Differential Equations Coursework
Aeroplane Landing Modelling

Gleb Dianov

Contents

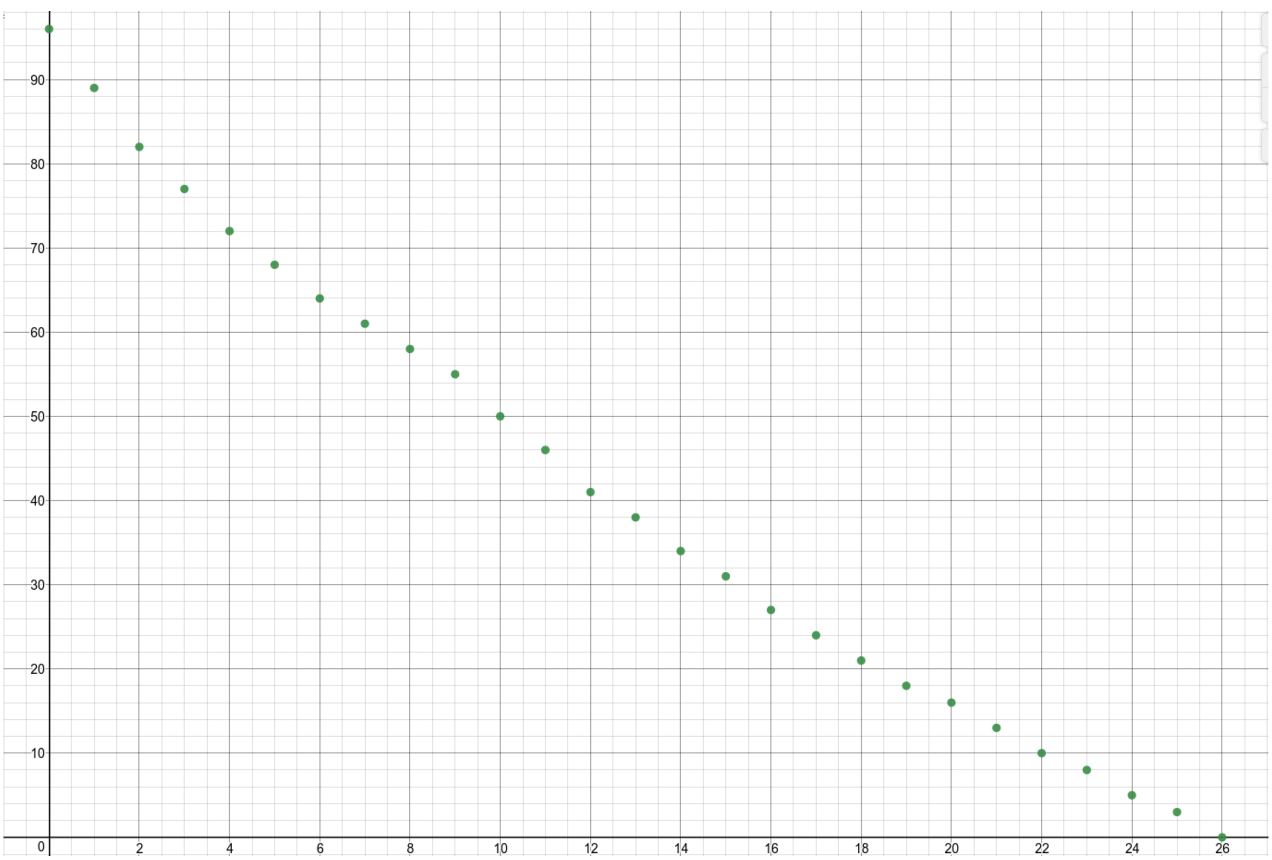
1 Simplifying situation and setting up the model

1.1 Given data

I've been given data about a landing aeroplane after touchdown. The mass of the aeroplane is 120000 kg. Initially air resistance slows the aeroplane and then, when it is slow enough, wheel brakes add a constant force to fully stop the aeroplane. The table below shows the aeroplane's speed, v ms^{-1}, t seconds after touchdown.

t	v	t	v
0	96	14	34
1	89	15	31
2	82	16	27
3	77	17	24
4	72	18	21
5	68	19	18
6	64	20	16
7	61	21	13
8	58	22	10
9	55	23	8
10	50	24	5
11	46	25	3
12	41	26	0
13	38		

The image below shows a chart of the original data.



1.2 Modeling assumptions

- **The aeroplane can be modeled as a point particle**. This assumption allows us to negate any complex resistances that would occur. Modeling this way is appropriate because we are not given velocity (only speed) of the aeroplane at different points in time, so we are not interested in any rotations of the plane.

- **The runway can be modeled as a plane**. Without this assumption we would need to consider the possibility that the runway has a complex shape, for example it can have hills and/or pits.

- **The runway is horizontal**. Without this assumption we would need to include components of the weight and the reaction force in the model which would increase the complexity of the model and affect the calculations.

- **The aeroplane is moving in the horizontal plane**. This assumption allows us to negate any vertical forces.

- **The constant force from the wheel brakes (after the brakes are applied) and the air resistance are the only forces acting on the aeroplane in the horizontal plane**. This assumption allows us to say that any other resistance is negligible and there are no other forces, alternatively we can say that the air resistance model is a model of the total resistance force.

- **The given values for speed are rounded to the nearest integer**. This assumption allows the model of speed to have an error ± 0.5. It's very unlikely that the values of speed are exactly integers, so it's a reasonable assumption.

- **Brakes are applied at $t = 9$**. The table below shows differences between consecutive speed values, Δv, and differences between consecutive differences, $\Delta(\Delta v)$. $\Delta(\Delta v)$ is the lowest when $t = 10$. This means that the highest decrease of acceleration occurred between $t = 9$ and $t = 10$, hence we will assume that the brakes were applied during that time. We will model this as if the brakes were applied at $t = 9$.

t	v	Δ v	Δ (Δ v)	t	v	Δ v	Δ (Δ v)
0	96			14	34	-4	-1
1	89	-7		15	31	-3	1
2	82	-7	0	16	27	-4	-1
3	77	-5	2	17	24	-3	1
4	72	-5	0	18	21	-3	0
5	68	-4	1	19	18	-3	0
6	64	-4	0	20	16	-2	1
7	61	-3	1	21	13	-3	-1
8	58	-3	0	22	10	-3	0
9	55	-3	0	23	8	-2	1
10	50	-5	-2	24	5	-3	-1
11	46	-4	1	25	3	-2	1
12	41	-5	-1	26	0	-3	-1
13	38	-3	2				

- **Braking force starts working instantly**. This assumption makes acceleration not continuous and simplifies the calculations.

- **There is no wind**. This allows us to assume that if the plane is not moving then there is no air resistance.

1.3 Derivation of the initial model

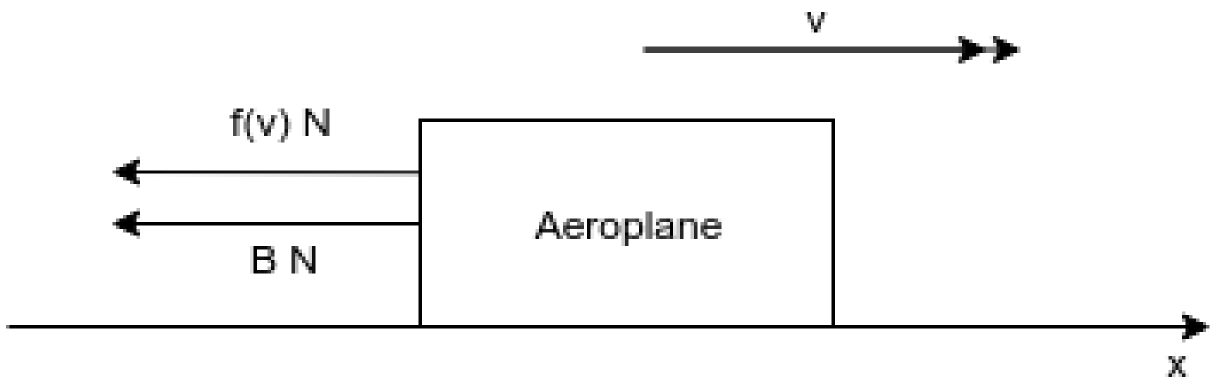


Figure 1: Forces diagram

The diagram shows the x-axis, direction of velocity, v, the air resistance, $f(v)$, and the constant braking force (when brakes are used), B.

$$F = ma \text{ (Newton's second law)}$$

$$F = \begin{cases} -f(v), & 0 \leq t < 9 \\ -f(v) - B, & 9 \leq t \leq 26 \end{cases}$$

$$a = \dot{v}$$

$$\Rightarrow \begin{cases} m\dot{v} = -f(v), & t < 9 \\ m\dot{v} = -f(v) - B, & t \geq 9 \end{cases}$$

$$\text{where } m = 120000$$

2 Model 1

2.1 Defining f(v)

As you can see in the table with the differences, speed for $t \in (0, 9)$ clearly isn't decreasing linearly. And you can see that, as speed decreases (in that interval), the rate at which it decreases also decreases. So it's reasonable to suggest that

$$f(v) \propto v^2$$

$$\Rightarrow f(v) = kv^2$$

$$\Rightarrow m\dot{v} = \begin{cases} -kv^2, & 0 \leq t < 9 \\ -kv^2 - B, & 9 \leq t \leq 26 \end{cases}$$

To make v continuous I will assume that both models are valid for $t = 9$.

$$m\dot{v} = \begin{cases} -kv^2, & 0 \leq t \leq 9 \\ -kv^2 - B, & 9 \leq t \leq 26 \end{cases}$$

2.2 General solution

2.2.1 $0 \leq t \leq 9$

$$m\dot{v} = -kv^2$$
$$\Rightarrow m\frac{\dot{v}}{v^2} = -k$$
$$\Rightarrow m\int \frac{\dot{v}}{v^2}dt = -k\int dt$$
$$\Rightarrow m\int v^{-2}dv = -kt + c_1$$
$$\Rightarrow m\frac{v^{-1}}{-1} = -kt + c_1$$
$$\Rightarrow \frac{-m}{v} = -kt + c_1$$
$$\Rightarrow v = \frac{m}{kt - c_1}$$

2.2.2 $9 \leq t \leq 26$

$$m\dot{v} = -kv^2 - B$$
$$m\frac{\dot{v}}{kv^2 + B} = -1$$
$$m\int \frac{\dot{v}}{kv^2 + B}dt = -\int dt$$
$$m\int \frac{1}{kv^2 + B}dv = -\int dt$$
$$\text{arctan}'(\psi) = \frac{1}{\psi^2 + 1}$$
$$\Rightarrow \text{arctan}'(\frac{a}{b}\psi) = \frac{a}{b} \times \frac{1}{(\frac{a}{b}\psi)^2 + 1} = \frac{a}{b} \times \frac{b^2}{a^2\psi^2 + b^2} = \frac{ab}{a^2\psi^2 + b^2}$$
$$\Rightarrow \text{arctan}'(\sqrt{\frac{a}{b}}\psi) = \frac{\sqrt{ab}}{a\psi^2 + b}$$
$$\Rightarrow \int \frac{1}{kv^2 + B}dv = \frac{1}{\sqrt{kB}}\int \frac{\sqrt{kB}}{kv^2 + B} = \frac{\text{arctan}(\sqrt{\frac{k}{B}}v)}{\sqrt{kB}} + c$$
$$\Rightarrow \frac{m\,\text{arctan}(\sqrt{\frac{k}{B}}v)}{\sqrt{kB}} = -t + c$$
$$\Rightarrow \sqrt{\frac{k}{B}}v = \tan\frac{\sqrt{kB}(-t + c)}{m}$$
$$\Rightarrow v = \sqrt{\frac{B}{k}}\tan\frac{\sqrt{kB}(-t + c)}{m}$$
$$\text{Let } c_2 = \sqrt{kB}c$$

$$\Rightarrow v = \sqrt{\frac{B}{k}} \tan(\frac{c_2 - \sqrt{kB}t}{m})$$

$$\Rightarrow v = \begin{cases} \frac{m}{kt - c_1}, & 0 \leq t \leq 9 \\ \sqrt{\frac{B}{k}} \tan(\frac{c_2 - \sqrt{kB}t}{m}), & 9 \leq t \leq 26 \end{cases}$$

2.3 Particular solution

The general solution has 4 unknown constants: the constant braking force, B, the coefficient from the air resistance model, k, and 2 constants of integration from solving the 2 differential equations, c_1 and c_2. To make v continuous I will use point $(t = 9, v = 55)$.

$$\begin{cases} \frac{m}{9k - c_1} = 55 \\ \sqrt{\frac{B}{k}} \tan \frac{c_2 - 9\sqrt{kB}}{m} = 55 \end{cases}$$

$$\Rightarrow 9k - c_1 = \frac{m}{55}$$

$$\Rightarrow k = \frac{c_1 + \frac{m}{55}}{9}$$

S is the set of given data.

$$(t_1, v_1), (t_2, v_2) \in S; t_1, t_2 \leq 9, t_1 \neq t_2$$

$$\begin{cases} v_1 = \frac{m}{t_1 k - c_1} \\ v_2 = \frac{m}{t_2 k - c_1} \end{cases}$$

$$\Rightarrow \begin{cases} v_1 v_2 t_2 t_1 k - v_1 v_2 t_2 c_1 = m v_2 t_2 \\ v_1 v_2 t_1 t_2 k - v_1 v_2 t_1 c_1 = m v_1 t_1 \end{cases}$$

$$\Rightarrow c_1 v_1 v_2 (t_2 - t_1) = m(v_1 t_1 - v_2 t_2)$$

$$\Rightarrow c_1 = \frac{m(v_1 t_1 - v_2 t_2)}{v_1 v_2 (t_2 - t_1)}$$

$$\sqrt{\frac{B}{k}} \tan \frac{c_2 - 9\sqrt{kB}}{m} = 55$$

$$\Rightarrow c_2(B) = m \arctan(\frac{55k\sqrt{\frac{B}{k}}}{B}) + 9\sqrt{Bk}$$

$$(t_3, v_3) \in S; t_3 > 9$$

$$\Rightarrow v_3 = \sqrt{\frac{B}{k}} \tan \frac{c_2(B) - \sqrt{kB}t_3}{m}$$

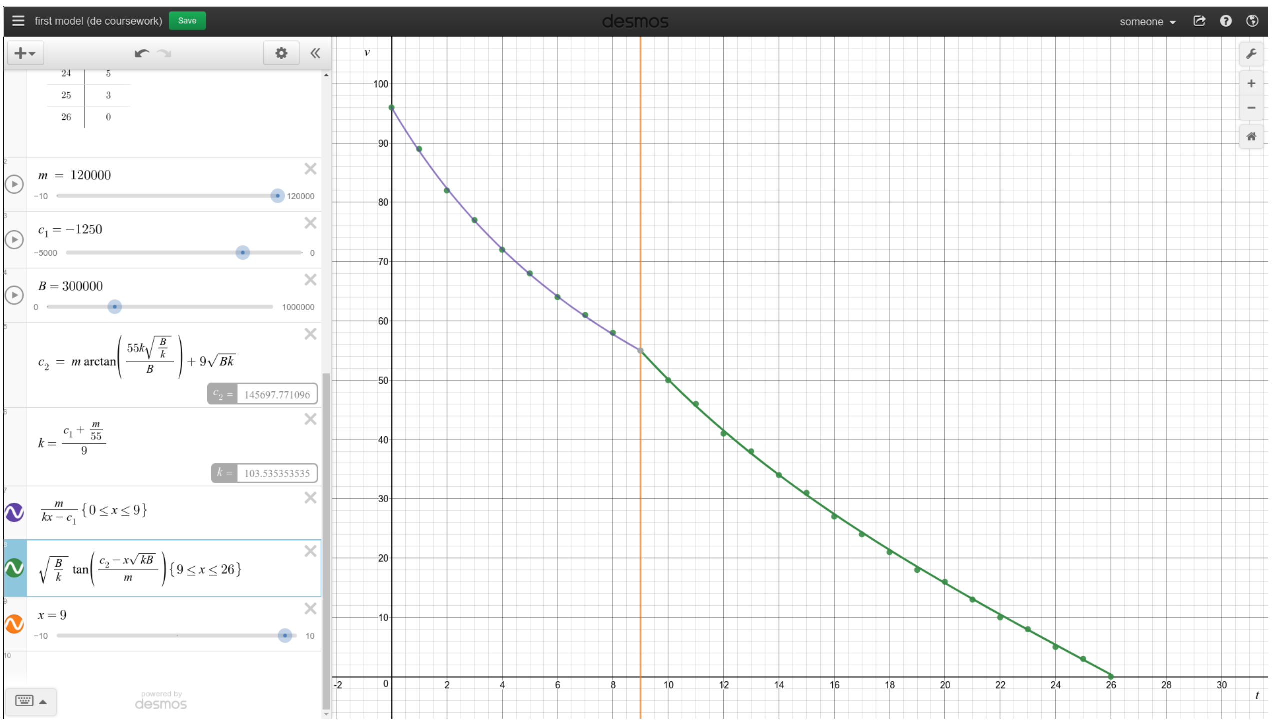
$$\Rightarrow \arctan(v_3 \sqrt{\frac{k}{B}}) = \frac{c_2(B) - \sqrt{kB}t_3}{m}$$

$$\text{Let } \lambda(B) = \frac{c_2(B) - \sqrt{kB}t_3}{m} - \arctan(v_3 \sqrt{\frac{k}{B}})$$

$$\text{Newton-Raphson's rule: } B_{n+1} = B_n + \frac{\lambda'(B_n)}{\lambda(B_n)}$$

$$\begin{cases} c_1 = \frac{m(v_1 t_1 - v_2 t_2)}{v_1 v_2 (t_2 - t_1)} \\ k = \frac{c_1 + \frac{m}{55}}{9} \\ c_2(B) = m \arctan(\frac{55k\sqrt{\frac{B}{k}}}{B}) + 9\sqrt{Bk} \\ \lambda(B) = \frac{c_2(B) - \sqrt{kB} t_3}{m} - \arctan(v_3 \sqrt{\frac{k}{B}}) \\ B_{n+1} = B_n - \frac{\lambda(B_n)}{\lambda'(B_n)} B_0 = 300000 \end{cases} \tag{1}$$

To find B_0 I used a graphing web app. First I put in all the formulas and given data points. $v(0) = \frac{m}{-c_1} > 0$, hence $c_1 < 0$. I started decreasing c_1 until it roughly matched first 10 points. $B > 0$, so for this value of c_1 I started increasing B until I found that $B \approx 300,000$.



Now I can use programming to find the parameters. To do this I wrote 5 modules in Haskell. The first module is a module for automatic differentiation using dual numbers and Newton-Raphson method.

```haskell
module AutoDiff where

import Data.Matrix
import Data.Vector (Vector)
import qualified Data.Vector as V

data Dual a =
    Dual { val  :: a
         , diff :: a
         } deriving (Eq, Show)

constDual :: Num a => a -> Dual a
constDual x = Dual x 0

-- to use we put Dual (f x) (f' x). Eg if using "x", then write Dual x 1, cause dx/dx = 1
```

```haskell
instance Num a => Num (Dual a) where
    (Dual u u') + (Dual v v') = Dual (u + v) (u' + v')
    (Dual u u') - (Dual v v') = Dual (u - v) (u' - v')
    (Dual u u') * (Dual v v') = Dual (u * v) (u' * v + u * v')
    abs (Dual u u') = Dual (abs u) (u' * signum u)
    signum (Dual u u') = Dual (signum u) 0
    fromInteger n = Dual (fromInteger n) 0

instance Fractional a => Fractional (Dual a) where
    (Dual u u') / (Dual v v') = Dual (u / v) ((u' * v - u * v') / v^2)
    fromRational q = Dual (fromRational q) 0

instance (Eq a, Floating a) => Floating (Dual a) where
    pi                = Dual pi        0
    exp   (Dual u u') = Dual (exp u)   (u' * exp u)
    log   (Dual u u') = Dual (log u)   (u' / u)
    sqrt  (Dual u u') = Dual (sqrt u)  (u' / (2 * sqrt u))
    sin   (Dual u u') = Dual (sin u)   (u' * cos u)
    cos   (Dual u u') = Dual (cos u)   (-1 * u' * sin u)
    tan   (Dual u u') = Dual (tan u)   (u' / (cos u ** 2))
    asin  (Dual u u') = Dual (asin u)  (u' / sqrt(1 - (u ** 2)))
    acos  (Dual u u') = Dual (acos u)  ((- 1) * u' / sqrt(1 - (u ** 2)))
    atan  (Dual u u') = Dual (atan u)  (u' / (1 + (u ** 2)))
    sinh  (Dual u u') = Dual (sinh u)  (u' * cosh u)
    cosh  (Dual u u') = Dual (cosh u)  (u' * sinh u)
    tanh  (Dual u u') = Dual (tanh u)  (u' * (1 - (tanh u ** 2)))
    asinh (Dual u u') = Dual (asinh u) (u' / sqrt(1 + (u ** 2)))
    acosh (Dual u u') = Dual (acosh u) (u' / (sqrt((u ** 2) - 1)))
    atanh (Dual u u') = Dual (atanh u) (u' / (1 - (u ** 2)))
    (Dual u u') ** (Dual n 0)       = Dual (u ** n) (u' * n * u ** (n - 1))
    (Dual a 0)  ** (Dual v v')      = Dual (a ** v) (v' * log a * a ** v)
    (Dual u u') ** (Dual v v')      = Dual (u ** v) ((u ** v) * (v' * log u + (v * u' / u)))
    logBase (Dual u u') (Dual v v') =
        Dual (logBase u v) ((log v * u' / u - log u * v' / v) / (log u ** 2))

instance Ord a => Ord (Dual a) where
    (Dual x _) <= (Dual y _ ) = x <= y

instance (Enum a, Num a) => Enum (Dual a) where
    toEnum n = constDual $ toEnum n
    fromEnum (Dual x _) = fromEnum x

d :: (Num a, Num c) => (Dual a -> Dual c) -> a -> c
d f x = diff . f $ Dual x 1

toNormalF :: (Num a, Num b) => (Dual a -> Dual b) -> a -> b
toNormalF f = val . f . constDual
```

```haskell
newton :: (Fractional a, Ord a) => (Dual a -> Dual a) -> Either Integer (a -> Bool) -> a -> a
newton f stop y = newtonHelper stop y y
  where newtonHelper stopCond minX x
          | either (== 0) ($ x) stopCond = if minCheck then x else minX
          | otherwise = newtonHelper (either (Left . (\n -> n - 1)) (const stopCond) stopCond)
                                     (if minCheck then x else minX)
                                     nextX
          where nextX = x - toNormalF f x / d f x
                minCheck = abs (toNormalF f x) < abs (toNormalF f minX)
```

The second module contains the given data.

```haskell
{-# LANGUAGE RecordWildCards #-}

module GivenData where

import           AutoDiff
import           Data.Vector (Vector)
import qualified Data.Vector as V

vs :: Num a => [a]
vs = [96,89,82,77,72,68,64,61,58,55,50,46,41,38,34,31,27,24,21,18,16,13,10,8,5,3,0]

data Point a =
    Point { pTime  :: a
          , pSpeed :: a
          } deriving (Show, Eq)

points :: (Num a, Enum a) => [Point a]
points = zipWith Point [0..26] vs

vectorPoints :: (Num a, Enum a) => Vector (Point a)
vectorPoints = V.fromList points

m :: Num a => a
m = 120000

toDualPoint :: Num a => Point a -> Point (Dual a)
toDualPoint Point{..} = Point (constDual pTime) (constDual pSpeed)
```

Then I defined a general interface for models.

```haskell
{-# LANGUAGE TypeFamilies #-}

module Solution.Model where

import           Data.Proxy

class ModelParams f where
    type SolutionGen f :: * -> *
    predict :: f Double -> Double -> Double
    findParams :: (Floating a, Ord a, Enum a) => SolutionGen f a -> f a
    combinations :: (Num a, Enum a, Eq a) => Proxy f -> [SolutionGen f a]
```

The fourth module is for the first model and finding parameters.

```haskell
{-# LANGUAGE RecordWildCards #-}
{-# LANGUAGE TypeFamilies    #-}

module Solution.First (Parameters, SolutionGen) where

import           Data.Proxy

import           AutoDiff
import           GivenData
import qualified Solution.Model as M


data Parameters a =
    Parameters { pC1  :: a
               , pK   :: a
               , pC2  :: a
               , pB   :: a
               , pGen :: SolutionGen a
               }


data SolutionGen a =
    SolutionGen { sgPoint1 :: Point a
                , sgPoint2 :: Point a
                , sgPoint3 :: Point a
                }


instance Show a => Show (SolutionGen a) where
    show SolutionGen{..} =
         ", p1 = (" ++ show (pTime sgPoint1) ++ ", " ++ show (pSpeed sgPoint1) ++ ")"
      ++ ", p2 = (" ++ show (pTime sgPoint2) ++ ", " ++ show (pSpeed sgPoint2) ++ ")"
      ++ ", p3 = (" ++ show (pTime sgPoint3) ++ ", " ++ show (pSpeed sgPoint3) ++ ")"
```

```haskell
instance Show a => Show (Parameters a) where
    show Parameters{..} =   "c1 = " ++ show pC1
                       ++  ", k = " ++ show pK
                       ++ ", c2 = " ++ show pC2
                       ++  ", b = " ++ show pB
                       ++ show pGen
                       ++ "\n"


c1F :: Fractional a => Point a -> Point a -> a
c1F (Point t1 v1) (Point t2 v2) = m * (v1 * t1 - v2 * t2) / (v1 * v2 * (t2 - t1))

kF :: Fractional a => a -> a
kF c1 = (c1 + m / 55) / 9

c2F :: Floating a => a -> a -> a
c2F k b = m * atan (55 * k * sqrt (b / k) / b) + 9 * sqrt (b * k)

lambda :: Floating a => Point a -> a -> a -> a
lambda (Point t3 v3) k b = (c2F k b - sqrt (k * b) * t3) / m - atan (v3 * sqrt (k / b))

findParams :: (Floating a, Ord a) => SolutionGen a -> Parameters a
findParams gen@SolutionGen{..} = parameters
  where parameters =
          Parameters { pC1  = c1F sgPoint1 sgPoint2
                     , pK   = kF $ pC1 parameters
                     , pC2 = c2F (pK parameters) (pB parameters)
                     , pB   = newton (lambda (toDualPoint sgPoint3) (constDual $ pK parameters))
                                  (Right $ (<= 0.0001) . abs . lambda sgPoint3 (pK parameters))
                                  300000
                     , pGen = gen
                     }

predict :: Parameters Double -> Double -> Double
predict Parameters{..} t
  | 0 <= t && t <= 9  = m / (pK * t - pC1)
  | 9 <  t && t <= 26 = sqrt (pB / pK) * tan ((pC2 - sqrt (pK * pB) * t) / m)
  | otherwise = error "Invalid time!"

combinations :: (Enum a, Num a, Eq a) => [SolutionGen a]
combinations = zipWith (flip $ uncurry SolutionGen) (drop 10 points)
             $ filter (uncurry (/=))
             $ (\l -> [ (p1, p2) | p1 <- l, p2 <- l ])
             $ take 10 points

instance M.ModelParams Parameters where
    type SolutionGen Parameters = SolutionGen
    predict = predict
    findParams = findParams
    combinations Proxy = combinations
```

And finally the main module, which runs this code to find the parameters.

```haskell
import           Data.List      (sort, sortOn)
import           Data.Proxy

import           GivenData
import qualified Solution.First as Sol
import           Solution.Model as Model

adjustLength :: (a -> String) -> (a -> String -> b) -> [a] -> [b]
adjustLength g s l =
  (\x -> let str = g x in s x $ str ++ replicate (longest - length str) ' ') <$> l
  where longest = maximum $ length . g <$> l

showPredictions :: (ModelParams f, Show (f Double)) => f Double -> String
showPredictions params =
  foldMap (\(x, y, z, w) -> "| " ++ x ++ " | " ++ y ++ " | " ++ z ++ " | " ++ w ++ " |\n")
     $  adjustLength (\(_, _, _, w) -> w) (\(x, y, z, _) w -> (x, y, z, w))
     $  adjustLength (\(_, _, z, _) -> z) (\(x, y, _, w) z -> (x, y, z, w))
     $  adjustLength (\(_, y, _, _) -> y) (\(x, _, z, w) y -> (x, y, z, w))
     $  adjustLength (\(x, _, _, _) -> x) (\(_, y, z, w) x -> (x, y, z, w))
     $  (("Time", "Predicted Speed", "Rounded predicted speed", "Observed Speed"):)
     $  (\(Point t v) -> let v' = predict params t in (show t, show v', rounded v', rounded v))
    <$> points
  where rounded = show . round

maxErr :: (a -> Double -> Double) -> a -> Double
maxErr predict params =
  maximum $ abs . (\(Point t v) -> predict params t - v) <$> points
  where n = fromIntegral (length points)

avrErr :: (a -> Double -> Double) -> a -> Double
avrErr predict params =
  (/ n) $ sum $ abs . (\(Point t v) -> predict params t - v) <$> points
  where n = fromIntegral (length points)

rms :: (a -> Double -> Double) -> a -> Double
rms predict params =
  (/ n) $ sqrt $ sum $ (^2) . (\(Point t v) -> predict params t - v) <$> points
  where n = fromIntegral (length points)

showErrors :: (ModelParams f, Show (f Double)) => f Double -> String
showErrors params =
  foldMap (\(x, y, z) -> "| " ++ x ++ " | " ++ y ++ " | " ++ z ++ " |\n")
  $  adjustLength (\(_, _, z) -> z) (\(x, y, _) z -> (x, y, z))
  $  adjustLength (\(_, y, _) -> y) (\(x, _, z) y -> (x, y, z))
  $  adjustLength (\(x, _, _) -> x) (\(_, y, z) x -> (x, y, z))
  [ ("Maximum Error", "Average Error", "Root mean square")
  , (show $ maxErr predict params, show $ avrErr predict params, show $ rms predict params)
  ]
```

```haskell
showResults :: (ModelParams f, Show (f Double)) => f Double -> String
showResults params = show params ++ "\n"
                  ++ showErrors params ++ "\n"
                  ++ showPredictions params

results :: [Sol.Parameters Double]
results =  sortOn (rms predict) $ findParams
      <$> combinations (Proxy :: Proxy Sol.Parameters)

main :: IO ()
main = do putStrLn "The best result:"
          putStrLn $ showResults $ head results
          putStrLn "The worst result:"
          putStr   $ showResults $ last results
```

Here is the result. It shows the best and the worst parameters found using the derived formulas and for both of the configurations it shows sums of squares of errors, and predictions for each of the given point with rounded versions.

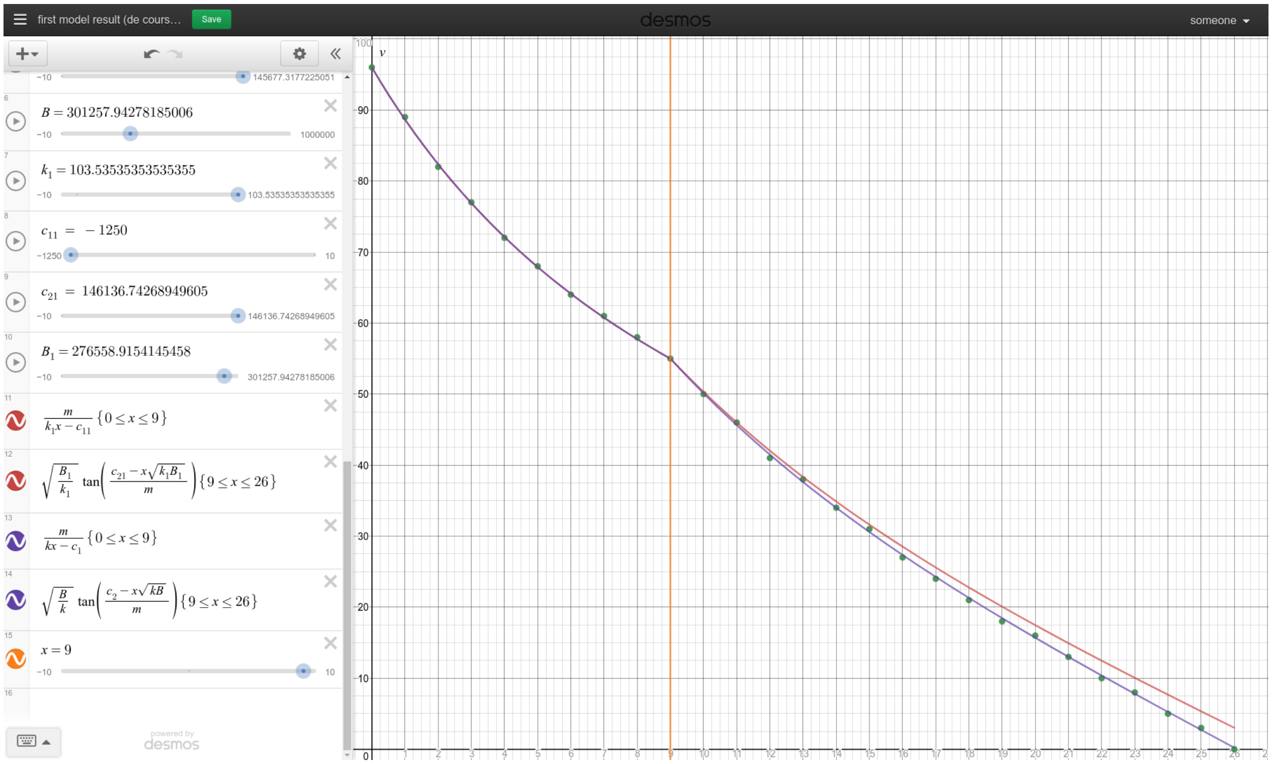
```
optimizer : bash — Konsole
λ.gleb::home-arch ⇒ optimizer → stack exec optimizer-exe
The best result:
c1 = -1250.0, k = 103.53535353535355, c2 = 145677.3177225051, b = 301257.94278185006, p1 = (0.0, 96.0), p2 = (5.0, 68.0), p3 = (14.0, 34.0)

| Maximum Error       | Average Error       | Root mean square   |
| 0.47598886269637575 | 0.23822424753496707 | 0.2776875104911393 |

| Time | Predicted Speed    | Rounded predicted speed | Observed Speed |
| 0.0  | 96.0               | 96                      | 96             |
| 1.0  | 88.65671641791045  | 89                      | 89             |
| 2.0  | 82.35701906412478  | 82                      | 82             |
| 3.0  | 76.89320388349515  | 77                      | 77             |
| 4.0  | 72.1092564491654   | 72                      | 72             |
| 5.0  | 67.88571428571429  | 68                      | 68             |
| 6.0  | 64.12955465587045  | 64                      | 64             |
| 7.0  | 60.767263427109974 | 61                      | 61             |
| 8.0  | 57.739975698663415 | 58                      | 58             |
| 9.0  | 54.99999999999999  | 55                      | 55             |
| 10.0 | 50.10816658852111  | 50                      | 50             |
| 11.0 | 45.622063036212374 | 46                      | 46             |
| 12.0 | 41.475988862696376 | 41                      | 41             |
| 13.0 | 37.61659930730753  | 38                      | 38             |
| 14.0 | 34.00002062308155  | 34                      | 34             |
| 15.0 | 30.589724513917773 | 31                      | 31             |
| 16.0 | 27.35493651754378  | 27                      | 27             |
| 17.0 | 24.269426053394415 | 24                      | 24             |
| 18.0 | 21.3105731783461   | 21                      | 21             |
| 19.0 | 18.45863841578133  | 18                      | 18             |
| 20.0 | 15.696183121094782 | 16                      | 16             |
| 21.0 | 13.00760227741885  | 13                      | 13             |
| 22.0 | 10.378741615250261 | 10                      | 10             |
| 23.0 | 7.796577949697343  | 8                       | 8              |
| 24.0 | 5.248946558956161  | 5                       | 5              |
| 25.0 | 2.7243028986249413 | 3                       | 3              |
| 26.0 | 0.2115083630415522 | 0                       | 0              |

The worst result:
c1 = -1250.0, k = 103.53535353535355, c2 = 146136.74268949605, b = 276558.9154145458, p1 = (0.0, 96.0), p2 = (2.0, 82.0), p3 = (11.0, 46.0)

| Maximum Error      | Average Error      | Root mean square   |
| 3.0224756774093864 | 1.0279882781712646 | 1.4042237215181517 |
```

2.4 Graphs and Analysis

Time	Predicted Speed	Rounded Predicted Speed	Observed Speed
0.0	96.0	96	96
1.0	88.65671641791045	89	89
2.0	82.35701906412478	82	82
3.0	76.89320388349515	77	77
4.0	72.1092564491654	72	72
5.0	67.88571428571429	68	68
6.0	64.12955465587045	64	64
7.0	60.767263427109974	61	61
8.0	57.739975698663415	58	58
9.0	54.99999999999999	55	55
10.0	50.10816658852111	50	50
11.0	45.622063036212374	46	46
12.0	41.475988862696376	41	41
13.0	37.61659930730753	38	38
14.0	34.00002062308155	34	34
15.0	30.589724513917773	31	31
16.0	27.35493651754378	27	27
17.0	24.269426053394415	24	24
18.0	21.3105731783461	21	21
19.0	18.45863841578133	18	18
20.0	15.696183121094782	16	16
21.0	13.00760227741885	13	13
22.0	10.378741615250261	10	10
23.0	7.796577949697343	8	8
24.0	5.248946558956161	5	5
25.0	2.7243028986249413	3	3
26.0	0.2115083630415522	0	0

Maximum Error	Average Error	Root mean square
0.47598886269637575	0.23822424753496707	0.2776875104911393

I'll now look at the two configurations that I obtained and plot them. The red line shows the worst configuration, and the purple line shows the best configuration.



As you can see in 2.3 rounded predictions of the best configuration exactly match the given data.

2.5 Conclusion of Model 1

$$\begin{cases} v(t) = \frac{m}{kt - c_1}, & 0 \le t \le 9 \\ v(t) = \sqrt{\frac{B}{k}} \tan(\frac{c_2 - \sqrt{kB}t}{m}), & 9 \le t \le 26 \\ c_1 = -1250.0 \\ k = 103.\dot{5}\dot{3} \\ c_2 = 145677.3177225051 \\ B = 301257.94278185006 \end{cases} \tag{2}$$

$$x = \int_0^{26} v dt = \int_0^9 v dt + \int_9^{26} v dt$$

$$\int \frac{m}{kt - c_1} dt = \frac{m}{k} \int \frac{k}{kt - c_1} dt = \frac{m}{k} \ln(kt - c_1) + \text{constant}$$

$$\int \sqrt{\frac{B}{k}} \tan(\frac{c_2 - \sqrt{kB}t}{m}) dt = \sqrt{\frac{B}{k}} \times \frac{-m}{\sqrt{kB}} \int \tan(\frac{c_2 - \sqrt{kB}t}{m}) \times \frac{-\sqrt{kB}}{m} dt =$$

$$= \frac{m}{k} \int \frac{-\sin(u)}{\cos u} du = \frac{m}{k} \ln(\cos(\frac{c_2 - \sqrt{kB}t}{m})) + \text{constant}$$

$$\Rightarrow x = \left[\frac{m}{k} \ln(kt - c_1)\right]_0^9 + \left[\frac{m}{k} \ln(\cos(\frac{c_2 - \sqrt{kB}t}{m}))\right]_9^{26} = \frac{m}{k} \left(\ln(\frac{c_1 - 9k}{c_1}) + \ln(\frac{\cos(\frac{c_2 - 26\sqrt{kB}}{m})}{\cos(\frac{c_2 - 9\sqrt{kB}}{m})}) \right)$$

$$\Rightarrow x = \frac{m}{k} \ln \left(\frac{(c_1 - 9k)\cos(\frac{c_2 - 26\sqrt{kB}}{m})}{c_1 \cos(\frac{c_2 - 9\sqrt{kB}}{m})} \right)$$

```
λ.gleb::home-arch ⇒ optimizer → ghci
optimizer-0.1.0.0: initial-build-steps (lib + exe)
The following GHC options are incompatible with GHCi and have not been passed to it: -threaded -O2
Configuring GHCi with the following packages: optimizer
Using main module: 1. Package `optimizer' component exe:optimizer-exe with main-is file: /media/storage/drive/maths/de_coursework/optimizer/app/Main.hs
GHCi, version 8.2.2: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/gleb/.ghci
[1 of 5] Compiling AutoDiff         ( /media/storage/drive/maths/de_coursework/optimizer/src/AutoDiff.hs, interpreted )
[2 of 5] Compiling GivenData        ( /media/storage/drive/maths/de_coursework/optimizer/src/GivenData.hs, interpreted )
[3 of 5] Compiling FirstModel       ( /media/storage/drive/maths/de_coursework/optimizer/src/FirstModel.hs, interpreted )
[4 of 5] Compiling Main             ( /media/storage/drive/maths/de_coursework/optimizer/app/Main.hs, interpreted )
[5 of 5] Compiling SecondModel      ( /media/storage/drive/maths/de_coursework/optimizer/src/SecondModel.hs, interpreted )
Ok, five modules loaded.
Loaded GHCi configuration from /tmp/ghci1576/ghci-script
⇝ parameters = head params
⇝ :set -XRecordWildCards
⇝ (\Model.Parameters{..} → (m / k) * log (((c1 - 9 * k) * cos ((c2 - 26 * sqrt (k * b)) / m)) / (c1 * cos ((c2 - 9 * sqrt ( k * b )) / m)))) parameters
1058.640842314513
```

$$\Rightarrow x \approx 1058.650842313513 \approx 1060$$

The aeroplane fully stops in approximately 1060 meters after touchdown, hence the runway has to be at least 1060 meters long. However, I would advise length of 2000 to be safe even if conditions change, for example in case touchdown occurs further, there is wind, friction between the wheels and the runway is lower.

3 Model 2

3.1 Redefining f(v)

Although the first model fits the given data very well, I think this model can be improved. This time I will add a linear term to the air resistance function.

$$f(v) = kv^2 + \lambda v$$

$$\Rightarrow m\dot{v} = \begin{cases} -kv^2 - \lambda v, & 0 \leq t \leq 9 \\ -kv^2 - \lambda v - B, & 9 \leq t \leq 26 \end{cases}$$

3.2 General solution

3.2.1 $0 \leq t \leq 9$

$$m\dot{v} = -kv^2 - \lambda v$$

$$\Rightarrow m \int \frac{1}{kv^2 + \lambda v} dv = -t + c_3$$

$$kv^2 + \lambda v \equiv v(kv + \lambda)$$

$$\frac{1}{v(kv+\lambda)} \equiv \frac{A}{v} + \frac{B}{kv + \lambda}$$

$$\Rightarrow 1 \equiv A(kv + \lambda) + Bv$$

$$\Rightarrow \begin{cases} A = \frac{1}{\lambda} \\ B = -\frac{k}{\lambda} \end{cases}$$

$$\Rightarrow \frac{1}{kv^2 + \lambda v} \equiv \frac{1}{\lambda v} - \frac{k}{k\lambda v + \lambda^2} (\equiv \frac{k\lambda v + \lambda^2 - k\lambda v}{\lambda v(k\lambda v + \lambda^2)} \equiv \frac{\lambda^2}{\lambda^2(kv^2 + \lambda v)} \equiv \frac{1}{kv^2 + \lambda v})$$

$$\Rightarrow \int \frac{1}{kv^2 - \lambda v} dv = \frac{1}{\lambda} \int v^{-1} dv - \frac{1}{\lambda} \int \frac{k}{kv + \lambda} dv$$

$$\Rightarrow \frac{m}{\lambda}(\ln(v) - \ln(kv + \lambda)) = c_3 - t$$

$$\Rightarrow \ln(\frac{v}{kv+\lambda}) = \frac{\lambda}{m}(c_3 - t)$$
$$\Rightarrow \frac{v}{kv+\lambda} = e^{\frac{\lambda}{m}(c_3-t)}$$
$$\Rightarrow v = ke^{\frac{\lambda}{m}(c_3-t)}v + \lambda e^{\frac{\lambda}{m}(c_3-t)}$$
$$v = \frac{\lambda e^{\frac{\lambda}{m}(c_3-t)}}{1 - ke^{\frac{\lambda}{m}(c_3-t)}}$$
$$v = \frac{\lambda e^{\frac{\lambda}{m}c_3}}{e^{\frac{\lambda}{m}t} - ke^{\frac{\lambda}{m}c_3}}$$

3.2.2 $\mathbf{9 \leq t \leq 26}$

$$m\dot{v} = -kv^2 - \lambda v - W$$
$$\Rightarrow m\int \frac{1}{kv^2+\lambda v+W}dv = -\int dt$$
$$kv^2 + \lambda v + W = (\sqrt{k}v + \frac{\lambda}{2\sqrt{k}})^2 + W - \frac{\lambda^2}{4k} = (W - \frac{\lambda^2}{4k})((\frac{\sqrt{k}v + \frac{\lambda}{2\sqrt{k}}}{\sqrt{W - \frac{\lambda^2}{4k}}})^2 + 1) = (W - \frac{\lambda^2}{4k})((\frac{2kv+\lambda}{\sqrt{4kW-\lambda^2}})^2 + 1)$$
$$\text{Let } h(x) = \frac{2kx+\lambda}{\sqrt{4kW-\lambda^2}}$$
$$\Rightarrow h'(x) = \frac{2k}{\sqrt{4kW-\lambda^2}}$$
$$\arctan'(x) = \frac{1}{1+x^2}$$
$$\Rightarrow (\arctan(h(x)))' = \arctan'(h(x))h'(x) = \frac{1}{h^2(x)+1}h'(x) = \frac{\frac{2k}{\sqrt{4kW-\lambda^2}}}{(\frac{2kx+\lambda}{\sqrt{4kW-\lambda^2}})^2+1}$$
$$\Rightarrow m\int \frac{1}{kv^2+\lambda v+W}dv = m\int \frac{1}{(W-\frac{\lambda^2}{4k})((\frac{2kv+\lambda}{\sqrt{4kW-\lambda^2}})^2+1)}dv = \frac{2m\sqrt{4kW-\lambda^2}}{4k(W-\frac{\lambda^2}{4k})}\int \frac{\frac{2k}{\sqrt{4kW-\lambda^2}}}{(\frac{2kv+\lambda}{\sqrt{4kW-\lambda^2}})^2+1}dv$$
$$\Rightarrow \frac{2m}{\sqrt{4kW-\lambda^2}}\arctan(\frac{2kv+\lambda}{\sqrt{4kW-\lambda^2}}) = -t + c_4$$
$$\Rightarrow \frac{2kv+\lambda}{\sqrt{4kW-\lambda^2}} = \tan(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4 - t))$$
$$\Rightarrow v = \frac{\sqrt{4kW-\lambda^2}\tan(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-t)) - \lambda}{2k}$$
$$\Rightarrow v = \begin{cases} \frac{\lambda e^{\frac{\lambda}{m}c_3}}{e^{\frac{\lambda}{m}t} - ke^{\frac{\lambda}{m}c_3}}, & t \in [0,9] \\ \frac{\sqrt{4kW-\lambda^2}\tan(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-t))-\lambda}{2k}, & t \in [9,26] \end{cases}$$

3.3 Particular solution

$$\Rightarrow \begin{cases} \frac{\lambda e^{\frac{\lambda}{m}c_3}}{e^{9\frac{\lambda}{m}} - ke^{\frac{\lambda}{m}c_3}} = 55 \\ \frac{\sqrt{4kW-\lambda^2}\tan(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-9))-\lambda}{2k} = 55 \end{cases}$$

$$\text{Let } (t_1, v_1), (t_2, v_2) \in S; t_1, t_2 \in [0, 9]; t_1 \neq t_2$$

$$\frac{\lambda e^{\frac{\lambda}{m}c_3}}{e^{t_1\frac{\lambda}{m}} - ke^{\frac{\lambda}{m}c_3}} = v_1$$

$$\Rightarrow v_1 k e^{\frac{\lambda}{m}c_3} = v_1 e^{t_1\frac{\lambda}{m}} - \lambda e^{\frac{\lambda}{m}c_3}$$

$$\Rightarrow k = \frac{v_1 e^{t_1\frac{\lambda}{m}} - \lambda e^{\frac{\lambda}{m}c_3}}{v_1 e^{\frac{\lambda}{m}c_3}}$$

$$\Rightarrow k = e^{\frac{\lambda}{m}(t_1 - c_3)} - \frac{\lambda}{v_1}$$

$$\frac{m}{\lambda}(\ln(v_2) - \ln(kv_2 + \lambda)) = c_3 - t_2$$

$$\Rightarrow c_3 = \frac{m}{\lambda}(\ln(v_2) - \ln(kv_2 + \lambda)) + t_2$$

$$\Rightarrow c_3 = \frac{m}{\lambda}(\ln(\frac{v_2}{v_2(e^{\frac{\lambda}{m}(t_1-c_3)} - \frac{\lambda}{v_1}) + \lambda})) + t_2$$

$$\Rightarrow e^{\frac{\lambda(c_3-t_2)}{m}} = \frac{v_2}{v_2(e^{\frac{\lambda}{m}(t_1-c_3)} - \frac{\lambda}{v_1}) + \lambda}$$

$$\Rightarrow e^{\frac{\lambda}{m}c_3} e^{-\frac{\lambda}{m}t_2} = \frac{v_2}{v_2(e^{\frac{\lambda}{m}t_1} e^{-\frac{\lambda}{m}c_3} - \frac{\lambda}{v_1}) + \lambda}$$

$$\Rightarrow (v_2 e^{\frac{\lambda}{m}t_1} e^{-\frac{\lambda}{m}c_3} - v_2\frac{\lambda}{v_1} + \lambda) e^{\frac{\lambda}{m}c_3} e^{-\frac{\lambda}{m}t_2} = v_2$$

$$\Rightarrow v_2 e^{\frac{\lambda}{m}(t_1-t_2)} - (v_2\frac{\lambda}{v_1} - \lambda) e^{\frac{\lambda}{m}(c_3-t_2)} = v_2$$

$$\Rightarrow e^{\frac{\lambda}{m}(c_3-t_2)} = \frac{v_1 v_2}{\lambda(v_2 - v_1)}(e^{\frac{\lambda}{m}(t_1-t_2)} - 1)$$

$$\Rightarrow e^{\frac{\lambda}{m}c_3} = \frac{v_1 v_2}{\lambda(v_2 - v_1)}(e^{\frac{\lambda}{m}t_1} - e^{\frac{\lambda}{m}t_2})$$

$$\Rightarrow c_3 = \frac{m}{\lambda}\ln(\frac{v_1 v_2(e^{\frac{\lambda}{m}t_1} - e^{\frac{\lambda}{m}t_2})}{\lambda(v_2 - v_1)})$$

$$\Rightarrow k = e^{\frac{\lambda}{m}(9-c_3)} - \frac{\lambda}{55} \text{ (to make } v \text{ continuous)}$$

$$\hat{v} = \frac{\lambda e^{\frac{\lambda}{m}c_3}}{e^{\frac{\lambda}{m}t} - ke^{\frac{\lambda}{m}c_3}}$$

$$J = \sum_{(t,v)\in S, t\leq 9} (\hat{v} - v)^2$$

J is a sum of squares of all errors for $t \leq 9$. We want to minimize this sum, so I will set its partial derivative to zero to find the value of λ that minimizes J.

$$\text{Let } \zeta(\lambda) = \frac{\partial J}{\partial \lambda} = 0$$

$$\Rightarrow \zeta(\lambda) = \partial_\lambda \sum_{(t,v)\in S, t\leq 9} (\hat{v} - v)^2$$

$$\lambda_{n+1} = \lambda_n - \frac{\zeta(\lambda)}{\zeta'(\lambda)} \text{ (Newton-Raphson method)}$$

$$\frac{\sqrt{4kW-\lambda^2}\tan(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-9)) - \lambda}{2k} = 55$$

$$\Rightarrow \sqrt{4kW-\lambda^2}\tan(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-9)) = 2k55 + \lambda$$

$$\Rightarrow \frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-9) = \arctan(\frac{2k55+\lambda}{\sqrt{4kW-\lambda^2}})$$

$$\Rightarrow c_4 = 9 + \frac{2m\arctan(\frac{2k55+\lambda}{\sqrt{4kW-\lambda^2}})}{\sqrt{4kW-\lambda^2}}$$

$$\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-t) = \arctan(\frac{2kv+\lambda}{\sqrt{4kW-\lambda^2}})$$

$$\Rightarrow \frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-t) - \arctan(\frac{2kv+\lambda}{\sqrt{4kW-\lambda^2}}) = 0$$

$$I = \sum_{(t,v)\in S, t\geq 9}\left(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-t) - \arctan(\frac{2kv+\lambda}{\sqrt{4kW-\lambda^2}})\right)^2$$

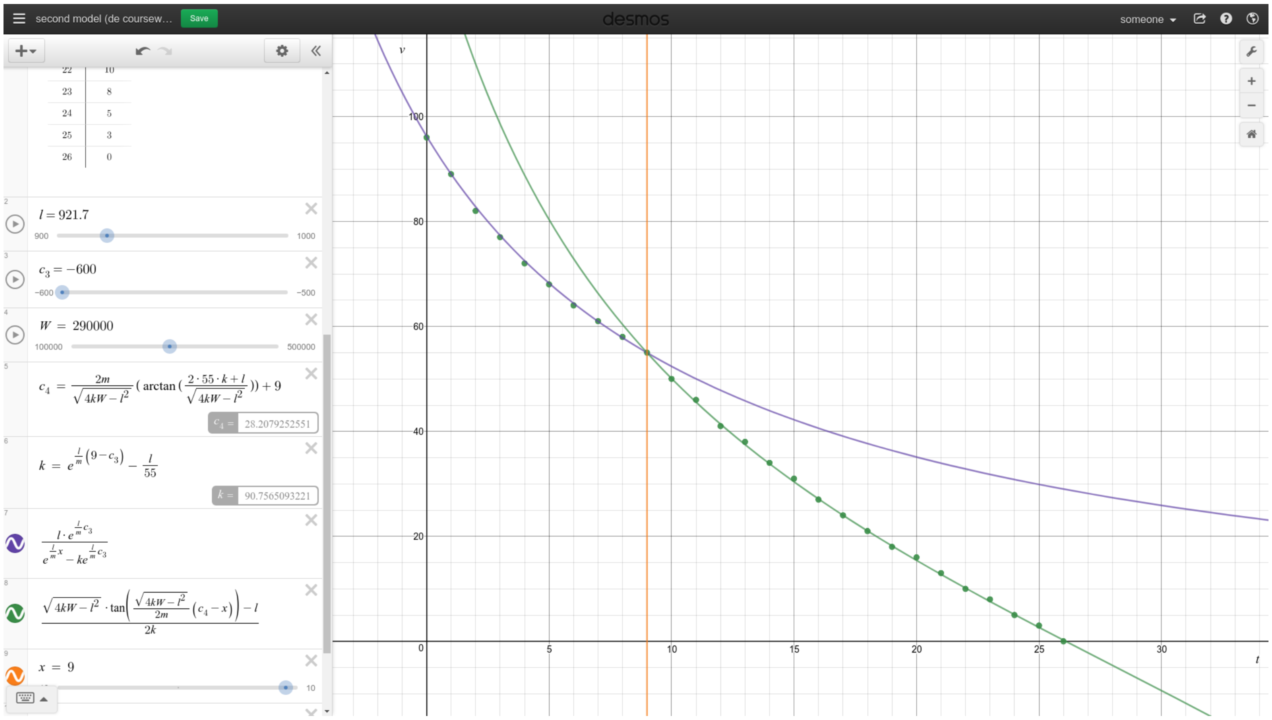
We want to minimize I.

$$\text{Let } \phi(W) = \frac{\partial I}{\partial W} = 0$$

$$\Rightarrow \phi(W) = \partial_W \sum_{(t,v)\in S, t\geq 9}\left(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-t) - \arctan(\frac{2kv+\lambda}{\sqrt{4kW-\lambda^2}})\right)^2$$

$$\Rightarrow W_{n+1} = W_n - \frac{\phi(W)}{\phi'(W)}$$

I found λ_0 and W_0 the same way I found B_0 for the first model - I made a graph and adjusted parameters c_3, λ, and W until the curve roughly fit the data.



$$
\begin{cases}
c_3 = \frac{m}{\lambda}\ln\left(\frac{v_1 v_2 (e^{\frac{\lambda}{m}t_1} - e^{\frac{\lambda}{m}t_2})}{\lambda(v_2 - v_1)}\right) \\
k = e^{\frac{\lambda}{m}(9-c_3)} - \frac{\lambda}{55} \\
\zeta(\lambda) = \partial_\lambda \sum_{(t,v)\in S, t\le 9}\left(\frac{\lambda e^{\frac{\lambda}{m}c_3}}{e^{\frac{\lambda}{m}t} - ke^{\frac{\lambda}{m}c_3}} - v\right)^2 \\
\lambda_{n+1} = \lambda_n - \frac{\zeta(\lambda_n)}{\zeta'(\lambda_n)} \\
\lambda_0 = 921.7 \\
c_4 = 9 + \frac{2m\arctan\left(\frac{110k+\lambda}{\sqrt{4kW-\lambda^2}}\right)}{\sqrt{4kW-\lambda^2}} \\
\phi(W) = \partial_W \sum_{(t,v)\in S, t\ge 9}\left(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4 - t) - \arctan\left(\frac{2kv+\lambda}{\sqrt{4kW-\lambda^2}}\right)\right)^2 \\
W_{n+1} = W_n - \frac{\phi(W)}{\phi'(W)} \\
W_0 = 290000
\end{cases} \tag{3}
$$

I wrote another module for the second model:

```haskell
{-# LANGUAGE RecordWildCards #-}
{-# LANGUAGE TypeFamilies    #-}

module Solution.Second (Parameters, SolutionGen) where


import           Data.Proxy

import           AutoDiff
import           GivenData
import qualified Solution.Model as M
```

```haskell
data Parameters a =
    Parameters { pLambda :: a
               , pC3     :: a
               , pK      :: a
               , pW      :: a
               , pC4     :: a
               , pGen    :: SolutionGen a
               }

data SolutionGen a =
    SolutionGen { sgPoint1 :: Point a
                , sgPoint2 :: Point a
                }

instance Show a => Show (SolutionGen a) where
    show SolutionGen{..} =
         ", p1 = (" ++ show (pTime sgPoint1) ++ ", " ++ show (pSpeed sgPoint1) ++ ")"
      ++ ", p2 = (" ++ show (pTime sgPoint2) ++ ", " ++ show (pSpeed sgPoint2) ++ ")"

instance Show a => Show (Parameters a) where
    show Parameters{..} = "lambda = " ++ show pLambda
                       ++ ", c3 = " ++ show pC3
                       ++  ", K = " ++ show pK
                       ++  ", W = " ++ show pW
                       ++ ", c4 = " ++ show pC4
                       ++ show pGen ++ "\n"

c3F :: Floating a => Point a -> Point a -> a -> a
c3F (Point t1 v1) (Point t2 v2) lambda =
  m / lambda * log ((v1*v2*(exp (lambda / m * t1) - exp (lambda / m * t2))) / (lambda * (v2 - v1)))

kF :: (Floating a, Ord a, Enum a) => a -> a -> a
kF c3 lambda = exp (lambda / m * (9 - c3)) - lambda / 55

sumOfSquares :: (Num a, Enum a) => [Point a] -> (Point a -> a -> a) -> a -> a
sumOfSquares ps f u = sum $ (^2) . flip f u <$> ps

zeta :: (Floating a, Ord a, Enum a) => Point a -> Point a -> a -> a
zeta p1 p2 = d $ sumOfSquares (take 10 points) j
  where j (Point t v) lambda = lambda * exp (h*c3) / (exp (h*t) - k * exp (h*c3)) - v
          where c3 = c3F (toDualPoint p1) (toDualPoint p2) lambda
                k = kF c3 lambda
                h = lambda / m

c4F :: Floating a => a -> a -> a -> a -> a
c4F c3 k lambda w =
  9 + 2 * m * atan ((2*v*k + lambda) / denom) / denom
  where denom = sqrt (4*k*w - lambda^2)
        h = lambda / m
        v = lambda * exp (h*c3) / (exp (h*9) - k * exp (h*c3))
```

```haskell
phi :: (Floating a, Enum a, Eq a) => a -> a -> a -> a -> a
phi c3 k lambda = d $ sumOfSquares (drop 9 points) j
  where j (Point t v) w =
          root * (c4 - t) / (2*m) - atan ((2 * constDual k * v + constDual lambda) / root)
          where c4 = c4F (constDual c3) (constDual k) (constDual lambda) w
                root = sqrt $ 4 * constDual k * w - constDual lambda ^2

findParams :: (Floating a, Ord a, Enum a) => SolutionGen a -> Parameters a
findParams gen@SolutionGen{..} = parameters
  where parameters = Parameters
          { pLambda = newton (zeta (toDualPoint sgPoint1) (toDualPoint sgPoint2)) (Left 1000) 921.7
          , pC3 = c3F sgPoint1 sgPoint2 (pLambda parameters)
          , pK = kF (pC3 parameters) (pLambda parameters)
          , pW = newton ( phi (constDual $ pC3 parameters)
                              (constDual $ pK parameters)
                              (constDual $ pLambda parameters) )
                        (Left 1000)
                        290000
          , pC4 = c4F (pC3 parameters) (pK parameters) (pLambda parameters) (pW parameters)
          , pGen = gen
          }

predict :: (Ord a, Floating a) => Parameters a -> a -> a
predict Parameters{..} t
  | 0 <= t && t <= 9  =
    let h = pLambda / m in pLambda * exp (h*pC3) / (exp (h*t) - pK * exp (h*pC3))
  | 9 <  t && t <= 26 =
    let r = sqrt $ 4*pK*pW - pLambda^2 in (r * tan (r * (pC4 - t) / (2 * m)) - pLambda) / (2 * pK)
  | otherwise = error "Invalid time!"


combinations :: (Num a, Enum a, Eq a) => [SolutionGen a]
combinations = fmap (uncurry SolutionGen)
             $ filter (uncurry (/=))
             $ (\l -> [ (p1, p2) | p1 <- l, p2 <- l])
             $ take 10 points

instance M.ModelParams Parameters where
    type SolutionGen Parameters = SolutionGen
    predict = predict
    findParams = findParams
    combinations Proxy = combinations
```

I also changed `import Solution.First as Sol` to `import Solution.Second as Sol` in the main module. This code gives us the following result:

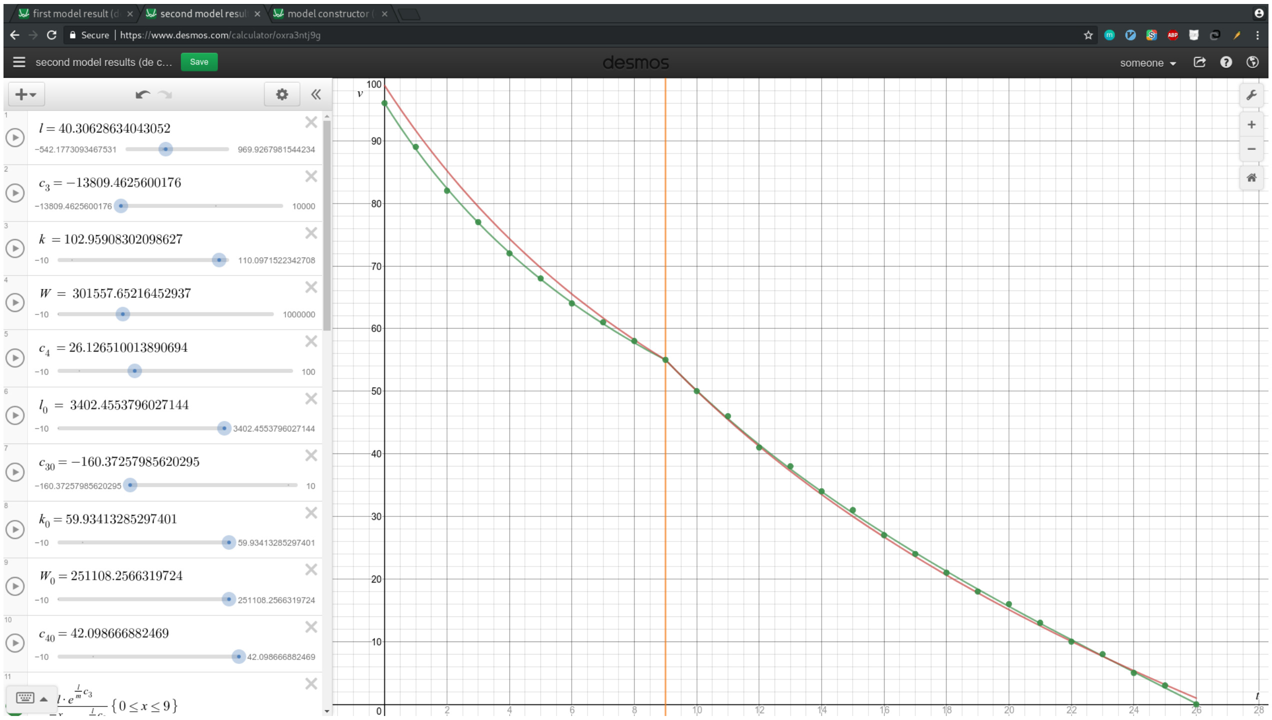
```
1 3                                                                                     23-04-2018 01:34:01
optimizer : bash  Konsole
λ.gleb::home-arch ⇒ optimizer → stack exec optimizer-exe
The best result:
lambda = 40.30628634043052, c3 = -13809.4625600176, K = 102.95908302098627, W = 301557.65216452937, c4 = 26.126510013890694, p1 = (0.0, 96.0), p2 = (9.0, 55.0)

| Maximum Error      | Average Error       | Root mean square    |
| 0.4558837377913889 | 0.23458170693637476 | 0.27311676642511934 |

| Time | Predicted Speed     | Rounded predicted speed | Observed Speed |
| 0.0  | 96.0000000000054    | 96                      | 96             |
| 1.0  | 88.66582512227822   | 89                      | 89             |
| 2.0  | 82.37077686914407   | 82                      | 82             |
| 3.0  | 76.90862555559508   | 77                      | 77             |
| 4.0  | 72.12432763292139   | 72                      | 72             |
| 5.0  | 67.89907302369585   | 68                      | 68             |
| 6.0  | 64.14028474770049   | 64                      | 64             |
| 7.0  | 60.77475738769026   | 61                      | 61             |
| 8.0  | 57.743842139329196  | 58                      | 58             |
| 9.0  | 55.00000000000179   | 55                      | 55             |
| 10.0 | 50.10157524003789   | 50                      | 50             |
| 11.0 | 45.60871998373162   | 46                      | 46             |
| 12.0 | 41.45588373779139   | 41                      | 41             |
| 13.0 | 37.5898231332195    | 38                      | 38             |
| 14.0 | 33.96673336319917   | 34                      | 34             |
| 15.0 | 30.550133354391157  | 31                      | 31             |
| 16.0 | 27.309281417030515  | 27                      | 27             |
| 17.0 | 24.217970303658813  | 24                      | 24             |
| 18.0 | 21.253597490201017  | 21                      | 21             |
| 19.0 | 18.396437545757497  | 18                      | 18             |
| 20.0 | 15.629064386691416  | 16                      | 16             |
| 21.0 | 12.935885534732083  | 13                      | 13             |
| 22.0 | 10.30276043083657   | 10                      | 10             |
| 23.0 | 7.716681813438704   | 8                       | 8              |
| 24.0 | 5.165504072768728   | 5                       | 5              |
| 25.0 | 2.6377059483218517  | 3                       | 3              |
| 26.0 | 0.12217734574065967 | 0                       | 0              |

The worst result:
lambda = 3402.4553796027144, c3 = -160.37257985620295, K = 59.93413285297401, W = 251108.2566319724, c4 = 42.098666882469, p1 = (6.0, 64.0), p2 = (5.0, 68.0)

| Maximum Error      | Average Error      | Root mean square   |
| 3.1959655309168937 | 0.9307094506962831 | 1.3222806170638275 |
```

3.4 Graphs and Analysis

Time	Predicted Speed	Rounded Predicted Speed	Observed speed
0.0	96.0000000000054	96	96
1.0	88.66582512227822	89	89
2.0	82.37077686914407	82	82
3.0	76.90862555559508	77	77
4.0	72.12432763292139	72	72
5.0	67.89907302369585	68	68
6.0	64.14028474770049	64	64
7.0	60.77475738769026	61	61
8.0	57.743842139329196	58	58
9.0	55.00000000000179	55	55
10.0	50.10157524003789	50	50
11.0	45.60871998373162	46	46
12.0	41.45588373779139	41	41
13.0	37.5898231332195	38	38
14.0	33.96673336319917	34	34
15.0	30.550133354391157	31	31
16.0	27.309281417030515	27	27
17.0	24.217970303658813	24	24
18.0	21.253597490201017	21	21
19.0	18.396437545757497	18	18
20.0	15.629064386691416	16	16
21.0	12.935885534732083	13	13
22.0	10.30276043083657	10	10
23.0	7.716681813438704	8	8
24.0	5.165504072768728	5	5
25.0	2.6377059483218517	3	3
26.0	0.12217734574065967	0	0

Maximum Error	Average Error	Root mean square
0.4558837377913889	0.23458170693637476	0.27311676642511934

I'll now look at the two configurations that I obtained and plot them. The red line shows the worst configuration, and the green line shows the best configuration.



As you can see in 3.3 rounded predictions of the best configuration exactly match the given data.

3.5 Conclusion of Model 2

$$\begin{cases} v = \frac{\lambda e^{\frac{\lambda}{m}c_3}}{e^{\frac{\lambda}{m}t} - ke^{\frac{\lambda}{m}c_3}}, & t \in [0,9] \\ v = \frac{\sqrt{4kW-\lambda^2}\tan(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4-t))-\lambda}{2k}, & t \in [9,26] \\ \lambda = 40.30628634043052 \\ c_3 = -13809.4625600176 \\ k = 102.95908302098627 \\ W = 301557.65216452937 \\ c_4 = 26.126510013890694 \end{cases} \tag{4}$$

$$x = \int_0^{26} v dt = \int_0^9 v dt + \int_9^{26} v dt$$

$$v_1 = \frac{\lambda e^{\frac{\lambda}{m}c_3}}{e^{\frac{\lambda}{m}t} - ke^{\frac{\lambda}{m}c_3}} = \frac{\lambda e^{\frac{\lambda}{m}(c_3-t)}}{1 - ke^{\frac{\lambda}{m}(c_3-t)}} = \frac{\frac{\lambda}{k} - \frac{\lambda}{k} + \lambda e^{\frac{\lambda}{m}(c_3-t)}}{1 - ke^{\frac{\lambda}{m}(c_3-t)}} = \frac{\frac{\lambda}{k}}{1 - ke^{\frac{\lambda}{m}(c_3-t)}} - \frac{\frac{\lambda}{k} - \lambda e^{\frac{\lambda}{m}(c_3-t)}}{1 - ke^{\frac{\lambda}{m}(c_3-t)}} = \frac{\frac{\lambda}{k}}{1 - ke^{\frac{\lambda}{m}(c_3-t)}} - \frac{\lambda}{k}$$

$$\Rightarrow \int v_1 dt = \frac{\lambda}{k}\left(\frac{m}{\lambda}\int \frac{\frac{\lambda}{m}e^{\frac{\lambda}{m}(t-c_3)}}{e^{\frac{\lambda}{m}(t-c_3)} - k}dt - \int 1dt\right) = \frac{m\ln(e^{\frac{\lambda}{m}(t-c_3)} - k) - \lambda t}{k} + \text{constant}$$

$$v_2 = \frac{\sqrt{4kW - \lambda^2}\tan(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4 - t)) - \lambda}{2k} = \frac{1}{2k}\left(\sqrt{4kW - \lambda^2}\tan\left(\frac{\sqrt{4kW - \lambda^2}}{2m}(c_4 - t)\right) - \lambda\right)$$

$$\text{Let } u = \frac{\sqrt{4kW - \lambda^2}}{2m}(c_4 - t)$$

$$\Rightarrow u'_t = -\frac{\sqrt{4kW - \lambda^2}}{2m}$$

$$\Rightarrow \int v_2 dt = \frac{1}{2k}\left(\sqrt{4kW-\lambda^2} \times (-\frac{2m}{\sqrt{4kW-\lambda^2}}) \int \tan(u) du - \int \lambda dt\right)$$

$$\Rightarrow \int v_2 dt = \frac{1}{2k}\left(2m \int \frac{-\sin u}{cosu} du - \lambda \int dt\right) = \frac{2m \ln(\cos(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4 - t))) - \lambda t}{2k} + \text{constant}$$

$$\Rightarrow x = \left[\frac{m\ln(e^{\frac{\lambda}{m}(t-c_3)} - k) - \lambda t}{k}\right]_0^9 + \left[\frac{2m\ln(\cos(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4 - t))) - \lambda t}{2k}\right]_9^{26}$$

$$\Rightarrow x = \frac{1}{k}\left(\left[m\ln(e^{\frac{\lambda}{m}(t-c_3)} - k)\right]_0^9 + \left[m\ln(\cos(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4 - t)))\right]_9^{26} - (13\lambda + 4.5\lambda)\right)$$

$$\Rightarrow x = \frac{m}{k}(\ln\left(\frac{e^{\frac{\lambda}{m}(9-c_3)} - k}{e^{\frac{\lambda}{m}(-c_3)} - k}\right) + \ln\left(\frac{\cos(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4 - 26))}{\cos(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4 - 9))}\right)) - 17.5\frac{\lambda}{k}$$

$$\Rightarrow x = \frac{m}{k}\ln\left(\frac{e^{\frac{\lambda}{m}(9-c_3)} - k}{e^{\frac{\lambda}{m}(-c_3)} - k} \times \frac{\cos(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4 - 26))}{\cos(\frac{\sqrt{4kW-\lambda^2}}{2m}(c_4 - 9))}\right) - 17.5\frac{\lambda}{k}$$

```
optimizer : stack Konsole
⟿ (\Parameters{..} → (m / pK) * log (((exp ((9 - pC3) * pLambda / m) - pK) / (exp ((- pC3) * pLambda / m) - pK)) * (cos ((pC4 - 26) * sqrt (4 * pK * pW - pLambda^2) / (2 * m)) / cos ((pC4 - 9) * sqrt (4 * pK * pW - pLambda^2) / (2 * m)))) - 17.5 * pLambda / pK) config
1057.8652498929307
```

$$\Rightarrow x \approx 1057.8652498929307 \approx 1060$$

As you can see the value of x is approximately equal to the one found from the first model, so recommendations are the same.

4 Assessment of the improvement obtained

4.1 Prediction

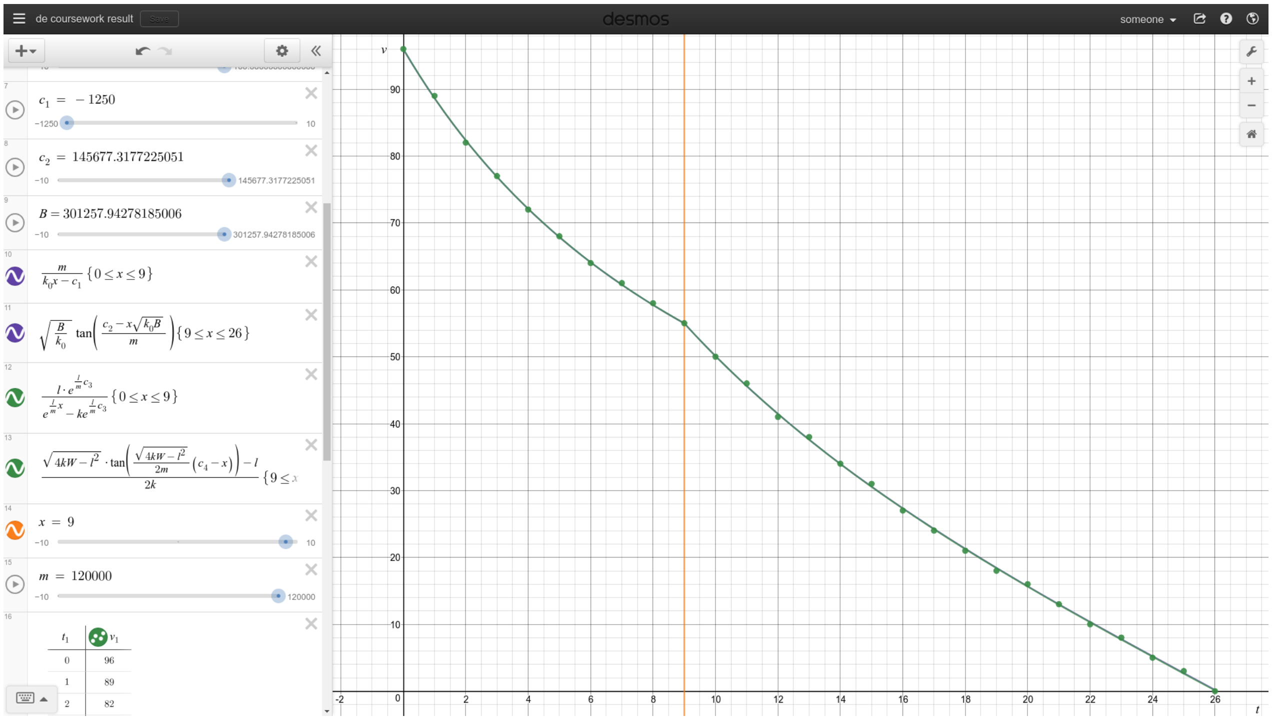
Based on the models I predict that the aeroplane stopped in 1060 meters. The length of runway that I recommend is 2000 meters for the reasons stated in the work.

4.2 Comparing the models

Time	Predicted Speed (1st Model)	Predicted Speed (2nd Model)	Observed Speed	Better Model
0.0	96.0	96.0000000000054	96	1st
1.0	88.65671641791045	88.66582512227822	89	2nd
2.0	82.35701906412478	82.37077686914407	82	1st
3.0	76.89320388349515	76.90862555559508	77	2nd
4.0	72.1092564491654	72.12432763292139	72	1st
5.0	67.88571428571429	67.89907302369585	68	2nd
6.0	64.12955465587045	64.14028474770049	64	1st
7.0	60.767263427109974	60.77475738769026	61	2nd
8.0	57.739975698663415	57.743842139329196	58	2nd
9.0	54.99999999999999	55.00000000000179	55	1st
10.0	50.10816658852111	50.10157524003789	50	2nd
11.0	45.622063036212374	45.60871998373162	46	2nd
12.0	41.475988862696376	41.45588373779139	41	2nd
13.0	37.61659930730753	37.5898231332195	38	1st
14.0	34.00002062308155	33.96673336319917	34	1st
15.0	30.589724513917773	30.550133354391157	31	1st
16.0	27.35493651754378	27.309281417030515	27	2nd
17.0	24.269426053394415	24.217970303658813	24	2nd
18.0	21.3105731783461	21.253597490201017	21	2nd
19.0	18.45863841578133	18.396437545757497	18	2nd
20.0	15.696183121094782	15.629064386691416	16	1st
21.0	13.00760227741885	12.935885534732083	13	1st
22.0	10.378741615250261	10.30276043083657	10	2nd
23.0	7.796577949697343	7.716681813438704	8	1st
24.0	5.248946558956161	5.165504072768728	5	2nd
25.0	2.7243028986249413	2.6377059483218517	3	1st
26.0	0.2115083630415522	0.12217734574065967	0	2nd

Model	Maximum Error	Average Error	Root mean square	Num. of better predictions
1st	0.47598886269637575	0.23822424753496707	0.2776875104911393	12
2nd	0.4558837377913889	0.23458170693637476	0.27311676642511934	15

Here is a graph of both of the models.



As you can see the second model gave more accurate predictions, but the increase in accuracy is very small.