# Numeryczne metody rozwiązywania ODE

## 1. Numeryczne metody rozwiązywania równań różniczkowych zwyczajnych pierwszego rzędu

### 1.1. Równania pierwszego rzędu

Rozważamy równanie

$$y' = f(t, y) \tag{1}$$

**Zagadnienie Cauche'go** polega na znalezieniu funkcji $y = y(t)$ spełniającej powyższe równanie różniczkowe (w przedziale $(a, b)$ oraz **warunek początkowy**

$$y(t_0) = y_0,$$

dla danych $t_0$, $y_0$.

**Przykład 1**. Niech będzie

$$f(t, y) = 1 - 2t + 4y.$$

In [1]:
```
f(t,y) = 1-2t+4y;
```

Można sprawdzić, że rozwiązanie ogólne wyraża się wzorem

$$y(t) = c_1 \exp(4t) + \frac{t}{2} - \frac{1}{8}$$

Rozpatrzmy następujące **zagadnienie początkowe**

$$\begin{cases} y' & = f(t, y), \\ y(0) & = 1 \end{cases}$$

Można sprawdzić, że rozwiązaniem jest funkcja

$$y(t) = \frac{9}{8}e^{4t} + t/2 - \frac{1}{8}.$$

In [2]:
```
t0 = 0.0;
y0 = 1.0;

c1 = ( y0+1/8-t0/2 ) / exp(4*t0);
@show c1
exactSolution(t) = c1 * exp(4t)+t/2-1/8;
```

```
c1 = 1.125
```

In [3]:
```
using Plotly
plot( exactSolution, 0.0, 2.0 )
```

### 1.2. Metody numeryczne

Rozważamy aproksymację rozwiązania $y(t)$ w przedziale $(t_0, b)$ w równoodległych punktach

$$t_j = t_0 + jh, \qquad t_N = b$$

### Metoda Eulera * Najprostszą metodą jest **metoda jawna Eulera**

$$y_{n+1} = y_n + hf(t_n, y_n)$$

Jest to metoda rzędu pierwszego.

In [4]:
```
# Metoda jawna Eulera
a = t0
b = 2.0       # konstruujemy rozwiązanie y(t) dla t \in (0, 2)

N = 10000  # liczba iteracji w metodzie Eulera
h = (b-a)/N   # krok w metodzie Eulera

tic()
t, y = t0, y0
for i=1:N
    y = y+h*f(t,y)
    t = t+h
end
toc()

@printf("Liczba kroków = %d\n", N);
@printf("Krok = %.2e\n", h);
@printf("Rozwiązanie przybliżone = %5.16f\n", y);
exact = exactSolution(b)
@printf("Rozwiązanie dokładne    = %5.16f\n", exact);
@printf("Błąd bezwzględny        = %5.3e\n", abs(y-exact));
```

```
elapsed time: 0.032921974 seconds
Liczba kroków = 10000
Krok = 2.00e-04
Rozwiązanie przybliżone = 3343.7441404238365976
Rozwiązanie dokładne    = 3354.4527354219444533
Błąd bezwzględny        = 1.071e+01
```

### 1.3. Metody niejawne

Dużo lepsze rezultaty można uzyskać stosując **metody niejawne**.

— **wzór wsteczny Eulera**

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$$

Jak widać, niewiadoma $y_{n+1}$ jest podana wyżej w sposób niejawny. Aby zaprogramować taką metodę, należy wcześniej rozwiązać powyższe równanie ze względnu na niewiadomą $y_{n+1}$. Metoda ta jest również rzędu pierwsego. >W wypadku, gdy $f(t, y)$ jest funkcją liniową zmiennej $y$, to sytuacja jest bardzo prosta. Mianowicie, mamy

$$y_{n+1} = y_n + h\,(1 - 2t_{n+1} + 4y_{n+1})\,,$$

czyli

$$y_{n+1} = \frac{y_n + h - 2ht_{n+1}}{1 - 4h} = y_n + \frac{h}{1-4h} f(t_{n+1}, y_n).$$

In [5]:
```
# wzór wsteczny Eulera (metoda niejawna)
tic()
t, y = t0, y0
```

```
        for i=1:N
            y = y+h/(1-4h)*f(t+h,y)
            t = t+h
        end
        toc()

        @printf("Liczba kroków = %d\n", N);
        @printf("Krok = %.2e\n", h);
        @printf("Rozwiązanie przybliżone = %5.16f\n", y);
        @printf("Rozwiązanie dokładne    = %5.16f\n", exact);
        @printf("Błąd bezwzględny        = %5.3e\n", abs(y-exact));
```

```
elapsed time: 0.034434611 seconds
Liczba kroków = 10000
Krok = 2.00e-04
Rozwiązanie przybliżone = 3365.2071180588568495
Rozwiązanie dokładne    = 3354.4527354219444533
Błąd bezwzględny        = 1.075e+01
```

— **wzór trapezów** Ponieważ, rozwiązanie dokładne $y(t)$ spełnia równanie całkowe

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(y))\,\mathrm{d}t,$$

więc jeśli powyższą całkę przybliżamy za pomocą **wzoru trapezów**, to otrzymujemy następującą metodę niejawną

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right].$$

Znów, jeśli $f$ jest funkcją liniową zmiennej $y$, to łatwo uzyskujemy wzory dla $y_{n+1}$:

$$y_{n+1} = y_n + \frac{h}{2}\left[1 - 2t_n + 4y_n + 1 - 2t_{n+1} + 4y_{n+1}\right]$$

$$y_{n+1} = y_n + h\left[1 - t_n + 2y_n - t_{n+1} + 2y_{n+1}\right]$$

$$y_{n+1} = \frac{y_n + h - ht_n + 2hy_n - ht_{n+1}}{1 - 2h}$$

$$y_{n+1} = y_n + h\frac{f(t_{n+1}, y_n) + h}{1 - 2h}$$

```
In [6]: # wzór trapezów (metoda niejawna)
        h = (b-a)/N      # krok w metodzie

        tic()
        t, y = t0, y0
        for i=1:N
            y = y+h/(1-2h)*(f(t+h,y)+h)
            t = t+h
        end
        toc()

        @printf("Liczba kroków = %d\n", N);
        @printf("Krok = %.2e\n", h);
        @printf("Rozwiązanie przybliżone = %5.16f\n", y);
        @printf("Rozwiązanie dokładne    = %5.16f\n", exact);
        @printf("Błąd bezwzględny        = %5.3e\n", abs(y-exact));
```

```
elapsed time: 0.025064992 seconds
Liczba kroków = 10000
Krok = 2.00e-04
Rozwiązanie przybliżone = 3354.4541662822439321
```

```
Rozwiązanie dokładne     = 3354.4527354219444533
Błąd bezwzględny         = 1.431e-03
```

Można udowodnić, że jest to metoda **rzędu drugiego**.

### 1.4. Ulepszone metody

W metodach niejawnych, największym problemem jest rozwiązywanie równania, często nieliniowego, ze względu na niewiadomą $y_{n+1}$. Można się tego problemu pozbyć, jeśli wartość jej *przybliżymy* za pomocą innej — jawnej — metody. Na przykład: * **ulepszony wzór wsteczny Eulera** określony jest w następujący sposób:

$$y_{n+1} = y_n + hf(t_{n+1}, \hat{y}_{n+1}), \qquad \text{gdzie} \quad \hat{y}_{n+1} = y_n + hf(t_n, y_n).$$

* **ulepszony wzór trapezów**

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, \hat{y}_{n+1})\right], \qquad \text{gdzie} \quad \hat{y}_{n+1} = y_n + hf(t_n, y_n).$$

#### 1.4.1. Predict-Corrector

Co ciekawe, proces ulepszania można iterować, tzn. obliczać

$$\begin{aligned}
\hat{y}^{(0)}_{n+1} &= (\text{wzór jawny/niejawny/ulepszony metody})\\
\hat{y}^{(1)}_{n+1} &= (\text{ulepszony wzór niejawny metody dla } \hat{y}_{n+1} := \hat{y}^{(0)}_{n+1})\\
\hat{y}^{(2)}_{n+1} &= (\text{ulepszony wzór niejawny metody dla } \hat{y}_{n+1} := \hat{y}^{(1)}_{n+1})\\
&\vdots
\end{aligned}$$

W ten sposób otrzymujemy * **iterowany ulepszony wzór wsteczny Eulera**

$$\begin{aligned}
\hat{y}^{(0)}_{n+1} &= y_n + hf(t_n, y_n) && (\text{jawny wzór Eulera})\\
\hat{y}^{(1)}_{n+1} &= y_n + hf(t_{n+1}, \hat{y}^{(0)}_{n+1}) && (\text{ulepszony wzór wsteczny Eulera})\\
\hat{y}^{(2)}_{n+1} &= y_n + hf(t_{n+1}, \hat{y}^{(1)}_{n+1}) && (\text{ulepszony wzór wsteczny Eulera})\\
&\vdots\\
\hat{y}^{(K)}_{n+1} &= y_n + hf(t_{n+1}, \hat{y}^{(K-1)}_{n+1}) && (\text{ulepszony wzór wsteczny Eulera})
\end{aligned}$$

* **iterowany ulepszony wzór trapezów**

$$\begin{aligned}
\hat{y}^{(0)}_{n+1} &= y_n + hf(t_n, y_n) && (\text{jawny wzór Eulera})\\
\hat{y}^{(1)}_{n+1} &= y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, \hat{y}^{(0)}_{n+1})\right] && (\text{ulepszony wzór trapezów})\\
\hat{y}^{(2)}_{n+1} &= y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, \hat{y}^{(1)}_{n+1})\right] && (\text{ulepszony wzór trapezów})\\
&\vdots\\
\hat{y}^{(K)}_{n+1} &= y_n + \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, \hat{y}^{(K-1)}_{n+1})\right] && (\text{ulepszony wzór trapezów})
\end{aligned}$$

In [7]:
```
# Iterowany ulepszony wzór trapezów

h = (b-a)/N      # krok w metodzie

tic()
t,y = t0,y0
for i=1:N
    y_predictor = y+h*f(t,y)                                    # jawny wzór Eulera
    y_corrector = y_predictor
```

```
            for k=1:5
                y_corrector = y + h/2*( f(t,y) + f(t+h,y_corrector) )   # ulepszony wzór trapezów
            end
            t,y = t+h, y_corrector
        end
        toc()

        @printf("Liczba kroków = %d\n", N);
        @printf("Krok = %.2e\n", h);
        @printf("Rozwiązanie przybliżone = %5.16f\n", y);
        @printf("Rozwiązanie dokładne    = %5.16f\n", exact);
        @printf("Błąd bezwzględny        = %5.3e\n", abs(y-exact));
        #  wyniki podobne do metody niejawnej
```

```
elapsed time: 0.057214324 seconds
Liczba kroków = 10000
Krok = 2.00e-04
Rozwiązanie przybliżone = 3354.4541662822389299
Rozwiązanie dokładne    = 3354.4527354219444533
Błąd bezwzględny        = 1.431e-03
```

### 1.5. Metody z punktem środkowym

Przypomnijmy, że wszystkie poprzednie metody otrzymano stąd, że

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y(t))\,\mathrm{d}t.$$

Do przybliżenia powyższej całki używano jedynie wartości na końcach przedziału całkowania, tj. w punktach $t_n$, $t_{n+1}$. Oczywiste jest, że jeśli użyjemy większej informacji o funkcji $f(t, y(t))$, to powinniśmy otrzymać jeszcze lepsze przybliżenia powyższej całki.

W metodach z punktem środkowym wykorzystuje się dodatkowy punkt $t_{n+1/2} = t_n + h/2$. Podobnie, jak poprzednio, możemy w ten sposób otrzymywać metody jawne, niejawne, ulepszone, a nawet możemy stosować itrowane ulepszanie.

Na przykład, korzystając z przybliżenia

$$\int_a^b f(t)\,\mathrm{d}t \approx (b-a)\, f(\tfrac{a+b}{2}),$$

otrzymujemy następującą **jawną metodę punktu środkowego**

$$y_{n+1} = y_n + hf(t_n + \tfrac{h}{2}, y_{n+1/2}), \qquad \text{gdzie} \quad y_{n+1/2} = y_n + \tfrac{h}{2} f(t_n, y_n).$$

Można ją zapisać w następującej postaci:

$$\begin{aligned} K_1 &= h\, f(t_n, y_n) \\ K_2 &= h\, f(t_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}K_1) \\ y_{n+1} &= y_n + K2. \end{aligned}$$

Drobna zmiana daje **niejawną metodę z punktem środkowym**

$$\begin{aligned} K_1 &= h\, f\left(t_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}K_1\right) \\ y_{n+1} &= y_n + K_1. \end{aligned}$$

In [8]:
```
# Jawna metoda punktu środkowego

        h = (b-a)/N      # krok w metodzie

        tic()
```

```
t,y = t0,y0
for i=1:N
    K1 = h*f(t,y)
    K2 = h*f(t+h/2,y+K1/2)
    y = y + K2
    t = t+h
end
toc()

@printf("Liczba kroków = %d\n", N);
@printf("Krok = %.2e\n", h);
@printf("Rozwiązanie przybliżone = %5.16f\n", y);
@printf("Rozwiązanie dokładne    = %5.16f\n", exact);
@printf("Błąd bezwzględny        = %5.3e\n", abs(y-exact));
```

```
elapsed time: 0.031048116 seconds
Liczba kroków = 10000
Krok = 2.00e-04
Rozwiązanie przybliżone = 3354.4498754200058102
Rozwiązanie dokładne    = 3354.4527354219444533
Błąd bezwzględny        = 2.860e-03
```

### 1.6. Metody Rungego-Kutty

Jeśli w metodzie punktu środkowego zastosujemy proces ulepszania, to możemy uzyskać w ten sposób, np. następujące dwa warianty **ulepszonej metody punktu środkowego**:

$$
\begin{aligned}
K_1 &= h\,f(t_n, y_n) \\
K_2 &= h\,f(t_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}K_1) \\
K_3 &= h\,f(t_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}K_2) \quad \text{(ulepszona wartość } K_2\text{)} \\
y_{n+1} &= y_n + K_3 \qquad\qquad\qquad\qquad \text{(wariant 1)} \\
y_{n+1} &= y_n + \tfrac{1}{2}K_2 + \tfrac{1}{2}K_3 \qquad\qquad \text{(wariant 2)}
\end{aligned}
$$

W ogólności, można tak

$$y_{n+1} = y_n + c_1K_1 + c_2K_2 + c_3K_3, \qquad \text{gdzie} \quad c_1 + c_2 + c_3 = 1.$$

W metodzie Rungego-Kutty stosuje się ten sam pomysł, z tym, że starutujemy od wzoru

$$\int_a^b f(t)\mathrm{d}t = h(\tfrac{1}{6}f(a) + \tfrac{4}{6}f(\tfrac{a+b}{2}) + \tfrac{1}{6}f(b)).$$

Otrzumujemy wówczas związek

$$y_{n+1} = y_n + \tfrac{1}{6}hf(t_n, y_n) + \tfrac{4}{6}hf(t_n + \tfrac{h}{2}, y_{n+1/2}) + \tfrac{1}{6}hf(t_n + h, y_{n+1}),$$

w którym występują nieznane $y_{n+1/2}$ i $y_{n+1}$ odpowiednio w drugim i trzecim składniku. Proponuje się rozbić drugi składnik na dwa:

$$\tfrac{4}{6}hf(t_n + \tfrac{h}{2}, y_{n+1/2}) = \tfrac{2}{6}hf(t_n + \tfrac{h}{2}, y_{n+1/2}) + \tfrac{2}{6}hf(t_n + \tfrac{h}{2}, y_{n+1/2}),$$

a następnie w pierwszym zastosować przybliżenie

$$y^{(0)}_{n+1/2} := y_n + \tfrac{h}{2}f(t_n, y_n) \qquad \text{czyli wzór jawny Eulera,}$$

a w drugim — przybliżenie

$$y^{(1)}_{n+1/2} := y_n + \tfrac{h}{2}f(t_n + \tfrac{h}{2}, y^{(0)}_{n+1}) \qquad \text{czyli ulepszony wzór jawny Eulera,}$$

Klasyczną wersję tej metody opisują więc wzory

$$\begin{aligned}
K_1 &= h\, f(t_n, y_n) \\
K_2 &= h\, f(t_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}K_1) \\
K_3 &= h\, f(t_n + \tfrac{h}{2}, y_n + \tfrac{1}{2}K_2) && \text{(ulepszona wartość } K_2\text{)} \\
K_4 &= h\, f(t_n + h, y_n + K_3) \\
y_{n+1} &= y_n + \tfrac{1}{6}K_1 + \tfrac{2}{6}K_2 + \tfrac{2}{6}K_3 + \tfrac{1}{6}K_4
\end{aligned}$$

In [10]:
```
# Metoda Rungego-Kutty

h = (b-a)/N      # krok w metodzie

tic()
t,y = t0,y0
for i=1:N
    K1 = f(t, y)
    K2 = f(t+h/2   , y+h*K1/2)
    K3 = f(t+h/2   , y+h*K2/2)
    K4 = f(t+h     , y+h*K3  )
    y = y + h/6*(K1+2*(K2+K3)+K4)
    t = t+h
end
toc()

@printf("Liczba kroków = %d\n", N);
@printf("Krok = %.2e\n", h);
@printf("Rozwiązanie przybliżone = %5.16f\n", y);
@printf("Rozwiązanie dokładne    = %5.16f\n", exact);
@printf("Błąd bezwzględny        = %5.3e\n", abs(y-exact));
```

```
elapsed time: 0.065161376 seconds
Liczba kroków = 10000
Krok = 2.00e-04
Rozwiązanie przybliżone = 3354.4527354218771507
Rozwiązanie dokładne    = 3354.4527354219444533
Błąd bezwzględny        = 6.730e-11
```

## 2. Równania różniczkowe wyższych rzędów

### 2.1. Przykład (równanie drugiego rzędu)

$$y'' = f(t, y, y')$$

Wówczas zagadnienie początkowe Cauchy'ego ma następujące **warunki początkowe**

$$\begin{cases} y(t_0) &= y_0, \\ y'(t_0) &= y_0'. \end{cases}$$

Równanie różniczkowe możemy sprowadzić do układu dwóch równań rzędu pierwszego. Mianowicie wprowadzamy oznaczenia

$$a(t) = y(t), \qquad b(t) = y'(t).$$

Szukamy zatem dwóch funkcji $a(t), b(t)$ spełniających układ równań

$$\begin{cases} a' &= b, \\ b' &= f(t, a, b). \end{cases}$$

### 2.1.1. Przykład

Rozważmy kulkę o masie $m$ zawieszoną na sprężynie o sprężystości $k$. Jeśli początkowa długość spreżyny wynosi $L$, a przez zawieszenie kulki, jej długość wzrosła o $\Delta L$, to łatwo sprawdzić, że $k = mg/\Delta L$, gdzie $g$ oznacza przyspieszenie związane z grawitacją.

Jeśli $x(t)$ oznacza długość sprężyny w chwili $t$, to z drugiej zasady dynamiki Newtona, można łatwo otrzymać następujące równanie ruchu kulki:

$$mx'' = mg - k(x - L).$$

Stąd

$$mx'' = -k(x - L - \Delta L),$$

a dodając jeszcze opór — otrzymujemy równanie

$$mx'' = -k(x - L - \Delta L) - rx',$$

gdzie $r$ jest pewnym współczynnikiem ($r = 0$ oznacza brak oporu).

## 2.2. Rozwiązywanie układu równań pierwszego rzędu

Całą teorię rozwiązywania równań pierwszego rzędu można również zastosować do układów równań. Wystarczy zastąpić funkcję $y = y(t)$ funkcją wektorową $\vec{y} = \vec{y}(t) = [y_1(t), y_2(t)]$ oraz przyjąć notację

$$\vec{y}' = [y_1'(t), y_2'(t)]$$

Na przykład, dla równania sprężyny otrzymujemy następujący układ równań pierwszego rzędu

$$\begin{cases} y_1' & = y_2, \\ y_2' & = -k(y_1 - L - \Delta L) - ry_2. \end{cases}$$

Inaczej możemy napisać

$$\vec{y}' = f(t, \vec{y}),$$

gdzie

$$f(t, \vec{y}) = [y_2, -k(y_1 - L - \Delta L) - ry_2].$$

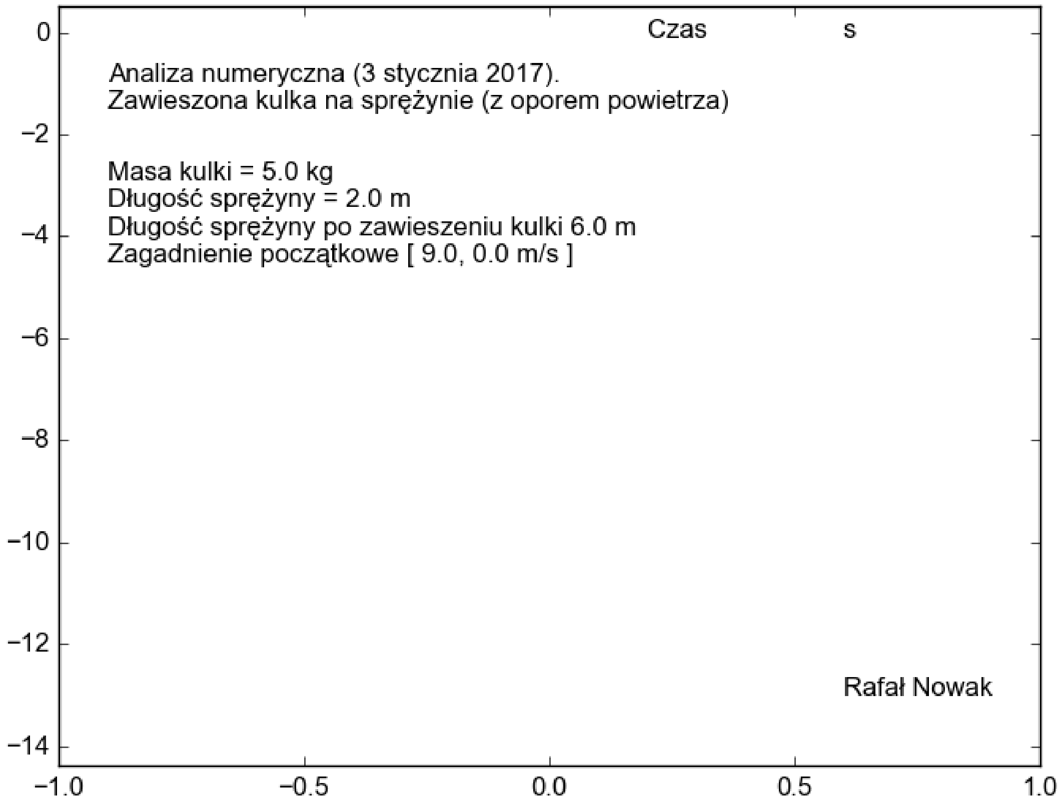
In [16]:
```
L  = 2.0;     # długość sprężyny (stan swobodny)
g  = 9.81;    # przyspieszenie ziemskie

m = 5.0;      # masa kulki

ΔL = 4.0;     # przyrost długości sprężyny po zawieszeniu kulki
k  = m*g/ΔL; # współczynnik sprężystości sprężyny

r = 0.10;     # współczynnik oporu

#### Zagadnienie początkowe
y_0 = [ L+ΔL+3.0, 0.0 ]; # umieszczamy kulkę w pozycji y_0[1] oraz nadajemy jej prędkość y_

# m*y'' + k*(y-L-ΔL) = 0
# y1 = y
# y2 = y'

f(x,y::Vector{Float64}) = [ y[2] , -k*(y[1]-L-ΔL) - r*y[2] ]

function method_step_RK(t, y::Vector{Float64}, h, f::Function)
    K1 = f(t, y)
    K2 = f(t+h/2  , y+h*K1/2)
    K3 = f(t+h/2  , y+h*K2/2)
    K4 = f(t+h    , y+h*K3  )
    return y + h/6*(K1+2*(K2+K3)+K4);
```

```
        end

        method_step = method_step_RK;

        a, b = 0.0,  30.0   #  b-a = czas (w sekundach)
        fps = 25;

        x_0 = a
        N = convert(Int, (b-a)*fps );
        x = linspace(a,b,N);
        global y = zeros(2,N)
        y[:,1] = y_0

        for i=1:N-1
            y[:,i+1] = method_step(x[i],y[:,i],x[i+1]-x[i],f);
            # @printf("y(%6.2f) = %19.16f\n", x[i+1], y[1,i+1]);
        end


        plot( y[1,:] )
```

```
WARNING: Method definition f(Any, Array{Float64, 1}) in module Main at In[15]:18 overwritten at In[1
WARNING: Method definition method_step_RK(Any, Array{Float64, 1}, Any, Function) in module Main at I
```

In [13]:
```
using PyPlot
using PyCall

y[1,:] = -y[1,:];

@pyimport matplotlib as mat_pl
mat_pl.rc("font", family="Arial")

fig = figure();
ax = axes(xlim=(-1, 1), ylim=(-1.6*(y_0[1]), 0.5));

# plot( 0.7, -(L+ΔL), "ko", markersize=5);
text(-0.9, -1.5, """Analiza numeryczna (3 stycznia 2017).
                    Zawieszona kulka na sprężynie (z oporem powietrza)""")
text(-0.9, -4.5, """Masa kulki = $m kg
                    Długość sprężyny = $L m
                    Długość sprężyny po zawieszeniu kulki $(L+ΔL) m
                    Zagadnienie początkowe [ $(y_0[1]), $(y_0[2]) m/s ]""")
text(0.6, -13, "Rafał Nowak")
global line = ax[:plot]([], [], lw=4, alpha=0.2)[1]
global vel  = ax[:plot]([], [], lw=1)[1]
global ball = ax[:plot]([],[], "ro", markersize=10)[1]
text(0.2, -0.1, "Czas ")
global time_text = ax[:text](0.4, -0.1, "")
text(0.6, -0.1, "s")

# initialization function: plot the background of each frame
function init()
    global line, time_text
    line[:set_data]([], [])
    vel[:set_data]([], [])
    time_text[:set_text]("");
    ball[:set_data]([], [])
    return (line,time_text)
```

```
end

# animation function.  This is called sequentially
function animate(i)
    global line, time_text, y
    line[:set_data]([0.7,0.7], [0,y[1,i]])
    vel[:set_data]( [0.7,0.7], [y[1,i],y[1,i]-y[2,i]])
    time_text[:set_text](@sprintf("%.2f",i/fps));
    ball[:set_data]( [0.7], [y[1,i]] )
    return (line,time_text)
end

@pyimport matplotlib.animation as anim

myanim = anim.FuncAnimation(fig, animate, 1:N, init_func=init, interval=1000.0/fps);
# show() # do not work in IJulia
myanim[:save]("/tmp/anim.mp4", extra_args=["-vcodec", "libx264", "-pix_fmt", "yuv420p"],fps
```




```
WARNING: using PyPlot.plot in module Main conflicts with an existing identifier.
```

```
In [14]: display("text/html", string("""<video autoplay controls><source src="data:video/x-m4v;base6
         base64encode(open(read,"/tmp/anim.mp4")),"""" type="video/mp4"></video>"""))
```

```
In [ ]:
```