UMEÅ UNIVERSITY
Department of Computing Science
Assignment report, v.2

January 22, 2019

5DV005 - Project 3

Scientific Computing
Autumn 2018, 7.5 Credits

Error estimation for artillery computations

Name Betty Törnkvist (`et16btt@cs.umu.se`)
Jonas Sjödin (`id16jsn@cs.umu.se`)
Emil Söderlind (`id15esd@cs.umu.se`)

Path ~`et16btt/edu/tvb/5dv005ht18/`

Teacher
Carl Christian Kjelgaard Mikkelsen

Contents

1 Introduction

After approximating the range of a gun, the flight time of a shell or the elevation necessary to hit a particular target, the error estimates can be computed. In this assignment we set out to approximate these error estimates in a reliable and accurate way.

We consider each approximation A as a function of the size of the time step h used when computing the trajectories, i.e., $A = Ah$. We then investigate if there exist any asymptotic error expansions of the form,

$$T - Ah = \alpha h^p + \beta h^q + O(h^r), \quad 0 < p < q < r \tag{1}$$

where the T is the target value and h the step size. The primary error term is represented by αh^p and the secondary error term by βh^q. From this asymptotic error expansion we obtain the difference between the target value T and the approximation Ah. Computing Ah will at best give us the floating point representation $\hat{A}h$, and the difference between Ah and $\hat{A}h$ is affected by many round off-errors. To determine when the round off-error $Ah - \hat{A}h$ is irrelevant compared with the error $T - Ah$ we use Richardson's extrapolation method.

2 Implementing Richardson's extrapolation

Richardson's extrapolation method is in this case an iterative method which is used to improve the result of a solution of a system of first order differential equations. In this task we write a function called `MyRichardson` which is an implementation of this method. To test if our implementation is correct we use a minimal working example file called `MyRichardson_MWE.m` and compares its out data to given out data (Figure 1), and can therefore conclude that our implementation is working as expected.

```
 k |   Approximation A_h | Fraction F_h |   Error estimate E_h
 1 |  2.895480163672e+00 |   0.00000000 |   0.000000000000e+00
 2 |  2.805025851403e+00 |   0.00000000 |  -9.045431226844e-02
 3 |  2.761200888902e+00 |   2.06399064 |  -4.382496250165e-02
 4 |  2.739629445828e+00 |   2.03161941 |  -2.157144307422e-02
 5 |  2.728927822736e+00 |   2.01571695 |  -1.070162309156e-02
 6 |  2.723597892360e+00 |   2.00783544 |  -5.329930376490e-03
 7 |  2.720938129638e+00 |   2.00391198 |  -2.659762721350e-03
 8 |  2.719609546672e+00 |   2.00195456 |  -1.328582965698e-03
 9 |  2.718945579511e+00 |   2.00097692 |  -6.639671619268e-04
10 |  2.718613676976e+00 |   2.00048837 |  -3.319025345263e-04
11 |  2.718447745963e+00 |   2.00024413 |  -1.659310128161e-04
12 |  2.718364785520e+00 |   2.00012206 |  -8.296044325107e-05
13 |  2.718323306559e+00 |   2.00006078 |  -4.147896106588e-05
14 |  2.718302567373e+00 |   2.00002842 |  -2.073918585666e-05
15 |  2.718292197853e+00 |   2.00001403 |  -1.036952016875e-05
16 |  2.718287013122e+00 |   2.00001123 |  -5.184730980545e-06
17 |  2.718284420669e+00 |   1.99993264 |  -2.592452801764e-06
18 |  2.718283124268e+00 |   1.99973060 |  -1.296401023865e-06
19 |  2.718282476068e+00 |   2.00000000 |  -6.482005119324e-07
20 |  2.718282151967e+00 |   2.00000000 |  -3.241002559662e-07
```

Figure 1: The expected output of `a3f2.m`

3 Range of shot with `a3f3` parameters

We can now use our implementation of Richardson's extrapolation method to compute the range of the shot given by `a3f3.m`. To do this we use the given trajectory methods, `rk1`, `rk2`, `rk3` and `rk4`. We use time steps $h_k = 2^{-k}$ seconds for $k = 0, 1, 2, ...$ We also use the function range rkx to move the shell from point to point using a time step which is fixed, except for the very last time step. Here a non-linear solver is used to compute the time step which will place the shell on the ground. This equation is solved to the limit of machine precision, specifically the tolerance passed to the underlying bisection routine is tol $= 2^-53$.

3.1 `Runge-Kutta 1`

We know that Richardson's fraction F_h, converges to 2^p. Therefore we can conclude by looking at Figure 2 that in this case $p = 1$. From Figure 6a we can calculate the slope by using $s = \frac{dy}{dx}$ which in this case is the following:

$$s = \frac{dy}{dx} = \frac{(-16.17) - (-3.003)}{16 - 3} = \frac{-13.167}{13} = -1.013 \approx -1 \tag{2}$$

Since we know that $s = -(q - p)$ we can now calculate our secondary error term q.

$$q = p - s = 1 - (-1) = 2 \tag{3}$$

Since $p = 1$ and $2^p = 2$ we can see in Figure 6a that it converges with the same slope until k=16, i.e. it roughly divides itself by 2 converging to 2. We can therefore say that we can trust our slope in $k = \{3..16\}$

Our best approximation of the range is row 16 in Figure 2.

k	Approximation A_h	Fraction F_h	Error estimate E_h
1	2.215475800698e+04	0.00000000	0.000000000000e+00
2	2.226834769726e+04	0.00000000	1.135896902761e+02
3	2.232180737277e+04	2.12477328	5.345967551322e+01
4	2.234768231668e+04	2.06607890	2.587494390464e+01
5	2.236039746277e+04	2.03497024	1.271514609780e+01
6	2.236670811603e+04	2.01487003	6.310653262091e+00
7	2.236984977888e+04	2.00869844	3.141662846803e+00
8	2.237141767501e+04	2.00374425	1.567896123950e+00
9	2.237220079351e+04	2.00211861	7.831185015530e−01
10	2.237259215891e+04	2.00099062	3.913654031312e−01
11	2.237278779330e+04	2.00049388	1.956343916754e−01
12	2.237288559799e+04	2.00025573	9.780469011821e−02
13	2.237293449721e+04	2.00012783	4.889921965878e−02
14	2.237295894604e+04	2.00006392	2.444882848795e−02
15	2.237297117025e+04	2.00003391	1.222420699560e−02
16	2.237297728231e+04	2.00001352	6.112062183092e−03
17	2.237298033832e+04	2.00001127	3.056013869355e−03

Figure 2: Output of Richardson's fraction RK1.

3.2 `Runge-Kutta 2`

We know that Richardson's fraction F_h, converges to 2^p. Therefore we can conclude by looking at Figure ?? that in this case $p = 2$. From Figure 6b we can calculate the slope by using $s = \frac{dy}{dx}$

which in this case is the following:

$$s = \frac{dy}{dx} = \frac{(-8.233) - (-0.3177)}{11 - 3} = \frac{-7.9263}{8} = -0.991 \approx -1 \tag{4}$$

Since we know that $s = -(q - p)$ we can now calculate our secondary error term q.

$$q = p - s = 2 - (-1) = 3 \tag{5}$$

Since $p = 2$ and $2^p = 4$ we can see in Figure 6b that it converges with the same slope until k=11, i.e. it roughly divides itself by 2 converging to 4. We can therefore say that we can trust our slope in $k = \{3..11\}$

Our best approximation of the range is row 11 in Figure 3.

k	Approximation A_h	Fraction F_h	Error estimate E_h
1	2.236937113617e+04	0.00000000	0.000000000000e+00
2	2.237221489000e+04	0.00000000	9.479179419698e−01
3	2.237280705137e+04	4.80232914	1.973871250727e−01
4	2.237294120066e+04	4.41419684	4.471643019982e−02
5	2.237297307937e+04	4.20811609	1.062623493393e−02
6	2.237298084435e+04	4.10544520	2.588327064586e−03
7	2.237298276044e+04	4.05251784	6.386960327897e−04
8	2.237298323630e+04	4.02651976	1.586223515915e−04
9	2.237298335488e+04	4.01313863	3.952575934818e−05
10	2.237298338448e+04	4.00668004	9.864965250017e−06
11	2.237298339187e+04	4.00332422	2.464193433601e−06
12	2.237298339372e+04	3.99980513	6.160783717254e−07
13	2.237298339418e+04	4.00655352	1.537676628989e−07
14	2.237298339429e+04	4.02712230	3.818301289963e−08
15	2.237298339432e+04	3.95664740	9.650345115612e−09

Figure 3: Output of Richardson's fraction RK2.

3.3 Runge-Kutta 3

We know that Richardson's fraction F_h, converges to 2^p. Therefore we can conclude by looking at Figure 4 that in this case $p = 3$. From Figure 6c we can calculate the slope by using $s = \frac{dy}{dx}$ which in this case is the following:

$$s = \frac{dy}{dx} = \frac{(-6.902) - (-1.159)}{8 - 3} = \frac{-5.743}{5} = -1.139 \approx -1 \tag{6}$$

Since we know that $s = -(q - p)$ we can now calculate our secondary error term q.

$$q = p - s = 3 - (-1) = 4 \tag{7}$$

Since $p = 3$ and $2^p = 8$ we can see in Figure 6c that it converges with the same slope until k=8, i.e. it roughly multiplies itself by 2 converging to 8. We can therefore say that we can trust our slope in $k = \{3..8\}$

Our best approximation of the range is row 8 in Figure 4.

k	Approximation A_h	Fraction F_h	Error estimate E_h
1	2.237327039413e+04	0.00000000	0.000000000000e+00
2	2.237302121565e+04	0.00000000	−3.559692690968e−02
3	2.237298822139e+04	7.55217806	−4.713464993464e−03
4	2.237298400328e+04	7.82203872	−6.025877862287e−04
5	2.237298347078e+04	7.92134046	−7.607144133155e−05
6	2.237298340391e+04	7.96329830	−9.552755467926e−06
7	2.237298339553e+04	7.98220289	−1.196756784339e−06
8	2.237298339448e+04	7.99163610	−1.497511610588e−07
9	2.237298339435e+04	7.98533976	−1.875326103930e−08
10	2.237298339433e+04	7.99202658	−2.346496330574e−09

Figure 4: Output of Richardson's fraction RK3

3.4 `Runge-Kutta 4`

We know that Richardson's fraction F_h, converges to 2^p. Therefore we can conclude by looking at Figure 5 that in this case $p = 4$. From Figure 6d we can calculate the slope by using $s = \frac{dy}{dx}$ which in this case is the following:

$$s = \frac{dy}{dx} = \frac{(-3.135) - (-0.2368)}{6 - 3} = \frac{-3.3718}{3} = -1.124 \approx -1 \tag{8}$$

Since we know that $s = -(q - p)$ we can now calculate our secondary error term q.

$$q = p - s = 4 - (-1) = 5 \tag{9}$$

Since $p = 4$ and $2^p = 16$ we can see in Figure 6d that it converges with the same slope until k=6, i.e. it roughly divides itself by 2 converging to 16. We can therefore say that we can trust our slope in $k = \{3..6\}$

Our best approximation of the range is row 6 in Figure 5.

k	Approximation A_h	Fraction F_h	Error estimate E_h
1	2.237296894265e+04	0.00000000	0.000000000000e+00
2	2.237298253527e+04	0.00000000	9.061748710034e−04
3	2.237298334202e+04	16.84861431	5.378334705407e−05
4	2.237298339111e+04	16.43565957	3.272357086341e−06
5	2.237298339413e+04	16.22025681	2.017450848750e−07
6	2.237298339432e+04	16.11384681	1.251998279865e−08
7	2.237298339433e+04	16.01675458	7.816803796838e−10
8	2.237298339433e+04	18.10674157	4.317068184415e−11
9	2.237298339433e+04	7.12000000	6.063298011820e−12
10	2.237298339433e+04	−1.04166667	−5.820766091347e−12

Figure 5: Output of Richardson's fraction RK4.

In the initial stage we wanted to compute the range of a shot given by parameters given in `a3f3.m`.

See `a3range.m` for Matlab source code.

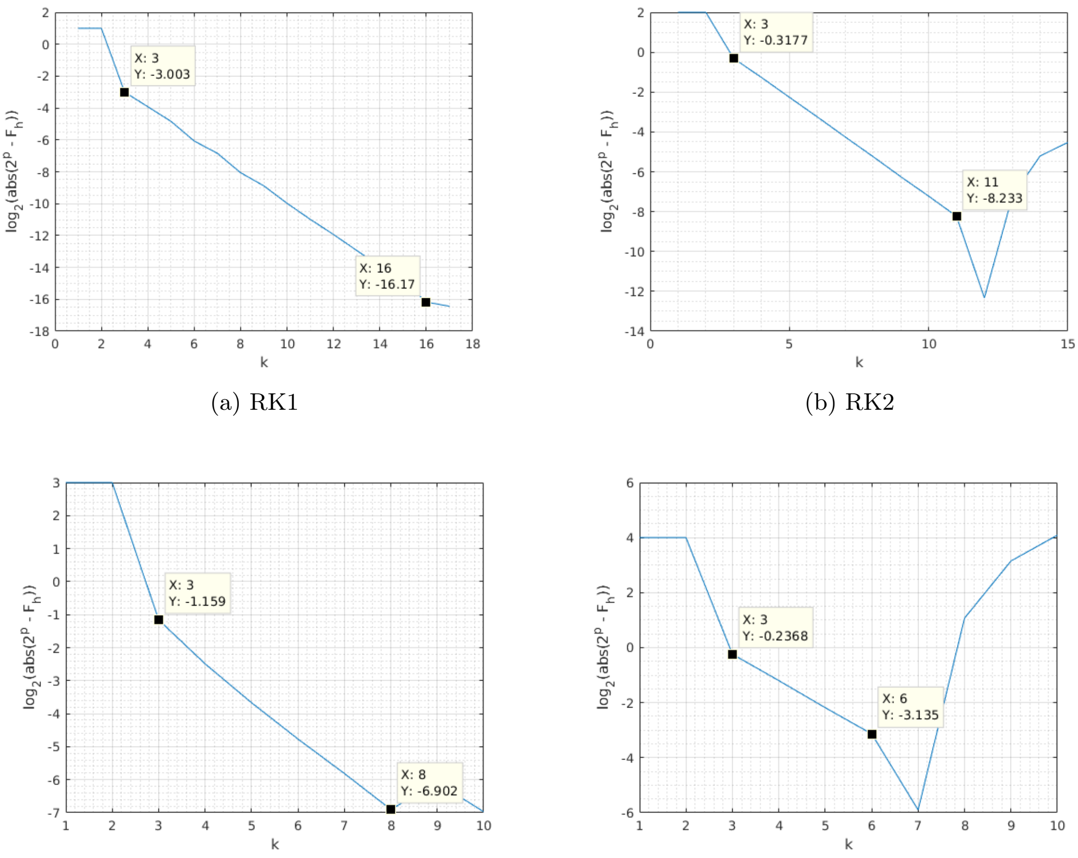


Figure 6: Richardson's fraction of shot range with `a3f3` parameters.

4 Compute trajectory time

To compute the flight time of the shot given by `a3f3.m` we develop a script called `3time.m`. The error estimate of rk2 is converging by roughly dividing itself by 2 to 4 which is 2^p which gives us an order of 2. Our error estimate is reliable as long as this monotonically behaviour occurs. In this case our best estimation has an error estimate of $\approx 3.57 * 10^{-8}$.

We know that Richardson's fraction F_h, converges to 2^p. Therefore we can conclude by looking at Figure 8 that in this case $p = 2$. From Figure 7 we can calculate the slope by using $s = \frac{dy}{dx}$ which in this case is the following:

$$s = \frac{dy}{dx} = \frac{(-9.847) - (-1.011)}{12 - 3} = \frac{-8.836}{9} = -0.982 \approx -1 \tag{10}$$

Since we know that $s = -(q - p)$ we can now calculate our secondary error term q.

$$q = p - s = 2 - (-1) = 3 \tag{11}$$

Since $p = 2$ and $2^p = 4$ we can see in Figure 7 that it converges with the same slope until k=12, i.e. it roughly divides itself by 2 converging to 4. We can therefore say that we can trust our slope in $k = \{3..12\}$

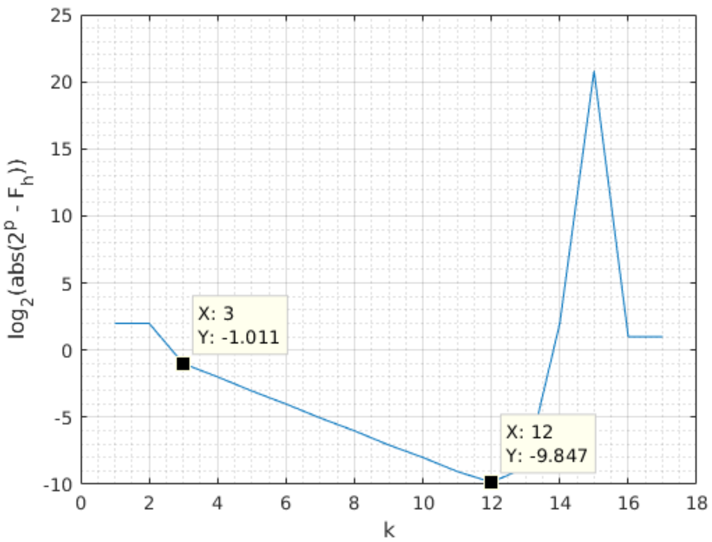


Figure 7: RK2

k	Approximation A_h	Fraction F_h	Error estimate E_h
1	7.840444726979e+01	0.00000000	0.000000000000e+00
2	7.841334458859e+01	0.00000000	2.965772931357e−03
3	7.841532345978e+01	4.49615863	6.596237306222e−04
4	7.841578893805e+01	4.25126439	1.551594231444e−04
5	7.841590188824e+01	4.12109312	3.765006481634e−05
6	7.841592969611e+01	4.06180657	9.269290449273e−06
7	7.841593659629e+01	4.03002319	2.300058833763e−06
8	7.841593831469e+01	4.01545956	5.728008953080e−07
9	7.841593874350e+01	4.00743369	1.429345909780e−07
10	7.841593885059e+01	4.00390580	3.569878970211e−08
11	7.841593887736e+01	4.00189521	8.920470880488e−09
12	7.841593888404e+01	4.00108570	2.229512574559e−09
13	7.841593888572e+01	3.99768971	5.577002563465e−10
14	2.237295894604e+04	0.00000000	7.431514335718e+03
15	2.237297117025e+04	1.8238e+06	4.074735665199e−03
16	2.237297728231e+04	2.00001352	2.037354061031e−03
17	2.237298033832e+04	2.00001127	1.018671289785e−03

Figure 8: Output of Richardson's fraction computing trajectory time with RK2.

Our best approximation of the range is row 12 in Figure 8.

See `a3time.m` for matlab source code.

5 Compute low trajectory

To compute the low firing solution for a target located 15 000m to the right of the gun given by `a3f3.m` we develop a script called `a3low.m`. In the script we calculate the elevation of the gun starting with a time step of 0.1. To calculate the elevation we use the bisection method. For each iteration we half the time step and double the max iteration. After we have done this x time (In our case 10 times) we perform Richardson's extrapolation on the approximated elevations.

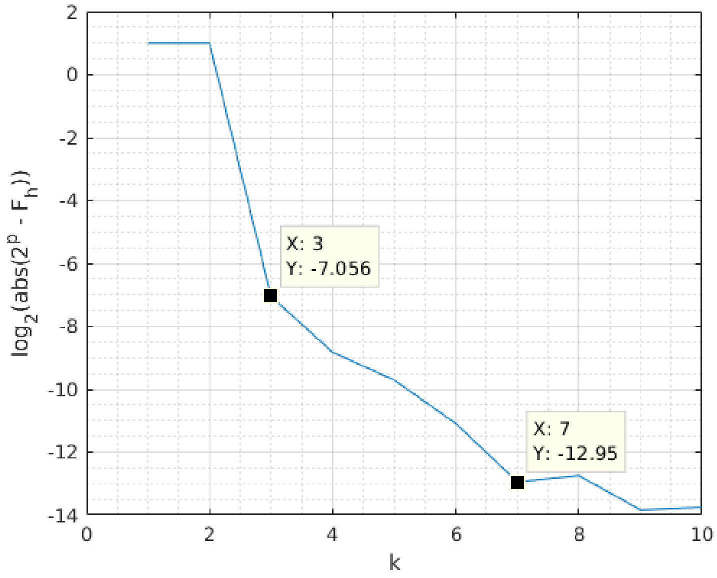


Figure 9: RK1

In the Figure 9 we can clearly observe a monotonous convergence in the range $k = \{3..7\}$ towards 2. Therefore the error estimates can only be trusted within these bounds. Our best approximation is row $k = 7$.

k	Approximation A_h	Fraction F_h	Error estimate E_h
1	2.745519462766e−01	0.00000000	0.000000000000e+00
2	2.748304712482e−01	0.00000000	2.785249716397e−04
3	2.749702590673e−01	1.99248385	1.397878190441e−04
4	2.750402300932e−01	1.99779576	6.997102595530e−05
5	2.750752364919e−01	1.99880675	3.500639866216e−05
6	2.750927437051e−01	1.99954147	1.750721315541e−05
7	2.751014978665e−01	1.99987323	8.754161476943e−06
8	2.751058752648e−01	1.99985491	4.377398302391e−06
9	2.751080640390e−01	1.99993148	2.188774133605e−06
10	2.751091584657e−01	1.99992757	1.094426704040e−06

Figure 10: Output of Richardson's fraction computing low trajectory with RK2

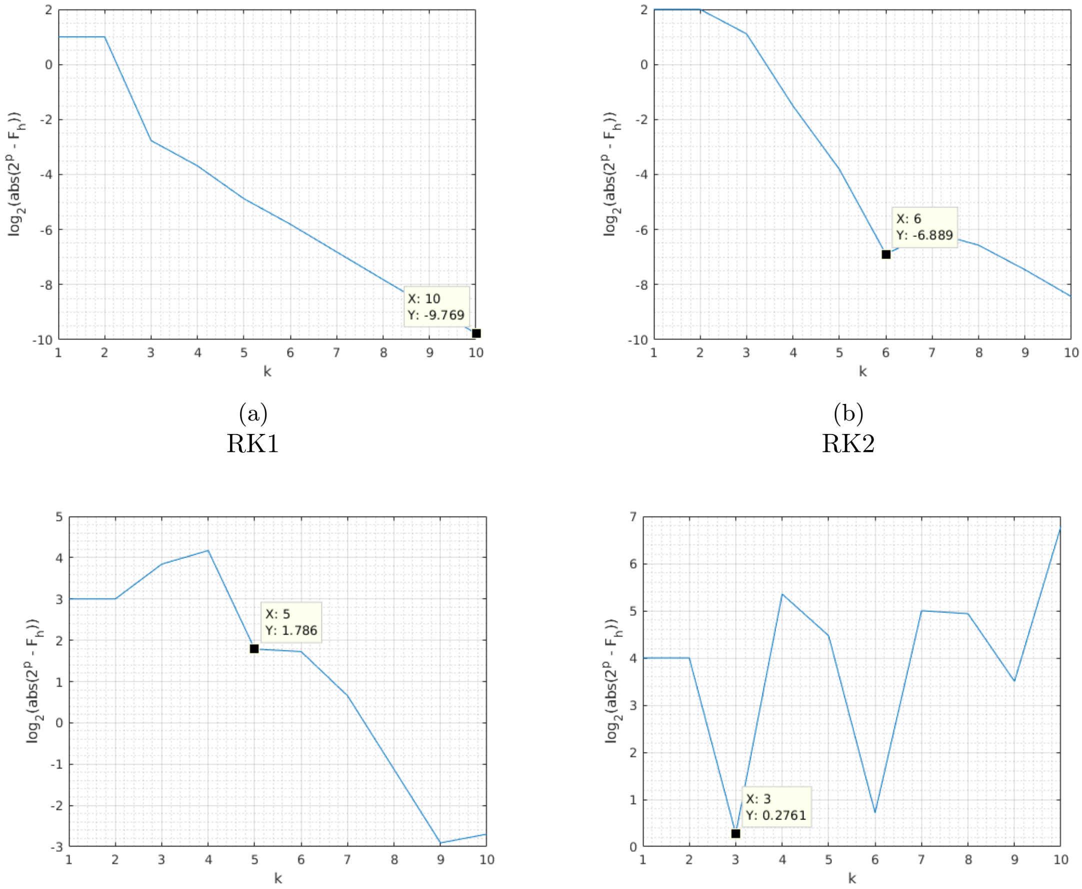
See `a3low.m` for Matlab source code.

6 Range of shot with `a3f4` parameters

We develop a script called `a3range_g7.m` which computes the range of a shot that is defined by `a3f4.m`. In Figure 6 we can see that rk1 converges from 3 out of the viewed data and that rk2 converges in $\{3..6\}$. We can also see that rk3 and rk4 does not converge to a smaller value and can therefore not be trusted. We can therefore conclude that the methods which can be used to estimate the range accurately is rk1 and rk2.

7 Range of shot using `range_rkx_sabotage`

We develop a script called `a3_range_sabotage.m` which uses `range_rkx_sabotage.m` to compute the range of a shot that is given by `a3f3.m`. The function range_rkx_sabotage is identical to



(a)
RK1

(b)
RK2

(c)
RK3

(d)
RK4

Richardson's fraction of shot with `a3f4` parameters

Figure 11:
fig:ass2

range rkx except that the tolerance is much larger, specifically, $tol = 2^3$. This results in a sloppy computation of the last step size. When looking at the result from the four methods, see Figure 12, we see that the initial method (a) RK1 gives us the most accurate estimate of the range. Since we can observe the monotonous convergence between $k = \{3, 4\}$ the error estimate and approximation is trustworthy.

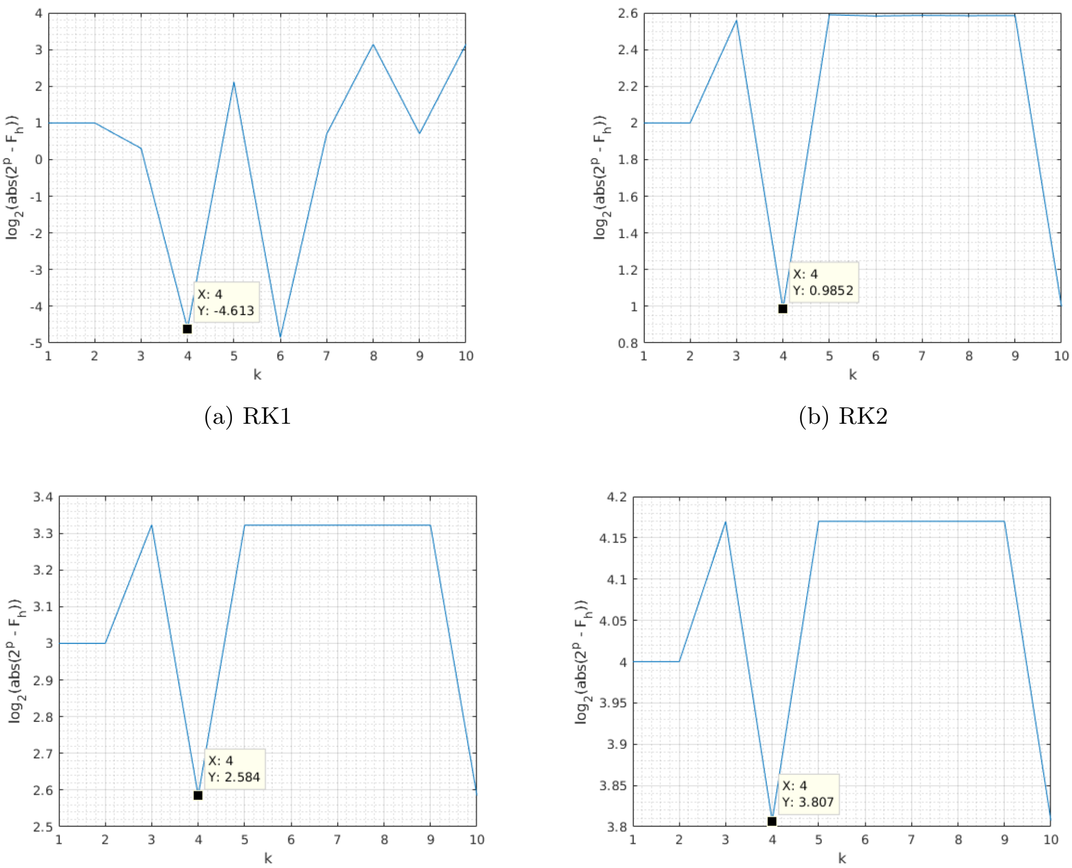
8 Compute length of trajectory

In this section we compute the length of the trajectory of a shot given parameters in `a3f3.m`. In the following subsections we compute the range of the trajectory using the Runge-Kutta 1-4 methods in decreasing step sizes h [1]. In the different methods we calculate Richardson's fraction to obtain the order of the primary error term p.

The source code can be found in `mya3length.m`, section 9.

8.1 `Runge-Kutta 1`

We know that Richardson's fraction F_h applies and are reliable as long as F_h converges monotonously towards 2^p. Using Runge–Kutta 1, In Figure 14, we can see that F_h converges monotonously towards $2 = 2^p$, therefore the order of the primary error term equals $p = 1$.



(a) RK1

(b) RK2

(c) RK3

(d) RK4

Figure 12: Richardson's fraction using `range_rkx_sabotage`

8.2 `Runge-Kutta 2`

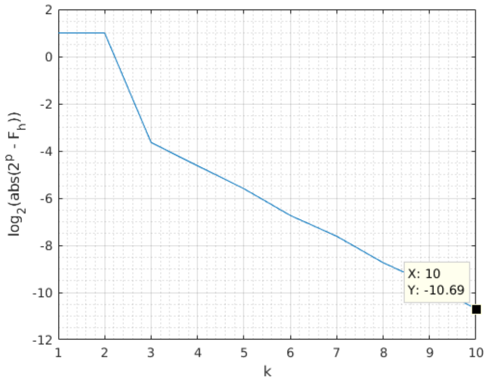
We know that Richardson's fraction F_h applies and are reliable as long as F_h converges monotonously towards 2^p. Using Runge–Kutta 2, In Figure 15, we can see that F_h converges monotonously towards $4 = 2^p$, therefore the order of the primary error term equals $p = 2$.

8.3 `Runge-Kutta 3`

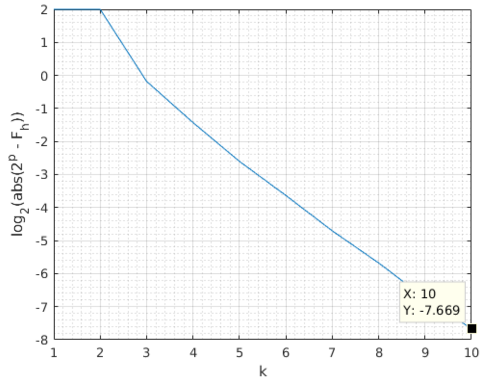
We know that Richardson's fraction F_h applies and are reliable as long as F_h converges monotonously towards 2^p. Using Runge–Kutta 3, In Figure 16, we can see that F_h converges monotonously towards $4 = 2^p$, therefore the order of the primary error term equals $p = 2$.

8.4 `Runge-Kutta 4`

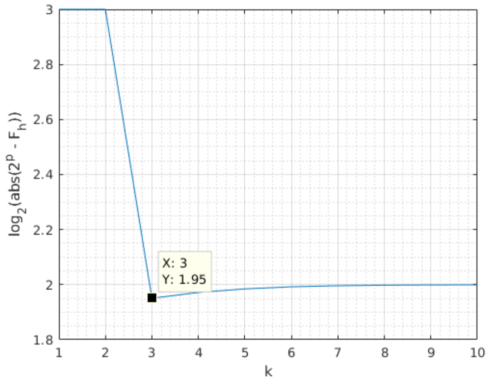
We know that Richardson's fraction F_h applies and are reliable as long as F_h converges monotonously towards 2^p. Using Runge–Kutta 4, In Figure 17, we can see that F_h converges monotonously towards $4 = 2^p$, therefore the order of the primary error term equals $p = 2$.



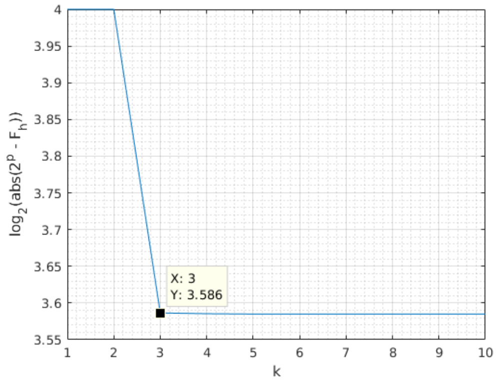
(a) RK1



(b) RK2



(c) RK3



(d) RK4

Figure 13: Richardson's fraction from computing length of trajectory

k	Approximation A_h	Fraction F_h	Error estimate E_h
1	2.778550749350e+04	0.00000000	0.000000000000e+00
2	2.797266970875e+04	0.00000000	1.871622152490e+02
3	2.806264768086e+04	2.08008928	8.997797210782e+01
4	2.810674546756e+04	2.04041923	4.409778670096e+01
5	2.812856952715e+04	2.02060421	2.182405959311e+01
6	2.813943058101e+04	2.00938692	1.086105385865e+01
7	2.814484731193e+04	2.00509385	5.416730916528e+00
8	2.814755250469e+04	2.00234563	2.705192760979e+00
9	2.814890425452e+04	2.00125252	1.351749831345e+00
10	2.814957992551e+04	2.00060362	6.756709924593e−01

Figure 14: Output of Richardson's fraction RK1.

9 mya3length.m

```
% Clean up
clear all

% Load parameters describing shot
a3f3

% Set initial time step
h0=1;
```

k	Approximation A_h	Fraction F_h	Error estimate E_h
1	2.815272680908e+04	0.00000000	0.000000000000e+00
2	2.815101077100e+04	0.00000000	−5.720126932235e−01
3	2.815046025197e+04	3.11712766	−1.835063418839e−01
4	2.815030851983e+04	3.62822946	−5.057738046526e−02
5	2.815026895635e+04	3.83515635	−1.318782752181e−02
6	2.815025886221e+04	3.91945055	−3.364713332606e−03
7	2.815025631423e+04	3.96162851	−8.493258076972e−04
8	2.815025567411e+04	3.98047048	−2.133732208070e−04
9	2.815025551370e+04	3.99065106	−5.346827310859e−05
10	2.815025547355e+04	3.99508585	−1.338351042553e−05

Figure 15: Output of Richardson's fraction RK2.

k	Approximation A_h	Fraction F_h	Error estimate E_h
1	2.815469282472e+04	0.00000000	0.000000000000e+00
2	2.815133289885e+04	0.00000000	−4.799894097965e−01
3	2.815052033276e+04	4.13495703	−1.160808700811e−01
4	2.815032107762e+04	4.07801817	−2.846502032064e−02
5	2.815027178877e+04	4.04260035	−7.041264998926e−03
6	2.815025953261e+04	4.02155947	−1.750879242796e−03
7	2.815025647707e+04	4.01112984	−4.365052523748e−04
8	2.815025571424e+04	4.00548737	−1.089768141225e−04
9	2.815025552366e+04	4.00282635	−2.722496667827e−05
10	2.815025547603e+04	4.00134657	−6.803951172125e−06

Figure 16: Output of Richardson's fraction RK3.

k	Approximation A_h	Fraction F_h	Error estimate E_h
1	2.815440305995e+04	0.00000000	0.000000000000e+00
2	2.815129497153e+04	0.00000000	−2.072058951012e−01
3	2.815051550969e+04	3.98747994	−5.196412223668e−02
4	2.815032047025e+04	3.99643187	−1.300262932151e−02
5	2.815027171259e+04	4.00018017	−3.250510917618e−03
6	2.815025952307e+04	3.99996736	−8.126343595601e−04
7	2.815025647588e+04	4.00024493	−2.031461505491e−04
8	2.815025571409e+04	4.00002268	−5.078624963062e−05
9	2.815025552364e+04	4.00009486	−1.269626130428e−05
10	2.815025547603e+04	3.99996378	−3.174094066101e−06

Figure 17: Output of Richardson's fraction RK4.

```

% Methods
m=["rk1","rk2","rk3","rk4"];

% Number of rows in table
kmax=10;

% Define the function needed for arc lenght
g=@(z)sqrt(z(3,:).^2+z(4,:).^2);

% Loop over methods
for i=1:4<
    % Select method
    method=m(i);

    % Initialize time step
```

```
    dt=h0;

    % Initialize maxstep
    maxstep=200;

    % Loop over approximations
    for k=1:kmax
      % Compute range
      [r, flag, t, tra]=range_rkx(param,v0,theta,method,dt,maxstep);
      % Save information
      a(k)=a3int(g,t,tra);
      % Decrease time step
      dt=dt/2;
      % Increase maxstep
      maxstep=maxstep*2;
    end

    % Run Richardsons techniques
    data=MyRichardson(a,i);

    % New figure
    h(i)=figure();

    % Print to screen
    rdifprint(data,i);
end
```

10 a3time.m

```
% A3F3  Physical parameters for firing a projectile

% Set parameters.
param.mass=10;
param.cali=0.088;

% Constant drag coefficient
param.drag=@(x)0.1873;

param.atmo=@(x)atmosisa(x);
param.grav=@(x)9.82;

% Select muzzle velocity and elevation
v0=780; theta=45*pi/180;

% From a3length VVVV

% Set initial time step
h0=1;

% Methods
m=["rk1","rk2","rk3","rk4"];

% Number of rows in table
kmax=10;

% Define the function needed for arc lenght
%g=@(z)sqrt(z(3,:).^2+z(4,:).^2);

% Loop over methods
for i=1:4
    % Select method
    method=m(i);

    % Initialize maxstep
    maxstep=200;

    % Loop over approximations
    for k=1:kmax

      %time step
      dt=2^-(k-1);

      % Compute range
      [r, flag, t, tra]=range_rkx(param,v0,theta,method,dt,maxstep);

      % VVVV -- Emil justerade 4:e
      % Tar sista tids-v rdet i tid-vektorn r
      a(k) = t(end);
      % ^^^ -- Emil justerade 4:e

      % Save information
      %a(k)=a3int(g,t,tra);
      %r + "h" -> Rich

      % Decrease time step
      %dt=dt/2;

      % Increase maxstep
      maxstep=maxstep*2;
    end

    % Run Richardsons techniques
    data=MyRichardson(a,i);

    % New figure
    h(i)=figure();

    % Print to screen
    rdifprint(data,i);
end
```

11 a3range_sabotage.m

```
% A3F4  Physical parameters for firing a projectile

% Load standard shell models
load shells

% A3F3  Physical parameters for firing a projectile

% Set parameters.
param.mass=10;
param.cali=0.088;

% Constant drag coefficient
param.drag=@(x)0.1873;

param.atmo=@(x)atmosisa(x);
param.grav=@(x)9.82;

% Select muzzle velocity and elevation
v0=780; theta=45*pi/180;



%---------------------------
h0=1;

% Methods
m=["rk1","rk2","rk3","rk4"];

% Number of rows in table
kmax=10;

% Define the function needed for arc lenght
%g=@(z)sqrt(z(3,:).^2+z(4,:).^2);

% Loop over methods
for i=1:4
    % Select method
    method=m(i);


    % Initialize maxstep
    maxstep=200;

    % Loop over approximations
    for k=1:kmax

      %time step
      dt=2^-(k-1);

      % Compute range
      [r, flag, t, tra]=range_rkx_sabotage(param,v0,theta,method,dt,maxstep);


      a(k) = r;

      % Save information
      %a(k)=a3int(g,t,tra);
      %r + "h" -> Rich

      % Decrease time step
      %dt=dt/2;

      % Increase maxstep
      maxstep=maxstep*2;
    end

    % Run Richardsons techniques
    data=MyRichardson(a,i);

    % New figure
    h(i)=figure();

    % Print to screen
    rdifprint(data,i);
end
```

12 a3range_g7.m

```
% A3F4  Physical parameters for firing a projectile

% Load standard shell models
load shells

% Set parameters.
param.mass=10;
param.cali=0.088;

% This drag coefficient is not constant
param.drag=@(x)mcg7(x);

param.atmo=@(x)atmosisa(x);
param.grav=@(x)9.82;

% Select muzzle velocity and elevation
v0=780; theta=45*pi/180;


%---------------------------
h0=1;

% Methods
m=["rk1","rk2","rk3","rk4"];

% Number of rows in table
kmax=10;

% Define the function needed for arc lenght
%g=@(z)sqrt(z(3,:).^2+z(4,:).^2);

% Loop over methods
for i=1:4
    % Select method
    method=m(i);


    % Initialize maxstep
    maxstep=200;

    % Loop over approximations
    for k=1:kmax

      %time step
      dt=2^-(k-1);

      % Compute range
      [r, flag, t, tra]=range_rkx(param,v0,theta,method,dt,maxstep);


      a(k) = r;

      % Save information
      %a(k)=a3int(g,t,tra);
      %r + "h" -> Rich

      % Decrease time step
      %dt=dt/2;

      % Increase maxstep
      maxstep=maxstep*2;
    end

    % Run Richardsons techniques
    data=MyRichardson(a,i);

    % New figure
    h(i)=figure();

    % Print to screen
    rdifprint(data,i);
end
```

13 a3range.m

```
% A3F3  Physical parameters for firing a projectile

% Set parameters.
param.mass=10;
param.cali=0.088;

% Constant drag coefficient
param.drag=@(x)0.1873;

param.atmo=@(x)atmosisa(x);
param.grav=@(x)9.82;

% Select muzzle velocity and elevation
v0=780; theta=45*pi/180;

% From a3length VVVV

% Set initial time step
h0=1;

% Methods
m=["rk1","rk2","rk3","rk4"];

% Number of rows in table
kmax=10;

% Define the function needed for arc lenght
%g=@(z)sqrt(z(3,:).^2+z(4,:).^2);

% Loop over methods
for i=1:4
    % Select method
    method=m(i);


    % Initialize maxstep
    maxstep=200;

    % Loop over approximations
    for k=1:kmax

      %time step
      dt=2^-(k-1);

      % Compute range
      [r, flag, t, tra]=range_rkx(param,v0,theta,method,dt,maxstep);


      a(k) = r;

      % Save information
      %a(k)=a3int(g,t,tra);
      %r + "h" -> Rich

      % Decrease time step
      %dt=dt/2;

      % Increase maxstep
      maxstep=maxstep*2;
    end

    % Run Richardsons techniques
    data=MyRichardson(a,i);

    % New figure
    h(i)=figure();

    % Print to screen
    rdifprint(data,i);
end
```

14 a3low.m

```
clear all;

% ////////////////////////////////////////////////////////////////////////
%  Initial setup of the gun and the method used to compute trajectories
% ///////////////////////////////////////////////////////////////////////

% Load shells models
load shells.mat

% Set parameters.
param.mass=10;
param.cali=0.088;

% Constant drag coefficient
param.drag=@(x)0.1873;
param.atmo=@(x)atmosisa(x);
param.grav=@(x)9.82;

% Define muzzle velocity
v0=780;

% Select the method which will be used to integrate the trajectory
method='rk1';

% Select the basic time step size and the maximum number of time steps
dt=0.1; maxstep=2000;

% Define location of target
d=15000;

alow=0;
blow=(pi/4);
delta=10^-15;
eps=10^-15;
maxit=200;

n = 10;
result = zeros(1, n);

for i=1:n
    % Define the range function
    range=@(theta)range_rkx(param,v0,theta,method,dt,maxstep);
    % Define the residual function res
    res=@(theta)range(theta)-d;

    [x, flag, it, a, b, his, res]=bisection(res,alow,blow,res(alow),res(blow),delta,eps,maxit,0);

    result(i) = x;
    dt = dt / 2;
    maxstep = maxstep * 2;
end

degree=1;
% Run Richardsons techniques
data=MyRichardson(result,degree);

% New figure
h(degree)=figure();

% Print to screen
rdifprint(data,degree);
```

15 MyRichardson.m

```
function data=MyRichardson(a,p,t)

% MyRichardson Computational kernel for Richardson's technique
%
% Does Richardson extrapolation for a set of values assuming that the
% user has determined the order of the primary error term correctly
%
% CALL SEQUENCE: data=MyRichardson(val,p);
%
% INPUT:
%   a       array of m approximations of t, such that if a(i) corresponds
%           stepsize h, then a(i+1) corresponds to stepsize h/2
%   p       the order of the primary order term
%   t       (optional) the target value of the approximations
%
% OUTPUT:
%   data   an array of information such that
%             data(i,1) = i
%             data(i,2) = a(i)
%             data(i,3) = Richardson's fraction for i > 2
%             data(i,4) = Richardson's error estimate for i > 1
%           if the exact target value is supplied, then
%             data(i,5) = exact error
%             data(i,6) = comparision of error estimate to exact error
%
% MINIMAL WORKING EXAMPLE: A3F2

% PROGRAMMING by Carl Christian K. Mikkelsen (spock@cs.umu.se)
%   2015-12-10  Initial programming amd testing
%   2018-12-09  Printing moved to minimal working example
%   2018-12-09  Skeleton extracted from working code
%   2018-12-14  Skeleton extracted and edited by:
%               Betty T rnkvist (et16btt@cs.umu.se)
%               Emil S derlind (id15esd@cs.umu.se)
%               Jonas Sj din (id16jsn@cs.umu.se)

% Reshape the input array as a colum vector
m=numel(a); a=reshape(a,m,1);

% Is the target value known?
if ~exist('target','var')
    % Set a flag to indicate that the target value is unknown
    flag=0;
    % Allocate space for the table used to print the results
    data=zeros(m,4);
else
    % Set a flag to indicate that the the target value is known
    flag=1;
    % Allocate space for the table used to print the results
    data=zeros(m,6);
end

% Initialize the first and the second columns of data
for i=1:m
    data(i, 1) = i;
    data(i, 2) = a(i);
end

% Process the data, computing Richardson's fractions
for i=3:m
    data(i, 3) = (a(i - 1) - a(i - 2)) / (a(i) - a(i - 1));
end

% Compute Richardson's error estimates assuming order p is correct!
for i=2:m
    data(i, 4) = (a(i) - a(i - 1)) / (2^p - 1);
end

% If possible, then compute the error and compare it to the error estimate
if (flag==1)
    for i=1:m
        % Compute the exact error
        data(i,5) = abs(data(i,3) - t);

        % Compare the error estimate to the true error
        % i.e. log10(abs(relative error))
        rel = data(i, 5) / t;
        data(i,6) = log10(abs(rel));

```

```
    end
end
```

16 MyRichardson_MWE.m

```
% A3F2  Minimal working example for A3F1

% Set trial function
f=@(x)exp(x);

% Set point where we want to estimate the derivative
x=1;

% Define the finite difference approximation
D1=@(g,x,h)(g(x+h)-g(x))./h;

% Set the theoretical order of the method
p=1;

% Set the basic stepsize
h0=0.125;

% Set the number of times we will reduce the stepsize by a factor of 2
kmax=20;

% Define the real derivative
df=@(x)exp(x);

% Allocate space for approximations
a=zeros(kmax,1);

% Construct the different approximations
h=h0;
for i=1:kmax
    % Compute approxmation
    a(i)=D1(f,x,h);
    % Reduce step size
    h=h/2;
end

% Compute target value
t=df(x);

% Apply Richardson's techniques
data=MyRichardson(a,p,t);

% Display results
rdifprint(data,p);
```

References

[1] C. Runge, "Über die numerische auflösung von differentialgleichungen," *Mathematische Annalen*, vol. 46, no. 2, pp. 167–178, 1895.