In [1]:

```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false
}
```

So this is a quick run through of all things Volterra and Generating series.

I've ran the generating series and serialised the outputs, it takes a fair while to run, around 35mins for $y_1 + y_2 + y_3 + y_4 + y_5$ with quadratic and cubic nonlinearities in symbolic form. I've currently got y_6 running but it's been going since ~7pm yesterday. The bottleneck in the code is the partial fractions decomposition, and I don't think there's much I can do about that?

All of the values used are the same as the ones in the paper "Analysis of Nonlinear System Response to an Impulse Excitation".

I'll load some of the serialised results in here so you can see what's going on :)

Comparison of Generating Series and Contour Integration

In [2]:

```
import numpy as np
import pickle as pkl
import dill
import matplotlib.pyplot as plt

# VCI is my shorthand for Volterra contour integration (aka the results from the paper).
from vci_quad_cube import y1 as y1_vci
from vci_quad_cube import y2 as y2_vci
from vci_quad_cube import y3 as y3_vci

# Runge-Kutta stuff.
from rk_quad_cube import duffing_equation
from scipy.integrate import solve_ivp

# Some parameters that are imported into all the files relating to this.
from params import plot
from params import iter_depth                              # Generating series param
from params import m, c, k1, k2, k3, init_cond, A_range  # Duffing params
from params import t_span, dt, t                           # Runge-kutta params

# iter_depth = 5 (number of Volterra terms to calculate)

# m = 1
# c = 20
# k1 = 1e4
# k2 = 1e7
# k3 = 5e9

# init_cond = (0, 0), Zero initial conditions.
# A_range = 0 to 0.15 in 0.01 steps.
# t_span = (0, 0.2)
# dt = 1e-4.
# t = 0 to 0.2 in steps of dt.
```

Creating a matrix for the Runge-Kutta solutions, time along rows, and amplitude along columns. I'll only use the seventh index ($0.07N$) in this first section, but I'll use them all in the error analysis in the second section.

In [3]:

```
y_rks = []
for A in A_range:
    sol = solve_ivp(
        duffing_equation, t_span, init_cond, method='RK45', t_eval=t,
        args=(m, c, k1, k2, k3, A)
    )
    y_rks.append(sol.y[0])

y_rks = np.vstack(y_rks).T
```

Load in the stored functions determined using the generating series method. These functions take two parameters, amplitude and time.

In [4]:

```
volterra_gen = []
for i in range(1, iter_depth+2):
    with open(f"quad_cube_y{i}_lambdify_A_t_gen.txt", "rb") as f_read:
        volterra_gen.append(dill.load(f_read))
```

I take the real part of the numpy array below, as previous numpy arrays in the calculation were complex. The values at this stage are completely real, just the array data type was preserved.

In [5]:

```
A = 0.07 # Fig. 8 in the paper.

y1_gen = volterra_gen[0](A, t).real
y2_gen = volterra_gen[1](A, t).real
y3_gen = volterra_gen[2](A, t).real
```

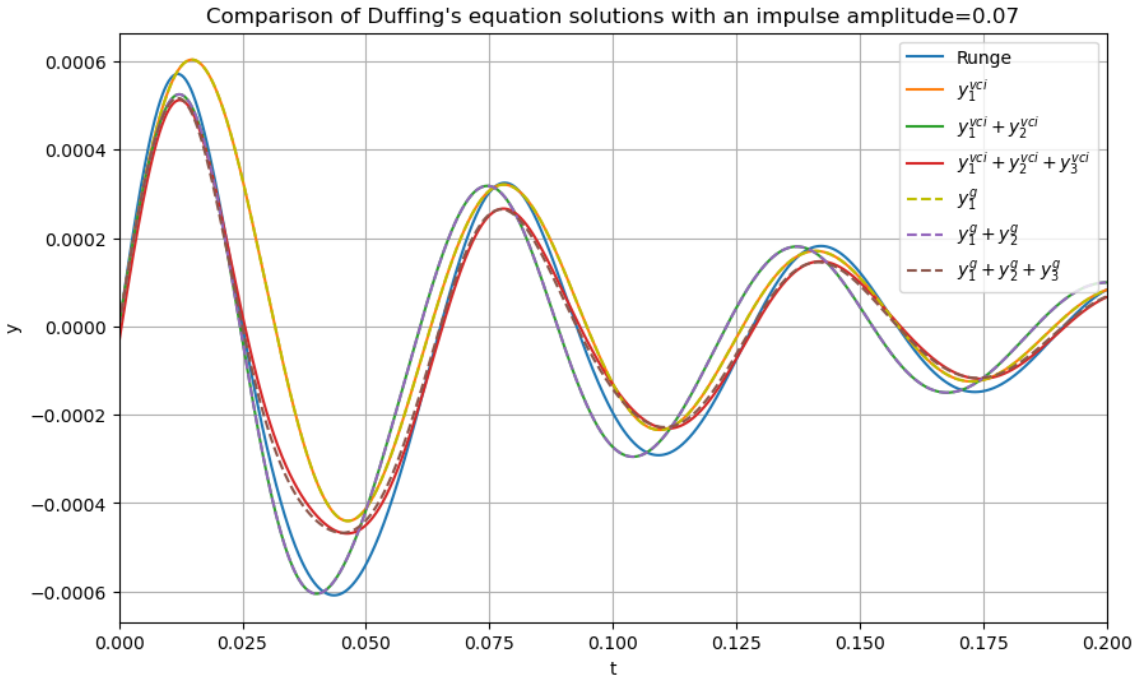
I'm going to overlay the Volterra terms deduced using countour integration and the generating series in the figure below. We *should* expect them to be the same.

In [6]:

```
figax = plot(y_rks[:, 7], None, "Runge")

figax = plot(y1_vci, figax, "$y_1^{vci}$")
figax = plot(y1_vci + y2_vci, figax, "$y_1^{vci} + y_2^{vci}$")
figax = plot(y1_vci + y2_vci + y3_vci, figax, "$y_1^{vci} + y_2^{vci} + y_3^{vci}$")

figax = plot(y1_gen, figax, "$y_1^g$", c="y", linestyle="--")
figax = plot(y1_gen + y2_gen, figax, "$y_1^g + y_2^g$", linestyle="--")
figax = plot(
    y1_gen + y2_gen + y3_gen, figax, "$y_1^g + y_2^g + y_3^g$", linestyle="--"
)
fig, ax = figax
ax.set_title(
    f"Comparison of Duffing's equation solutions with an impulse amplitude={A}"
)
```

Out[6]:

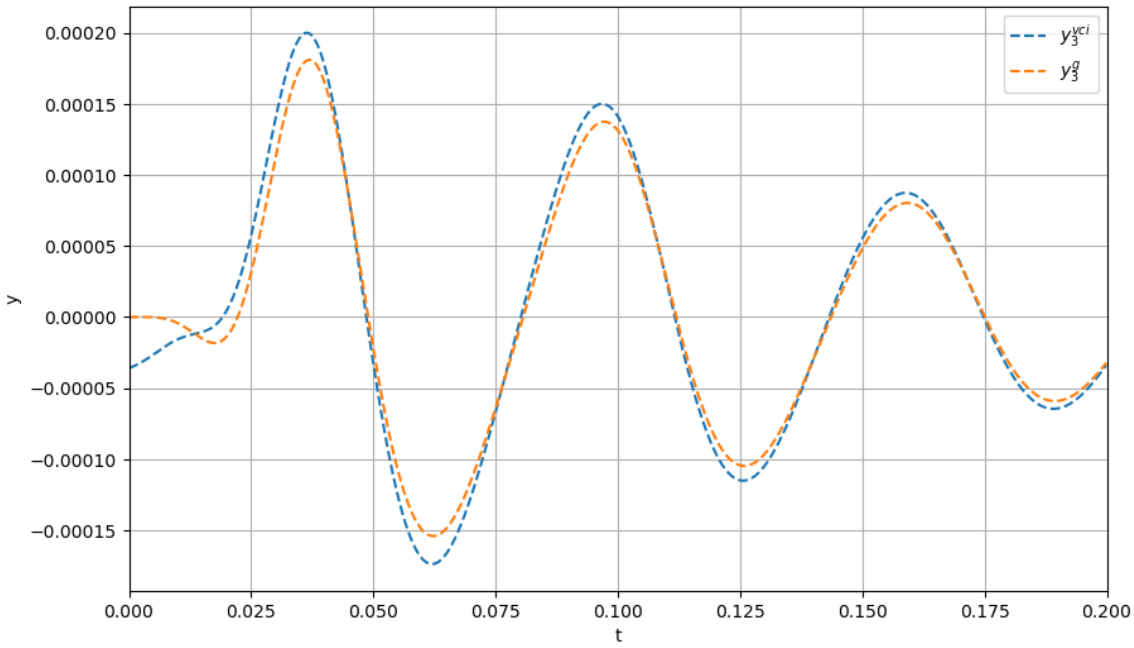
```
Text(0.5, 1.0, "Comparison of Duffing's equation solutions with an impulse
amplitude=0.07")
```



You can see here that y_1^v is the same as y_1^g and $y_1^v + y_2^v$ is the same as $y_1^g + y_2^g$ (I've equated these numerically and they're exact). However there seems to be a disagreement in the third term. I'll plot these terms by themselves below.

In [7]:

```
figax = plot(
    y3_vci, None, "$y_3^{vci}$", linestyle="--"
)
figax = plot(
    y3_gen, figax, "$y_3^g$", linestyle="--"
)
```



They're obviously different. I think there is an error with y_3^{vci}. It appears to be defying the zero initial conditions constraint.

There are two reasons that I can think for the contour integration and generating series expansions not being equivalent:

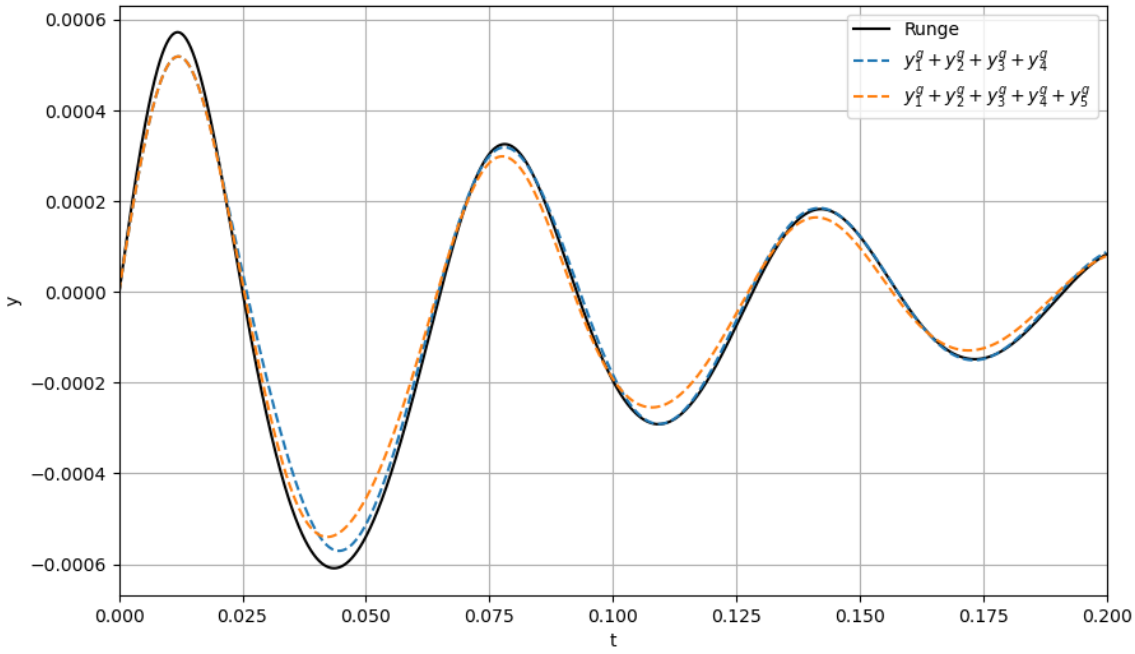
1. **Copying Error:** I used mathpix to convert the equation in the paper to LaTex and then I converted it to Python from LateX. I've also checked over my copied version for y_3 from the paper three times (once with Chris) and we're somewhat sure the version I've got in Python matches the one in the paper (I'm not ruling out the possibility of me being a massive idiot though). What I have typed up for y_3^{vci} doesn't align with the pictures in the paper, but my y_3^g does match. So I'm leaning towards there being a typo in $y_3^{k_2}$ in the paper (the contour integration and generating series $y_3^{k_3}$ are equivalent). If either of you have the derivation/ code used I'd be interested to see this.
2. **There is something wrong in my code** Hopefully not this, as I wouldn't know where to start!

Anyway, here are the next two terms that I've computed.

In [8]:

```
y4_gen = volterra_gen[3](A, t)
y5_gen = volterra_gen[4](A, t)

figax = plot(y_rks[:, 7], None, "Runge", c="k")
figax = plot(
    y1_gen + y2_gen + y3_gen + y4_gen, figax, "$y_1^g + y_2^g + y_3^g + y_4^g$", linestyl
)
figax = plot(
    y1_gen + y2_gen + y3_gen + y4_gen + y5_gen, figax, "$y_1^g + y_2^g + y_3^g + y_4^g +
)
```

```
C:\Users\trist\anaconda3\lib\site-packages\matplotlib\cbook\__init__.py:13
35: ComplexWarning: Casting complex values to real discards the imaginary
part
  return np.asarray(x, float)
```



Interestingly, the fourth order expansion looks to be a better fit than the fifth order? I assume y_5^g is diveregent at this impulse amplitude?

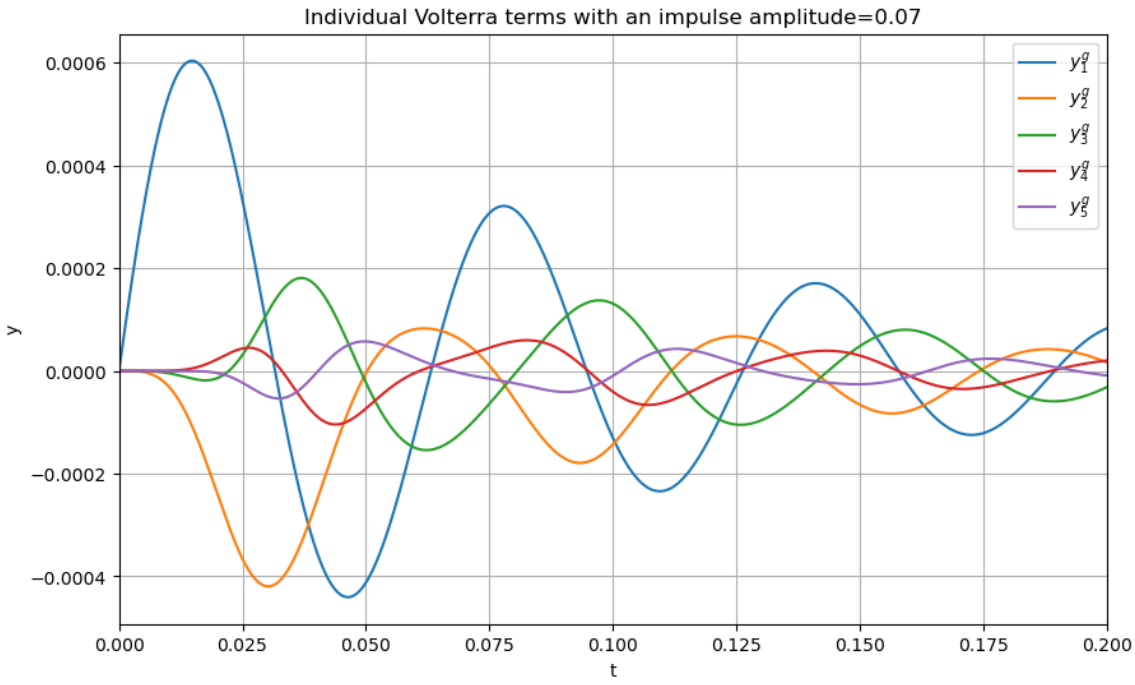
And here are all the terms by themselves.

In [9]:

```
figax = plot(y1_gen, None, "$y_1^g$")
figax = plot(y2_gen, figax, "$y_2^g$")
figax = plot(y3_gen, figax, "$y_3^g$")
figax = plot(y4_gen, figax, "$y_4^g$")
figax = plot(y5_gen, figax, "$y_5^g$")
fig, ax = figax
ax.set_title(
    f"Individual Volterra terms with an impulse amplitude={A}"
)
```

Out[9]:

```
Text(0.5, 1.0, 'Individual Volterra terms with an impulse amplitude=0.07')
```



Also, I'm going to display the symbolic representations of the terms that I've calculated as they're in a different from to the ones in the paper. The generating series ones are all in terms of exponentials.

In [10]:

```
symbolic_gen = []
for i in range(1, iter_depth+2):
    with open(f"quad_cube_y{i}_gen.txt", "rb") as f_read:
        symbolic_gen.append(pkl.load(f_read))
```

In the terms below, a_1 and a_2 are the roots to the quadratic $mx^2 + cx + k_1 x = 0$.

In [11]:

```
symbolic_gen[0]
```

Out[11]:

$$\frac{Ae^{-a_2 t}}{a_1 - a_2} - \frac{Ae^{-a_1 t}}{a_1 - a_2}$$

One thing I've not tested yet is for symmetry in a_1 and a_2. The linear term looks symmetric in a_1 and a_2 to me. This feels like something that should be explicitly addressed, the generating series papers don't mention anything. I wouldn't know how to go about this for the higher order terms, other than a crude numerical example like this:

In [12]:

```
from sympy import symbols
_a1, _a2 = symbols("a1 a2")
symbolic_gen[0].subs({_a1: 1, _a2: 10})
```

Out[12]:

$$\frac{Ae^{-t}}{9} - \frac{Ae^{-10t}}{9}$$

In [13]:

```
symbolic_gen[0].subs({_a1: 10, _a2: 1})
```

Out[13]:

$$\frac{Ae^{-t}}{9} - \frac{Ae^{-10t}}{9}$$

And the remaining terms:

In [14]:

```
symbolic_gen[1]
```

Out[14]:

$$\frac{A^2k_2e^{-2a_2t}}{a_1^3a_2 - 4a_1^2a_2^2 + 5a_1a_2^3 - 2a_2^4} + \frac{A^2k_2e^{-2a_1t}}{-2a_1^4 + 5a_1^3a_2 - 4a_1^2a_2^2 + a_1a_2^3} - \frac{2A^2k_2e^{-a_2t}}{2a_1^3a_2 - 3a_1^2a_2^2 + a}$$

In [15]:

```
symbolic_gen[2]
```

Out[15]:

$$-\frac{A^3k_2^2e^{-3a_2t}}{2a_1^6a_2-19a_1^5a_2^2+72a_1^4a_2^3-140a_1^3a_2^4+148a_1^2a_2^5-81a_1a_2^6+18a_2^7}-\frac{2a_1^6a_2-21a_1^5}{}$$
$$-\frac{4A^3k_2^2e^{-2a_1t}}{-4a_1^6a_2+24a_1^5a_2^2-55a_1^4a_2^3+60a_1^3a_2^4-31a_1^2a_2^5+6a_1a_2^6}+\frac{3A^3k_2^2e^{t}}{a_1^5a_2^2-4a_1^4a_2^3+4a_1^3a_2^4}$$
$$+\frac{A^3k_2^2e^{-3a_2t}}{a_1^5a_2^2-9a_1^4a_2^3+31a_1^3a_2^4-51a_1^2a_2^5+40a_1a_2^6-12a_2^7}+\frac{A^3k_2^2e^{-}}{-6a_1^7+23a_1^6a_2-34a_1^5a_2^2+}$$
$$-\frac{12A^3k_2^2e^{-2a_1t}}{4a_1^6a_2-20a_1^5a_2^2+35a_1^4a_2^3-25a_1^3a_2^4+6a_1^2a_2^5}-\frac{3A^3k_2^2e^{-a_2t}}{-6a_1^6a_2+5a_1^5a_2^2+5a_1^4a_2^3-5a_1^3a_2^4}$$
$$-\frac{2A^3k_2^2e^{t(-a_1-2a_2)}}{a_1^4a_2^3-3a_1^3a_2^4+a_1^2a_2^5+3a_1a_2^6-2a_2^7}-\frac{2A^3k_2^2e^{-a_1t}}{a_1^4a_2^3-5a_1^3a_2^4+5a_1^2a_2^5+5a_1a_2^6-6a_2^7}+\frac{}{-}$$
$$+\frac{A^3k_2^2e^{-a_2t}}{2a_1^6a_2-a_1^5a_2^2-2a_1^4a_2^3+a_1^3a_2^4}-\frac{A^3k_2^2e^{t(-2a_1-a_2)}}{a_1^6a_2-a_1^5a_2^2-a_1^4a_2^3+a_1^3a_2^4}+\frac{3A^3k_2^2e^{t(-2a}}{a_1^6a_2-2a_1^5a_2^2+2a}$$
$$+\frac{12A^3k_2^2e^{t(-a_1-a_2)}}{-2a_1^5a_2^2+7a_1^4a_2^3-7a_1^3a_2^4+2a_1^2a_2^5}+\frac{2A^3k_2^2e^{t(-a_1-2a_2)}}{a_1^4a_2^3-a_1^3a_2^4-a_1^2a_2^5+a_1a_2^6}+\frac{2A^3k}{a_1^4a_2^3-3a_1^3a_2^4}$$
$$+\frac{A^3k_3e^{-3a_2t}}{2a_1^4a_2-12a_1^3a_2^2+24a_1^2a_2^3-20a_1a_2^4+6a_2^5}+\frac{A^3k_3e^{-3a_1t}}{6a_1^5-20a_1^4a_2+24a_1^3a_2^2-12a_1^2a_2^3+}$$
$$+\frac{3A^3k_3e^{t(-a_1-2a_2)}}{2a_1^4a_2-4a_1^3a_2^2+4a_1a_2^4-2a_2^5}-\frac{3A^3k_3e^{-a_1t}}{2a_1^4a_2-6a_1^3a_2^2-2a_1^2a_2^3+6a_1a_2^4}+\frac{3A^3k_}{-2a_1^5+4a_1^4a_2}$$
$$+\frac{\left(-4A^3a_1^2k_2^2+8A^3a_1a_2k_2^2-12A^3a_2^2k_2^2\right)e^{t(-a_1-a_2)}}{a_1^2a_2^2\left(a_1+a_2\right)\left(a_1^4-6a_1^3a_2+13a_1^2a_2^2-12a_1a_2^3+4a_2^4\right)}+\frac{\left(12A^3a_1^2k_2^2-16A^3a_1\right.}{a_1^2a_2^2\left(a_1+a_2\right)\left(a_1^4-6a_1^3\right.}$$
$$+\frac{\begin{array}{l}(t(-a_1-a_2)+1)\left(-4A^3a_1^2a_2^2k_2^2\left(a_1+a_2\right)\left(a_1^4-6a_1^3a_2+13a_1^2a_2^2-12a_1a_2^3+4a_2^4\right)\right.\\ \left.-a_1^2a_2\left(-4A^3a_1^2k_2^2+8A^3a_1a_2k_2^2-12A^3a_2^2k_2^2\right)\left(a_1^4-6a_1^3a_2+13a_1^2a_2^2-12a_1a_2^3+\right.\right.\end{array}}{a_1^4a_2^3\left(a_1+a_2\right)\left(a_1^4-6a_1^3a_2+13a_1^2a_2^2-12a_1a_2^3+4a_2^4\right)^2}$$
$$+\frac{\begin{array}{l}(t(-a_1-a_2)+1)\left(a_1^2a_2^2\left(a_1+a_2\right)\left(8A^3a_1k_2^2-12A^3a_2k_2^2\right)\left(a_1^4-6a_1^3a_2+13a_1^2a_2^2-\right.\right.\\ \left.\left.+a_1^2a_2^2\left(-12A^3a_1^2k_2^2+16A^3a_1a_2k_2^2+4A^3a_2^2k_2^2\right)\left(a_1^4-6a_1^3a_2+13a_1^2a_2^2-12a_1a_2^3+\right.\right.\end{array}}{a_1^4a_2^4\left(a_1+a_2\right)\left(a_1^4-6a_1^3a_2+13a_1^2a_2^2-12a_1a_2^3+4a_2^4\right)^2}$$

In [16]:

```
# These are a bit long, so I've commented them out.
# symbolic_gen[3] # I displayed this out of interest as it's the first mixing term of k2
# symbolic_gen[4] # I've not even bothered displaying this.
```

With some mathematical jiggery-pokery I'd assume you'd be able to manipulate the contour integration and generating series forms into one another.

Error analysis with Runge-Kutta

From here on out, I will only consider the terms from the generating series method.

In [17]:

```
def rrse(pred, true):
    """
    Relative root squared error along each column.
    """
    numer = np.square(np.subtract(        pred, true)).sum(0)
    denom = np.square(np.subtract(true.mean(0), true)).sum(0)

    return 100 * np.sqrt(np.divide(numer, denom))

# Creating array of values, time along rows, amplitude along columns.
all_funcs = []
for func in volterra_gen:
    all_amps = []
    for amp in A_range:
        all_amps.append(func(amp, t).real)
    all_funcs.append(np.vstack(all_amps).T)

ys_summed = []
rrses = []
for i, times in enumerate(all_funcs):
    if i == 0:
        ys_summed.append(times)
    else:
        ys_summed.append(ys_summed[-1] + times)
        # assert (ys_summed[-1] == (ys_summed[-2] + times)).all()
    rrses.append(rrse(y_rks, ys_summed[-1]))
```

```
C:\Users\trist\AppData\Local\Temp\ipykernel_12840\2806243351.py:8: Runtime
Warning: invalid value encountered in divide
  return 100 * np.sqrt(np.divide(numer, denom))
```

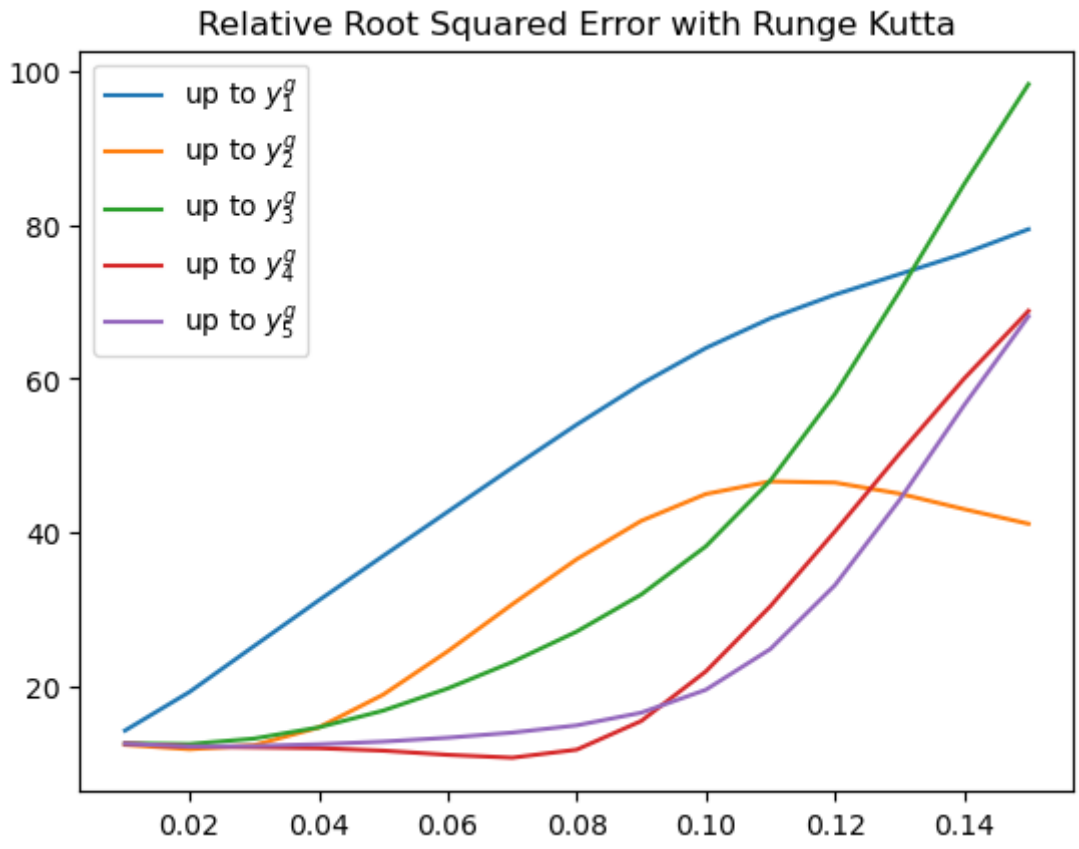
Divide by zero warning because I've included $A = 0$ and $t = 0$.

In [18]:

```
fig_error = plt.figure()
ax_error = fig_error.gca()
for i, root_rel_err in enumerate(rrses, 1):
    ax_error.plot(A_range, root_rel_err, label=f"up to $y^g_{i}$")
ax_error.legend()
ax_error.set_title(
    "Relative Root Squared Error with Runge Kutta"
)
```

Out[18]:

```
Text(0.5, 1.0, 'Relative Root Squared Error with Runge Kutta')
```

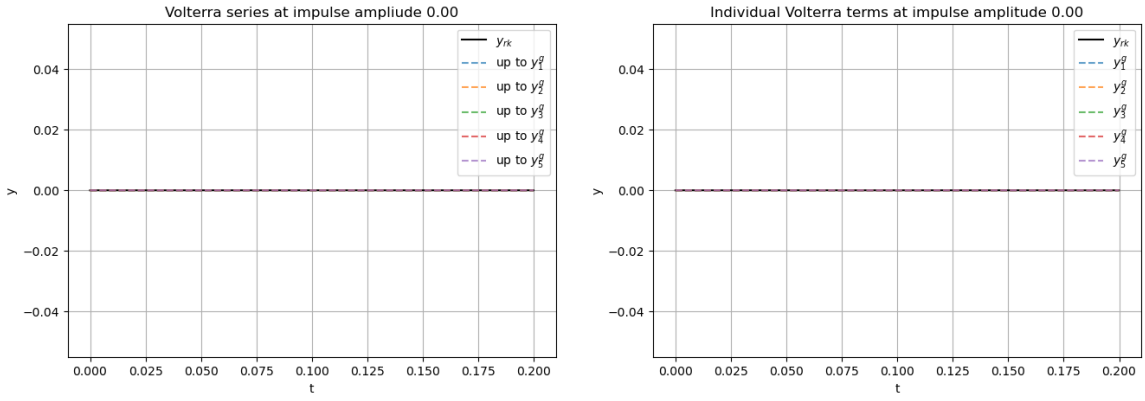


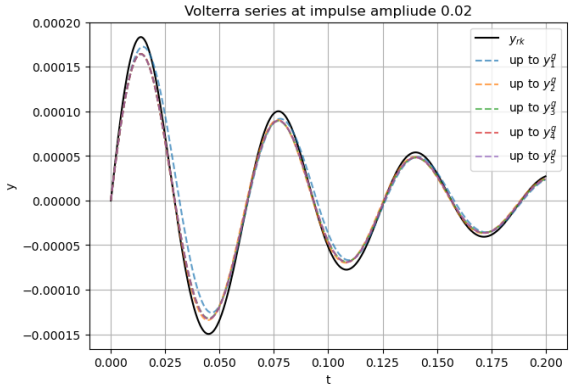
This seems to be a fair bit different to the plot in the paper, I'd I'm not entirely sure why?

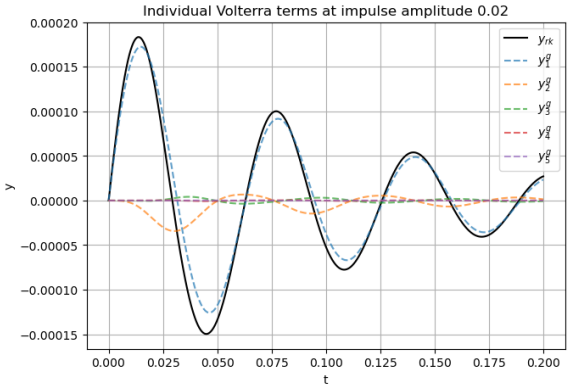
And then finally, I thought I'd list the responses at all the amplitudes.

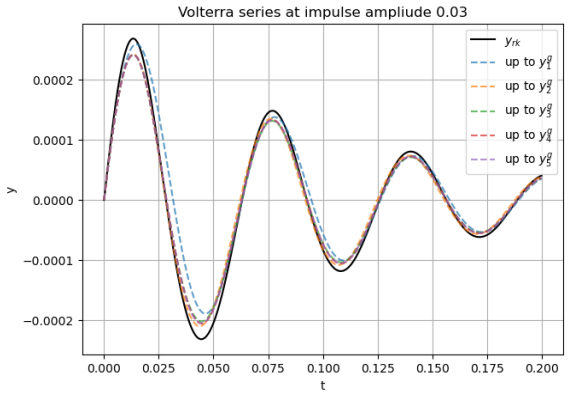
In [19]:

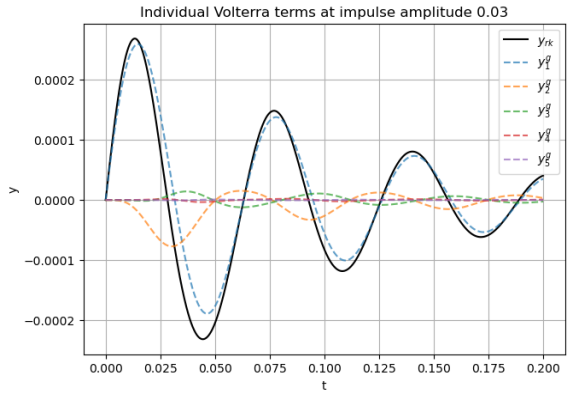
```
for amp_index in range(len(A_range)):
    fig, axs = plt.subplots(1, 2, figsize=(16, 5))

    axs[0].plot(t, y_rks[:, amp_index], label="$y_{rk}$", c="k")
    axs[1].plot(t, y_rks[:, amp_index], label="$y_{rk}$", c="k")

    for i, (y_s, y_i) in enumerate(zip(ys_summed, all_funcs), 1):
        axs[0].plot(
            t, y_s[:, amp_index], label=f"up to $y_{i}^g$", linestyle="--",
            alpha=0.7
        )
        axs[1].plot(
            t, y_i[:, amp_index], label=f"$y_{i}^g$", linestyle="--", alpha=0.7
        )
    axs[0].legend(loc=1, prop={"size": 10})
    axs[1].legend(loc=1, prop={"size": 10})
    axs[0].grid(True)
    axs[1].grid(True)
    axs[0].set_xlabel('t')
    axs[0].set_ylabel('y')
    axs[1].set_xlabel('t')
    axs[1].set_ylabel('y')
    axs[0].set_title(
        f"Volterra series at impulse ampliude {A_range[amp_index]:.2f}"
    )
    axs[1].set_title(
        f"Individual Volterra terms at impulse amplitude {A_range[amp_index]:.2f}"
    )
```

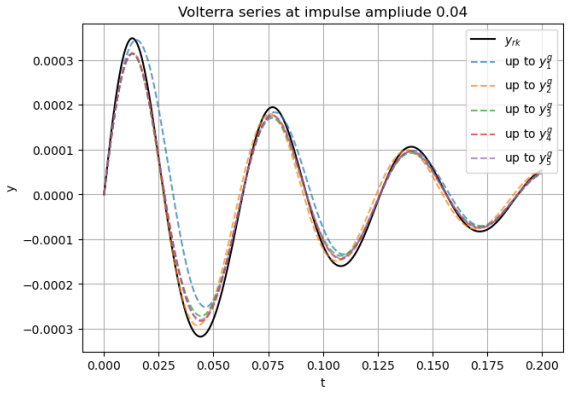


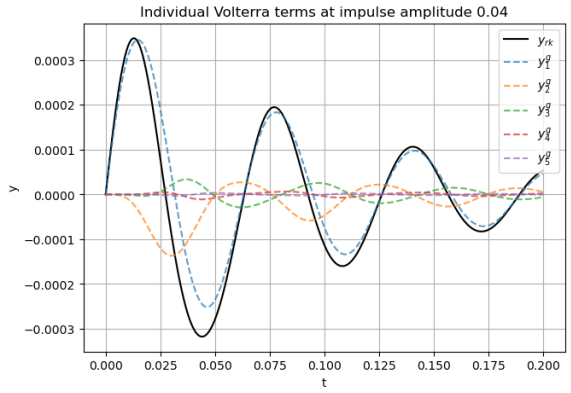

Volterra series at impulse ampliude 0.02
y
t
y_{rk}
up to y_1^g
up to y_2^g
up to y_3^g
up to y_4^g
up to y_5^g

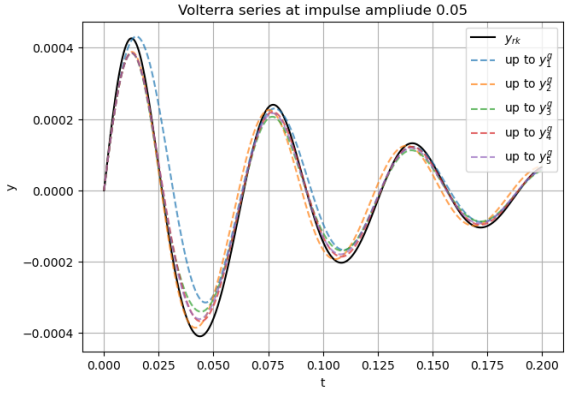

Individual Volterra terms at impulse amplitude 0.02
y
t
y_{rk}
y_1^g
y_2^g
y_3^g
y_4^g
y_5^g

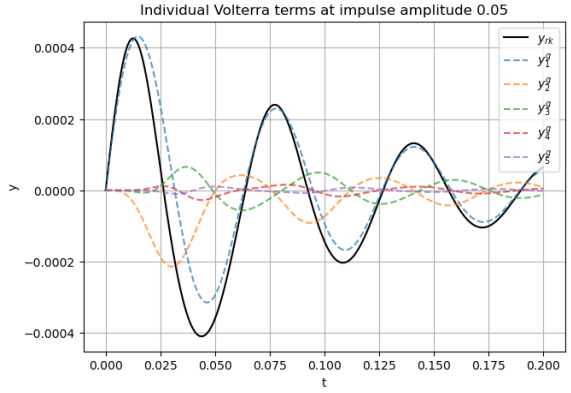

Volterra series at impulse ampliude 0.03
y
t
y_{rk}
up to y_1^g
up to y_2^g
up to y_3^g
up to y_4^g
up to y_5^g

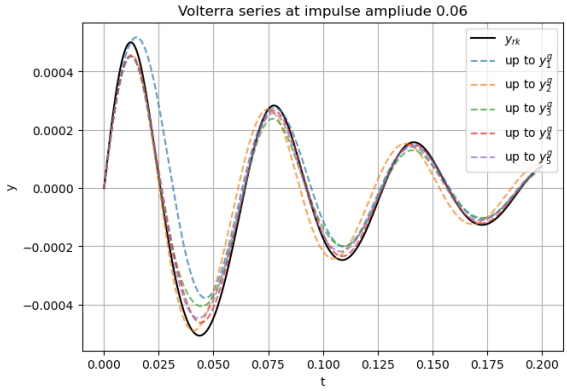

Individual Volterra terms at impulse amplitude 0.03
y
t
y_{rk}
y_1^g
y_2^g
y_3^g
y_4^g
y_5^g

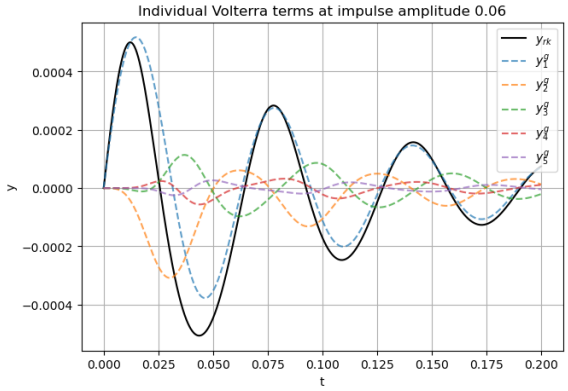

Volterra series at impulse ampliude 0.04
y
t
y_{rk}
up to y_1^g
up to y_2^g
up to y_3^g
up to y_4^g
up to y_5^g

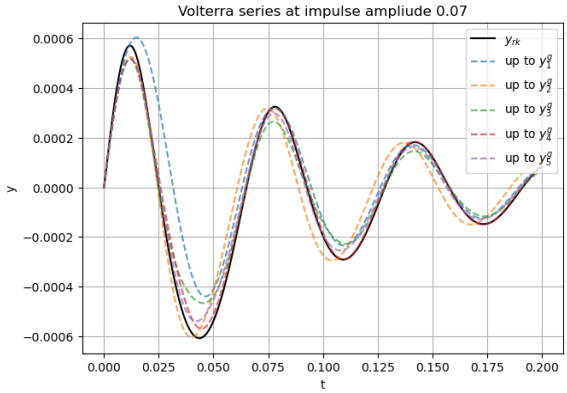

Individual Volterra terms at impulse amplitude 0.04
y
t
y_{rk}
y_1^g
y_2^g
y_3^g
y_4^g
y_5^g

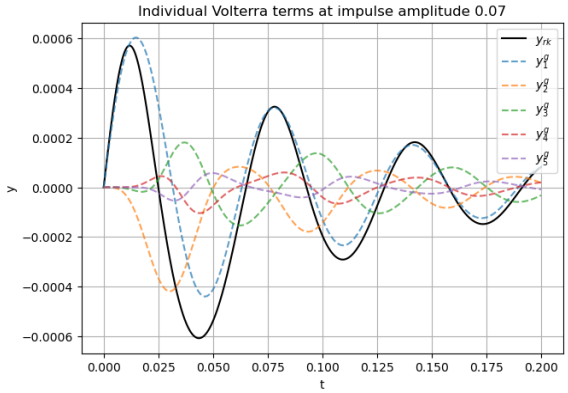

Volterra series at impulse ampliude 0.05
y
t
y_{rk}
up to y_1^g
up to y_2^g
up to y_3^g
up to y_4^g
up to y_5^g

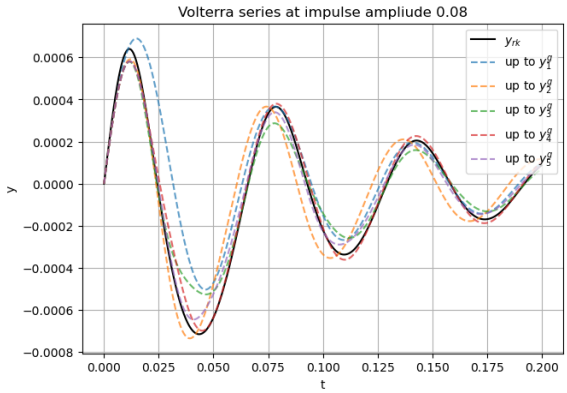

Individual Volterra terms at impulse amplitude 0.05
y
t
y_{rk}
y_1^g
y_2^g
y_3^g
y_4^g
y_5^g

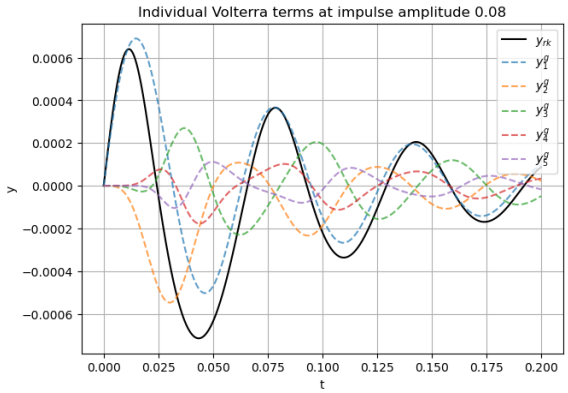

Volterra series at impulse ampliude 0.06
y_{rk}
up to y_1^g
up to y_2^g
up to y_3^g
up to y_4^g
up to y_5^g
y
t

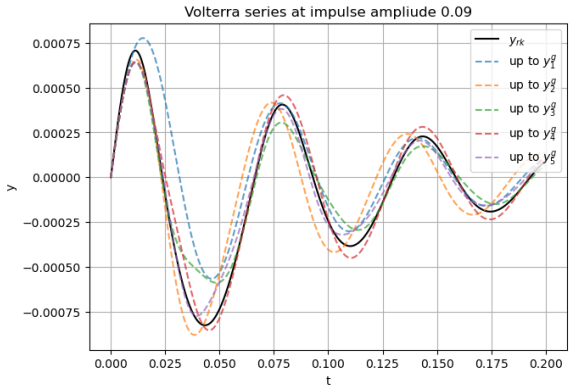

Individual Volterra terms at impulse amplitude 0.06
y_{rk}
y_1^g
y_2^g
y_3^g
y_4^g
y_5^g
y
t

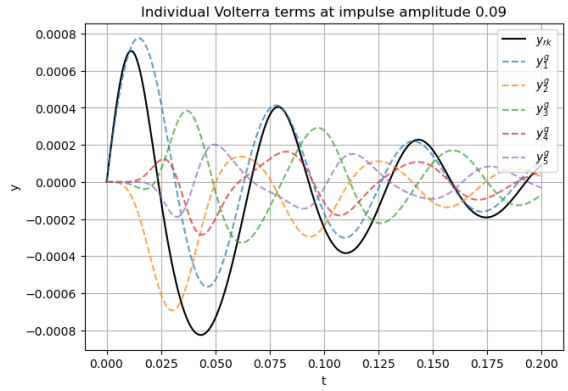

Volterra series at impulse ampliude 0.07
y_{rk}
up to y_1^g
up to y_2^g
up to y_3^g
up to y_4^g
up to y_5^g
y
t

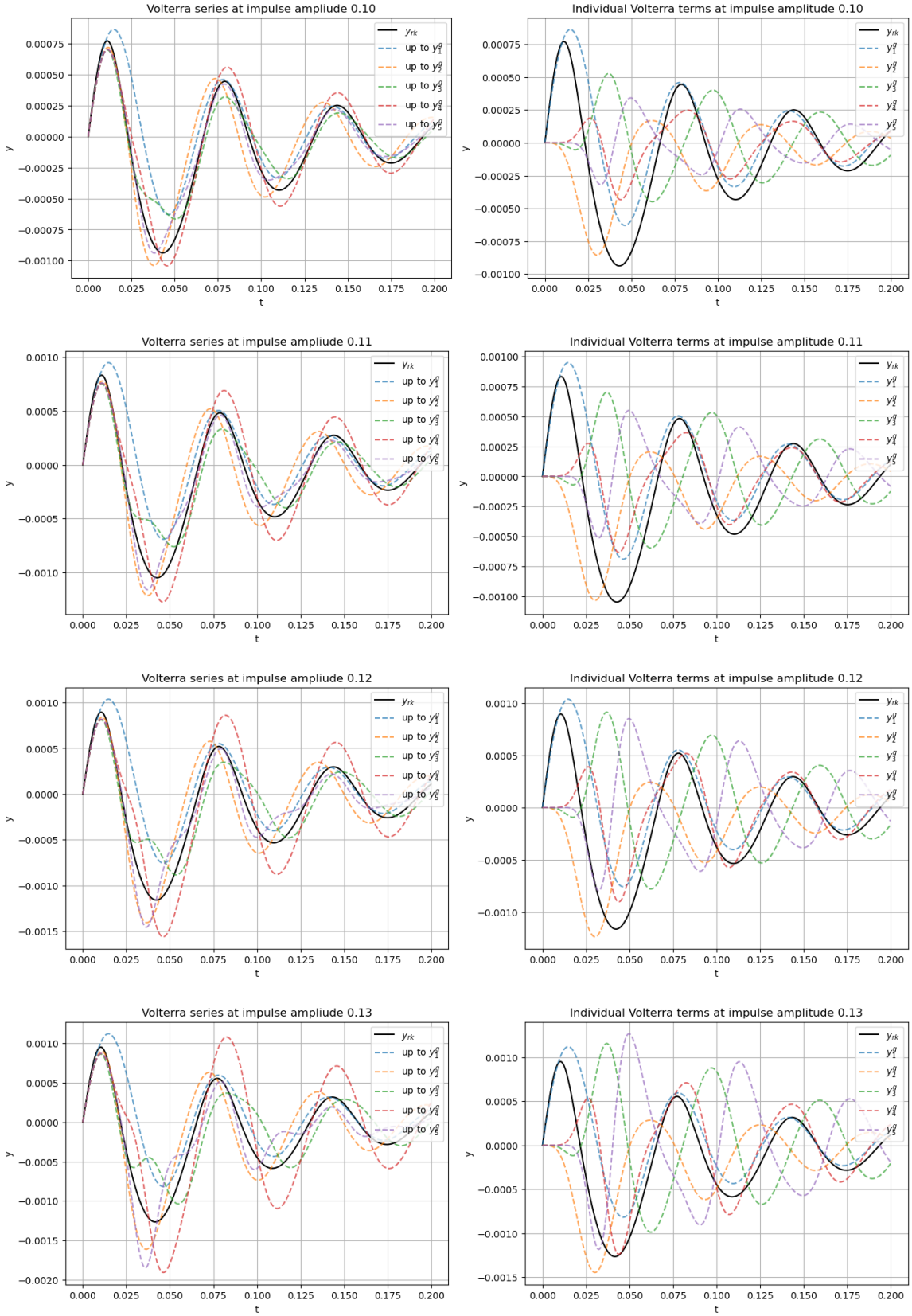

Individual Volterra terms at impulse amplitude 0.07
y_{rk}
y_1^g
y_2^g
y_3^g
y_4^g
y_5^g
y
t


Volterra series at impulse ampliude 0.08
y_{rk}
up to y_1^g
up to y_2^g
up to y_3^g
up to y_4^g
up to y_5^g
y
t


Individual Volterra terms at impulse amplitude 0.08
y_{rk}
y_1^g
y_2^g
y_3^g
y_4^g
y_5^g
y
t


Volterra series at impulse ampliude 0.09
y_{rk}
up to y_1^g
up to y_2^g
up to y_3^g
up to y_4^g
up to y_5^g
y
t

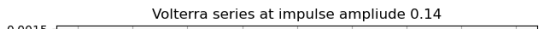

Individual Volterra terms at impulse amplitude 0.09
y_{rk}
y_1^g
y_2^g
y_3^g
y_4^g
y_5^g
y
t

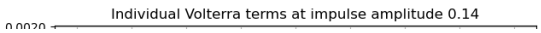



```
# Need this here to render in figure above
```

In [ ]:

```

```


Volterra series at impulse ampliude 0.14


Individual Volterra terms at impulse amplitude 0.14