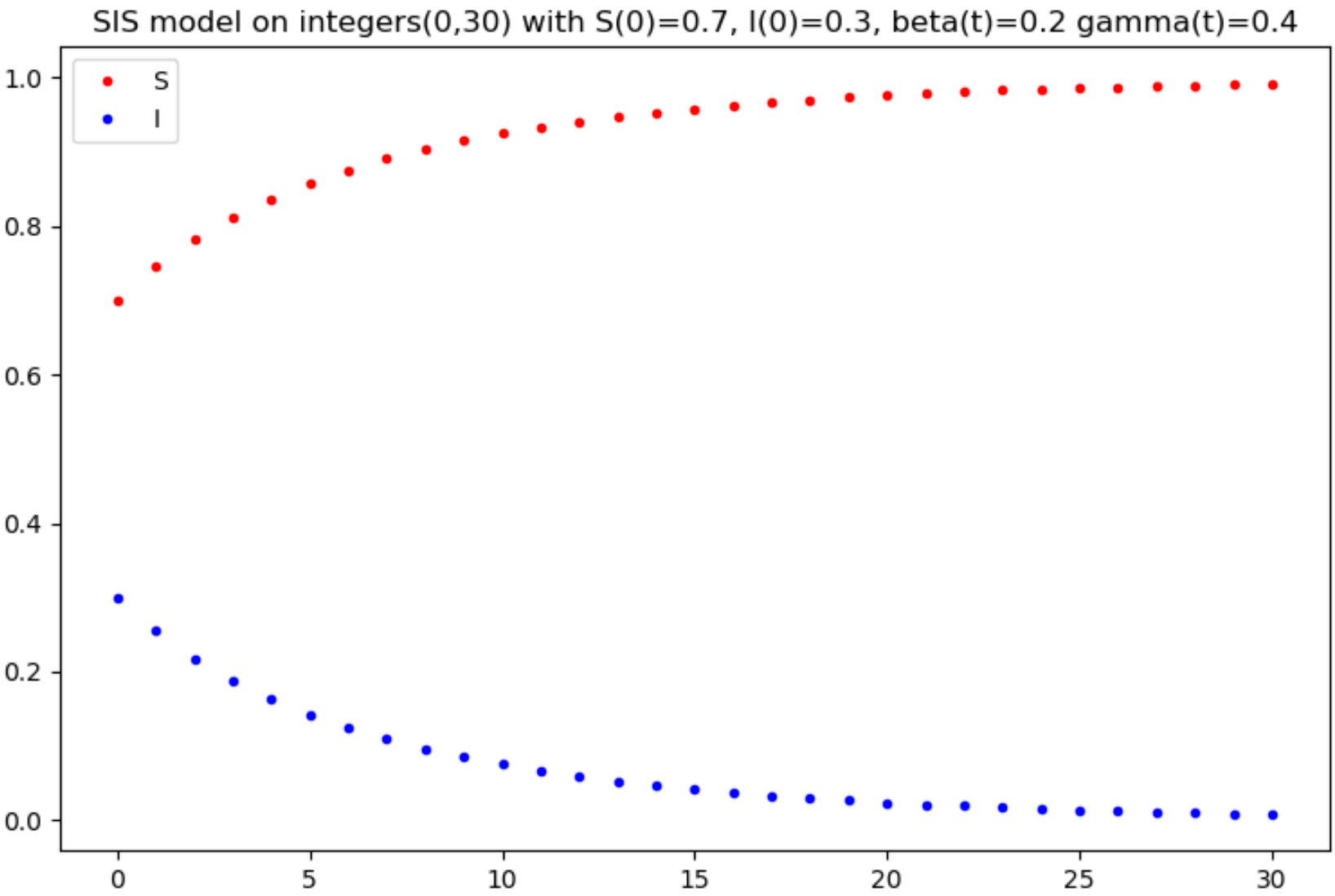
$$\begin{cases} S^{\Delta} &= -\beta S^{\sigma} I + \gamma I \\ I^{\Delta} &= \beta S^{\sigma} I - \gamma I. \end{cases}$$

Since $I^{\Delta} = -S^{\Delta}$, finding S^{Δ} is sufficient. Using $S^{\sigma} = S + \mu S^{\Delta}$ yields

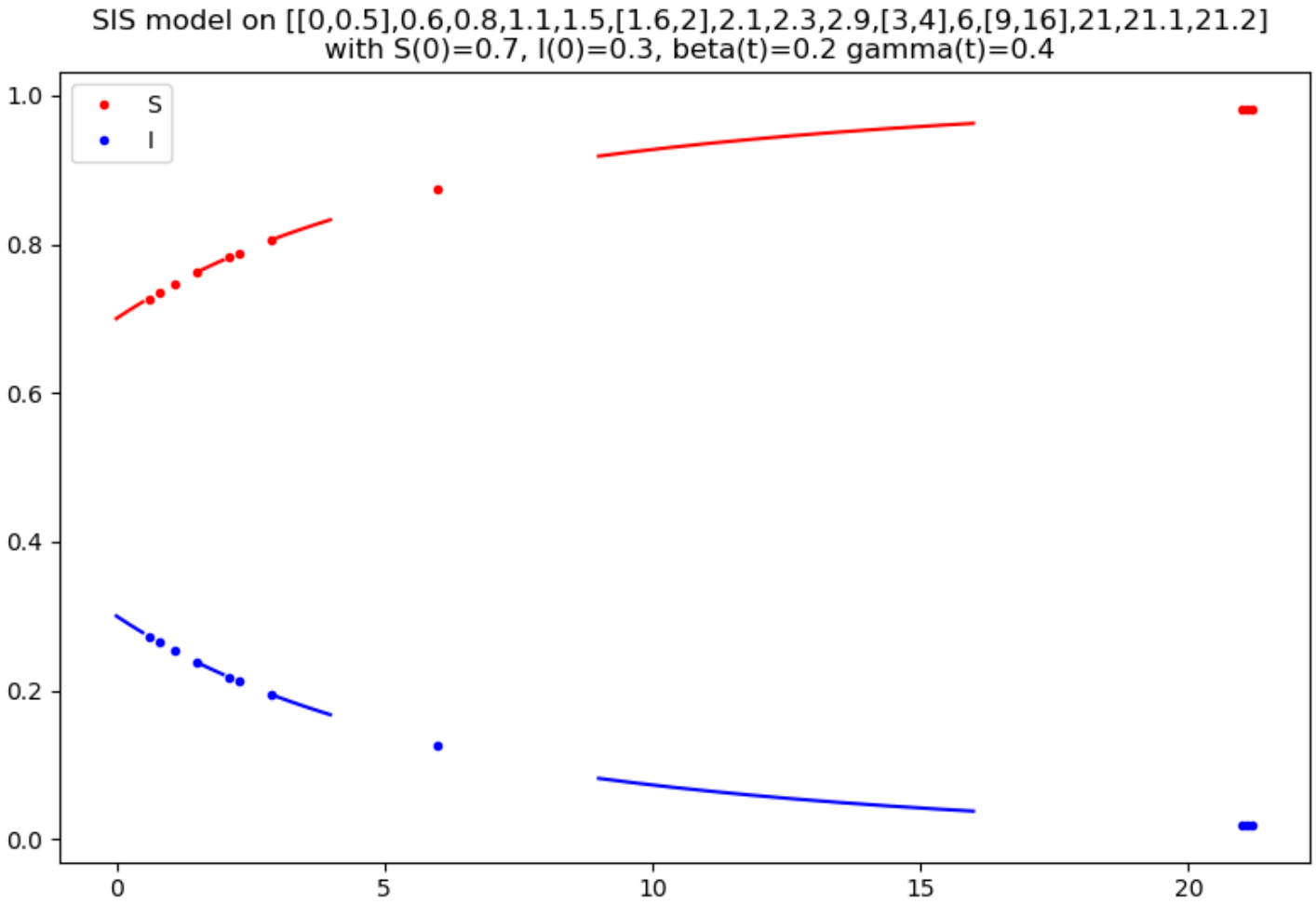
$$S^{\Delta} = \ominus(\beta I)\left[S - \frac{\alpha}{\beta}\right]$$

```
def y_prime_vector(vector, t):
    S, I = vector
    dt_vector=[ts.mucircleminus(lambda x: beta(x)*I,t)*
    (S-gamma(t)/beta(t)),
    (-1)*ts.mucircleminus(lambda x: beta(x)*I,t)*
    (S-gamma(t)/beta(t))]
    return dt_vector
```

SIS model


SIS model on integers(0,30) with S(0)=0.7, I(0)=0.3, beta(t)=0.2 gamma(t)=0.4
S
I
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

SIS model


SIS model on [[0,0.5],0.6,0.8,1.1,1.5,[1.6,2],2.1,2.3,2.9,[3,4],6,[9,16],21,21.1,21.2]
with S(0)=0.7, I(0)=0.3, beta(t)=0.2 gamma(t)=0.4
S
I
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20

SIR model

From

$$\begin{cases} x^{\Delta} = -\dfrac{b(t)xy^{\sigma}}{x+y} \\ y^{\Delta} = \dfrac{b(t)xy^{\sigma}}{x+y} - c(t)y^{\sigma} \\ z^{\Delta} = c(t)y^{\sigma} \\ x, y > 0 \end{cases}$$

By algebra, can conclude

$$y^{\Delta} = \frac{bxy - (x+y)cy}{x+y-bx\mu+(x+y)c\mu}.$$

So,

$$y^{\sigma} = y + \mu y^{\Delta} = y + \mu \frac{bxy - (x+y)cy}{x+y-bx\mu+(x+y)c\mu}$$

SIR model

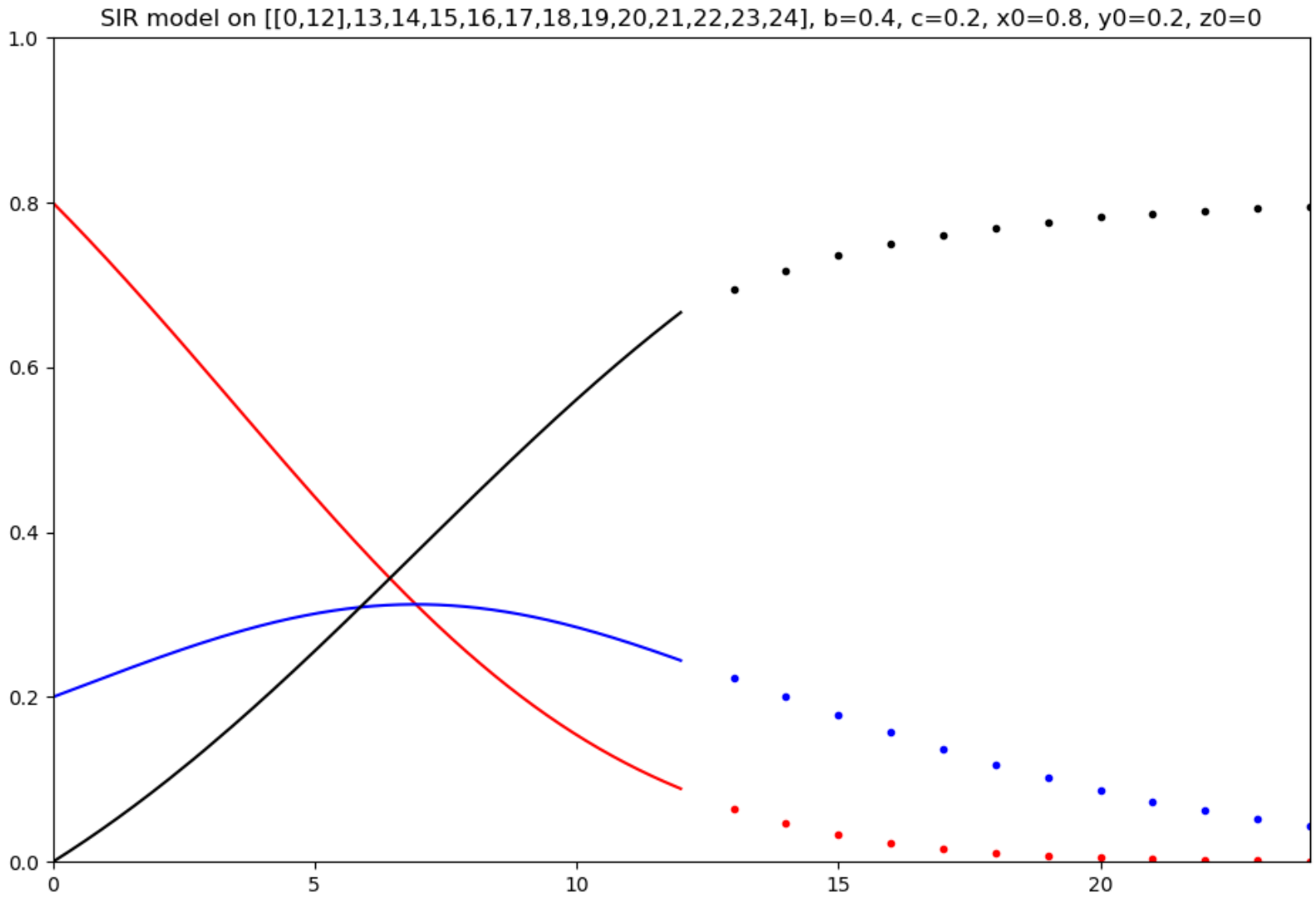
Therefore

$$\begin{cases} x^{\Delta} = \left(-\dfrac{bx}{x+y}\right)\left(y+\mu\dfrac{bxy-(x+y)cy}{x+y-bx\mu+(x+y)c\mu}\right) \\ y^{\Delta} = \dfrac{bxy-(x+y)cy}{x+y-bx\mu+(x+y)c\mu} \\ z^{\Delta} = cy+\mu\dfrac{bxy-(x+y)cy}{x+y-bx\mu+(x+y)c\mu} \\ x,y>0 \end{cases}$$

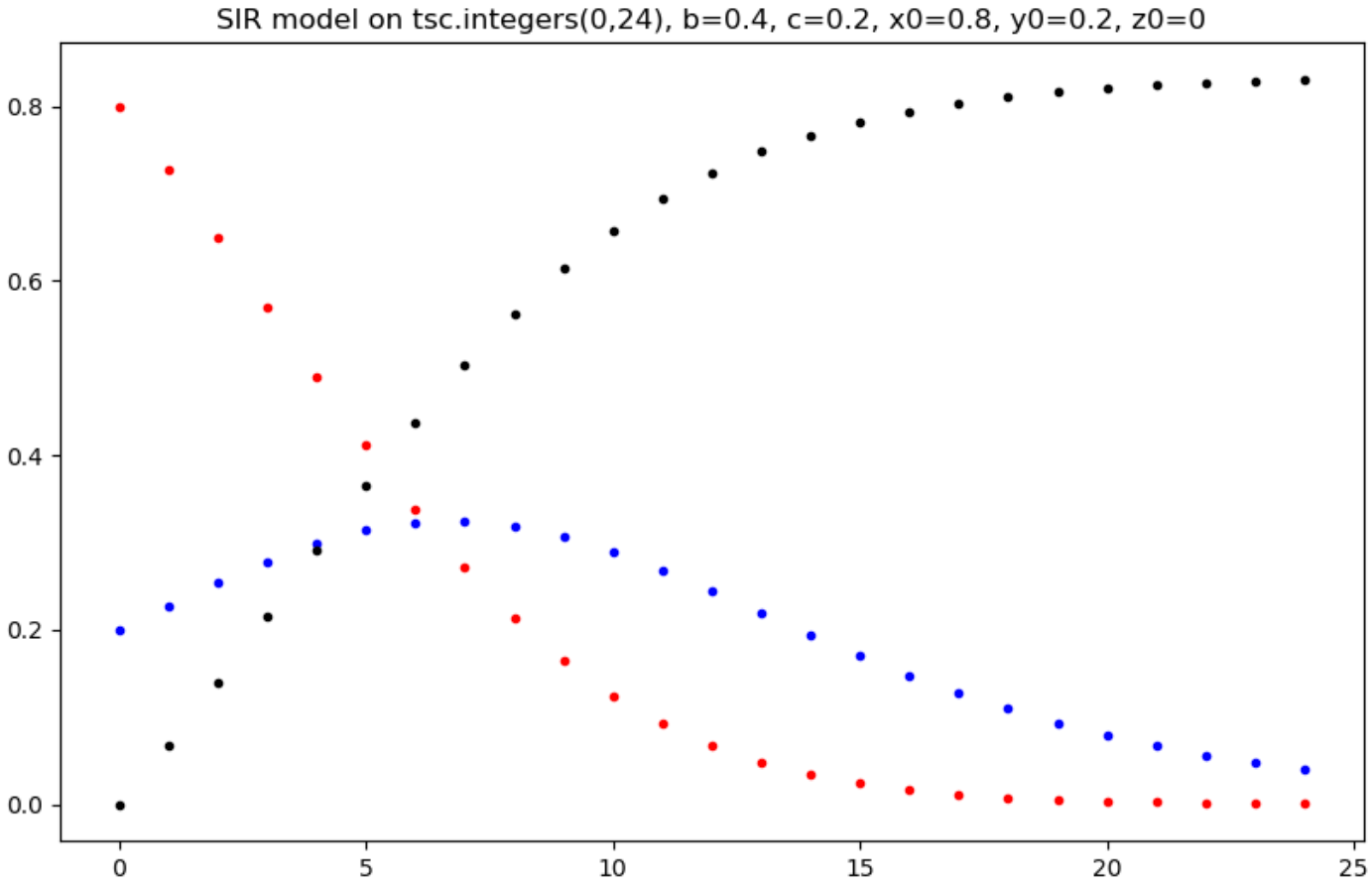
becomes

```
def y_prime_vector(vector, t):
    y1, y2, y3 = vector
    def bigpart(b,c,y1,y2,y3,t):
        return (b*y1*y2-(y1+y2)*c*y2)/(y1+y2-b*y1*ts.mu(t)+(y1+y2)*
    dt_vector=[(-b(t)*y1/(y1+y2))*(y2+ts.mu(t)*bigpart(b(t),c(t),y1
return dt_vector
```

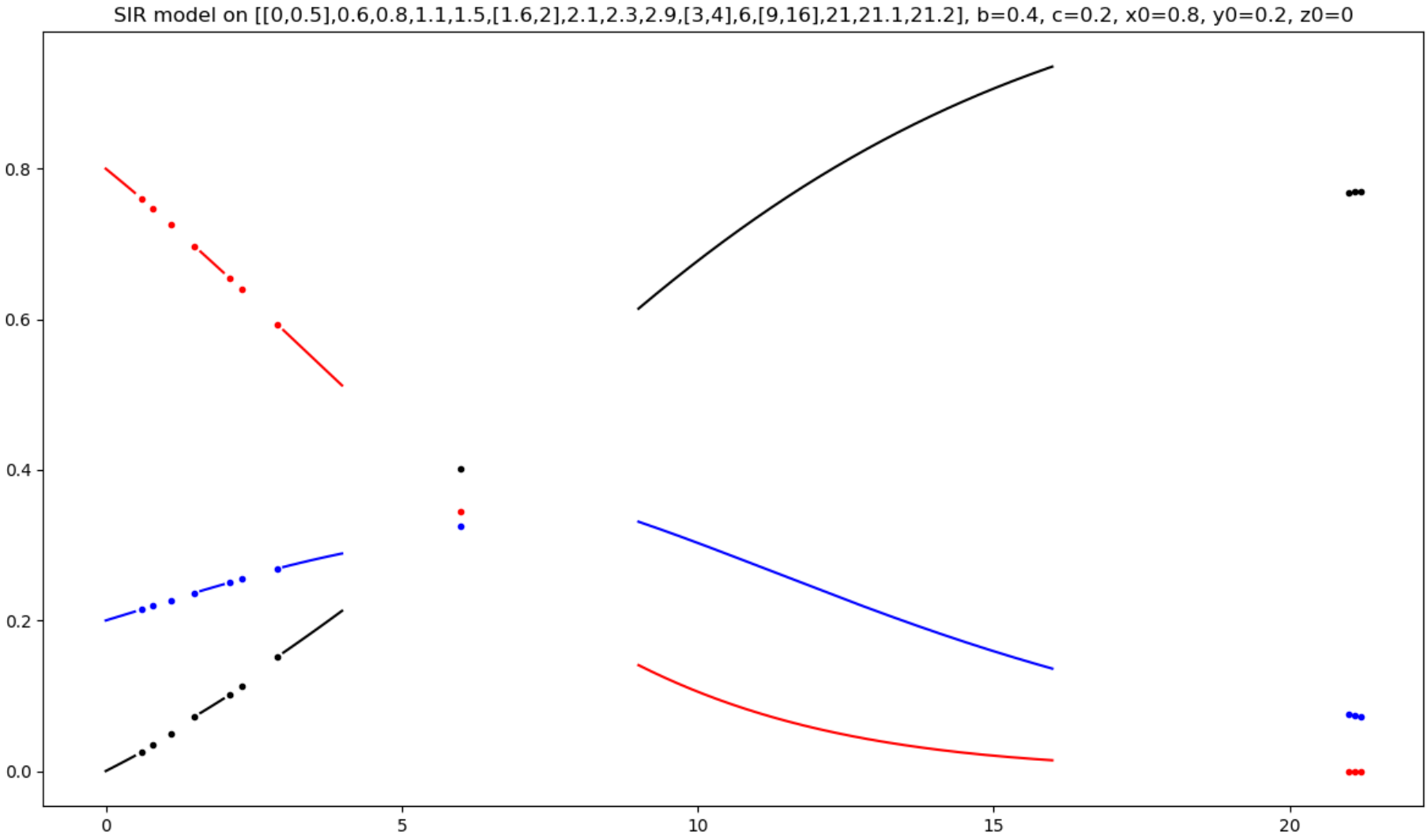
SIR model


SIR model on [[0,12],13,14,15,16,17,18,19,20,21,22,23,24], b=0.4, c=0.2, x0=0.8, y0=0.2, z0=0
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20

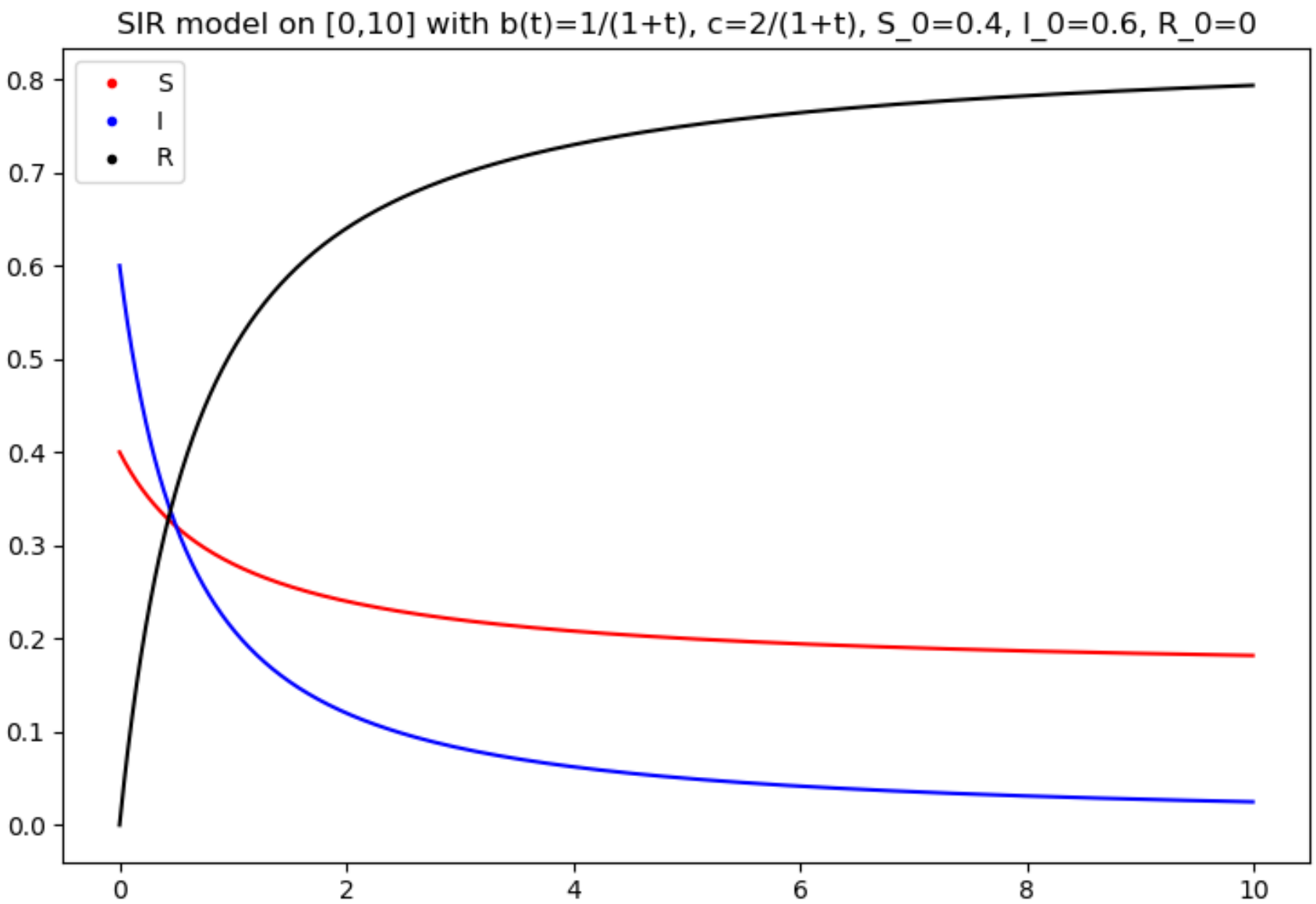
SIR model


SIR model on tsc.integers(0,24), b=0.4, c=0.2, x0=0.8, y0=0.2, z0=0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25

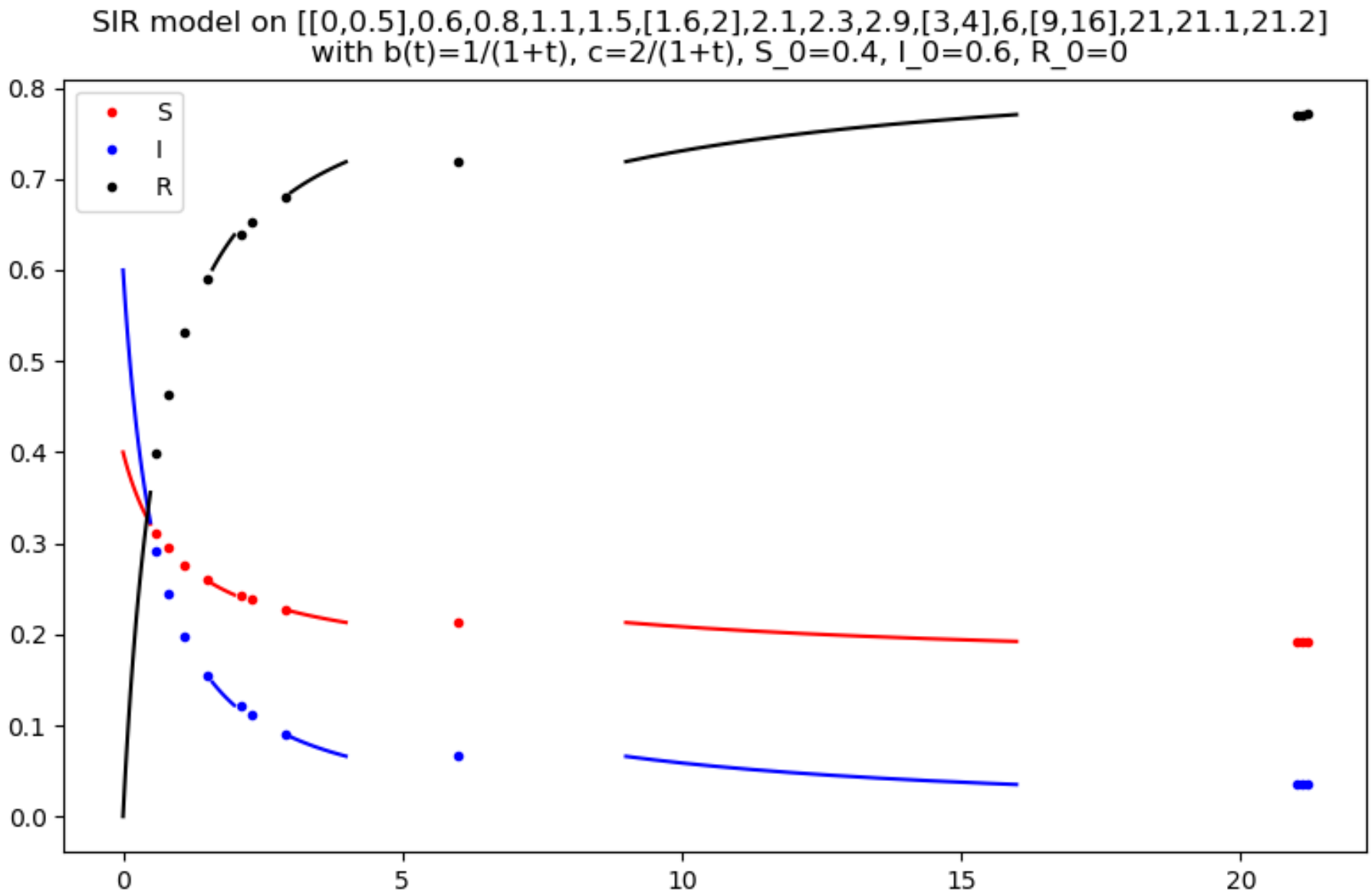
SIR model


SIR model on [[0,0.5],0.6,0.8,1.1,1.5,[1.6,2],2.1,2.3,2.9,[3,4],6,[9,16],21,21.1,21.2], b=0.4, c=0.2, x0=0.8, y0=0.2, z0=0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20

SIR model


SIR model on [0,10] with b(t)=1/(1+t), c=2/(1+t), S_0=0.4, I_0=0.6, R_0=0
S
I
R
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8
10

SIR model


SIR model on [[0,0.5],0.6,0.8,1.1,1.5,[1.6,2],2.1,2.3,2.9,[3,4],6,[9,16],21,21.1,21.2]
with b(t)=1/(1+t), c=2/(1+t), S_0=0.4, I_0=0.6, R_0=0
S
I
R
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
5
10
15
20

$y^{\Delta^4} = y^{\sigma\sigma}$

From [**?**, Section 4], consider

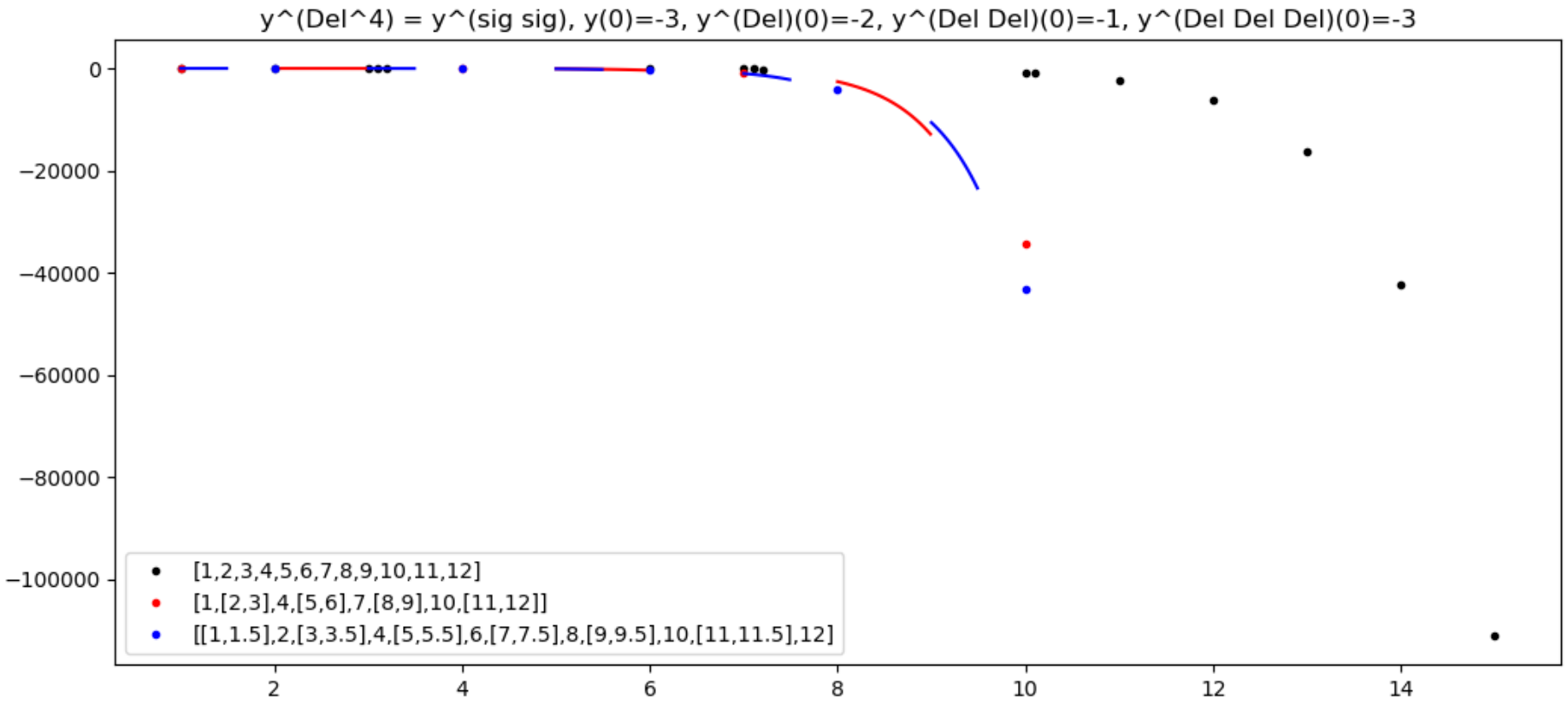
$$y^{\Delta^4} = y^{\sigma\sigma}.$$

Recall $y^{\sigma\sigma} = y^{\Delta\Delta} + 2y^{\Delta} - y$ and take $\begin{cases} y_1 = y \\ y_2 = y^{\Delta} \\ y_3 = y^{\Delta\Delta} \\ y_4 = y^{\Delta\Delta\Delta}. \end{cases}$ Consequently,

$$\begin{cases} y_1^{\Delta} = y_2 \\ y_2^{\Delta} = y_3 \\ y_3^{\Delta} = y_4 \\ y_4^{\Delta} = y_3 + 2y_2 + y_1. \end{cases}$$

```
def y_prime_vector(vector, t):
    y1, y2, y3, y4 = vector
    dt_vector=[y2, y3, y4, y3+2*y2+y1]
    return dt_vector
```

$y^{\Delta^4} = y^{\sigma\sigma}$



What's next?

1. currently working on systems of delay dynamic equations solver using `JiTCDDE`
2. more general time scales
3. backwards calculus, including $\rho, \nu, \hat{e}_p, \hat{\sin}, \hat{\cos}$, systems of dynamic equations solver, delay equations, etc etc
4. fractional calculus

Thank you

Dziękuję!

References