rocket equations

Michael Schuresko

June 6, 2021

1 Rocket equations

So, the well known rocket equation, relating the Δv induced by a rocket burn as a function of the exhaust velocity, v_e, and the initial and final masses of the rocket (including any remaining fuel and payload), m_f and m_i, comes from observing that

$$\Delta v = \int_{m_f}^{m_i} \frac{v_e}{m} dm = v_e \ln \frac{m_i}{m_f}$$

2 Energy density of fuel

This means that for fixed v_e, the initial mass of a rocket grows exponentially with the Δv desired. Which is an undesirable state of things.

A natural inclination is to try to get around this by increasing v_e.

But that has limitations as well.

Every imaginable fuel (even matter-antimatter) has some ratio of energy stored per unit mass. Call this R_e

Once the energy is extracted from a piece of fuel, the remaining mass of the burnt fuel has little use except to shoot it out the back of the rocket as reaction mass (or, worse, carry along with the rocket as dead weight).

This gives us, at least for non-relativistic exhaust speeds, and negligible loss of mass through conversion to energy

$$R_e(m_i - m_f) = \int_{m_f}^{m_i} \frac{1}{2} v_e^2 dm$$

3 Tying it together

This is all well and good, and gives us an upper bound for v_e for chemical rockets that have a constant exhaust velocity. You can work it all out, tie it together, and figure out why constant v_e chemical rockets have to be giant (and mostly consist of fuel) to achieve spaceflight Δv

But one can do more interesting things with these equations.

4 What if

What if we could independently schedule the release of spent fuel mass and spent fuel energy? What if we could store part of the energy from burning a piece of fuel, and use it, later, to send a different piece of exhaust at a higher v_e when it is more optimal to do so? Conversely, what if we could save some of the spent fuel exhaust from burning a piece of fuel, impart all the energy onto a smaller piece of fuel exhaust, and launch that reaction mass later?

In short, what if we could make $v_e(m)$ a function of the mass remaining (m) ?

Those equations then become

$$R_e(m_i - m_f) = \int_{m_f}^{m_i} \frac{1}{2} v_e^2(m) dm$$

which, if you notice, for fixed m_i, m_f and R_e, constrain the inner product of $v_e(m)$ with itself.

And, subject to that, you, ideally, want to maximize

$$\Delta v = \int_{m_f}^{m_i} \frac{v_e(m)}{m} dm$$

But that's just the inner product of $v_e(m)$ with $\frac{1}{m}$!

In short, the optimal schedule for v_e as a function of the mass remaining, m, is to make v_e proportional to the reciprocal of the total spacecraft remaining mass.

5 Let's solve it!!!

Start with the first equation,

$$R_e(m_i - m_f) = \int_{m_f}^{m_i} \frac{1}{2} v_e^2(m) dm$$

And assume $v_e(m) = \frac{k}{m}$ for some unknown k that we're trying to solve for

$$R_e(m_i - m_f) = \int_{m_f}^{m_i} \frac{1}{2} \frac{k^2}{m^2} dm = \frac{k^2}{2} (\frac{1}{m_f} - \frac{1}{m_i})$$

Or

$$k = \sqrt{\frac{2R_e(m_i - m_f)}{\frac{1}{m_f} - \frac{1}{m_i}}}$$

Which simplifies to $k = \sqrt{2R_e m_i m_f}$

Now we want to solve

$$\Delta v = \int_{m_f}^{m_i} \frac{v_e(m)}{m} dm$$

with $v_e(m) = \frac{k}{m}$, which gets us

$$\Delta v = \int_{m_f}^{m_i} \frac{k}{m^2} dm = k(\frac{1}{m_f} - \frac{1}{m_i})$$

$$\Delta v = k(\frac{1}{m_f} - \frac{1}{m_i}) = k\frac{m_i - m_f}{m_i m_f}$$

$$\Delta v = \sqrt{2R_e m_i m_f}\frac{m_i - m_f}{m_i m_f} = (m_i - m_f)\sqrt{\frac{2R_e}{m_i m_f}}$$

This looks comforting because we could multiply all the masses by, say, 10, and the Δv would remain the same, just as it does with the classic rocket equation for constant v_e. Lets work that one out next

[80]:
```
from math import sqrt, log, exp
```

[81]:
```
## Just for the heck of it, let's write a python function that computes delta v
## for an ideal-scheduled rocket as a function of m_f, m_i and Re

def deltav_optimal(m_i, m_f, Re, m_curr = None):
    """
    m_curr allows us to plan a burn schedule for a certain amount of fuel, and␣
 ↪calculate velocity
    at any point along that burn trajectory
    """
    if m_curr is None:
        m_curr = m_f
    k = sqrt(2.0 * Re * m_i * m_f)
    return k * (1 / m_curr - 1/m_i)
```

6 Expression for constant exhaust velocity rocket

Let's work this out for a constant velocity exhaust rocket.

Recall from earlier that

$$\Delta v = v_e \log(\frac{m_i}{m_f})$$

Now we just need to find v_e

We have

$$R_e(m_i - m_f) = \int_{m_f}^{m_i} \frac{1}{2} v_e^2(m) dm$$

and, since v_e is a constant, we just have

$$R_e(m_i - m_f) = \frac{1}{2} v_e^2 (m_i - m_f)$$

or

$$v_e = \sqrt{2R_e}$$

That gives us

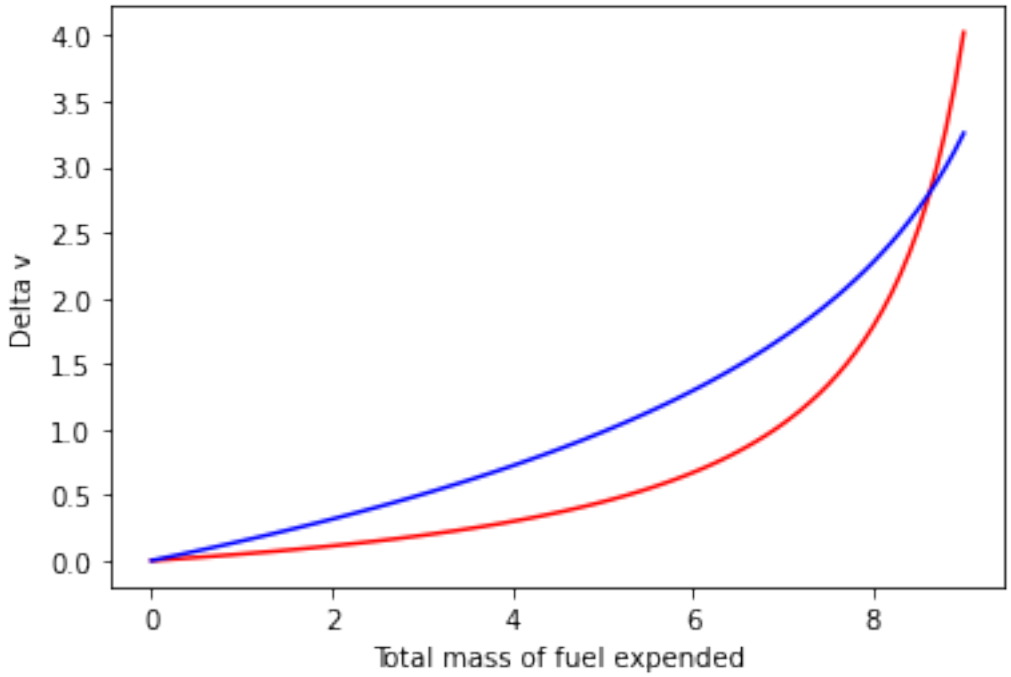
$$\Delta v = (\sqrt{2R_e}) \log(\frac{m_i}{m_f})$$

[82]:
```
## And we can calculate that in python too

def deltav_constant(m_i, m_f, Re, m_curr = None):
    """
    m_curr allows us to plan a burn schedule for a certain amount of fuel, and␣
 ↪calculate velocity
    at any point along that burn trajectory
    """
    if m_curr is None:
        m_curr = m_f
    v_e = sqrt(2.0 * Re)
    return v_e * log(m_i / m_curr)
```

[83]:
```
# Now that we have those calculations, let's plot velocity vs mass spent for␣
 ↪both kinds of rockets
from matplotlib import pyplot as plt
```

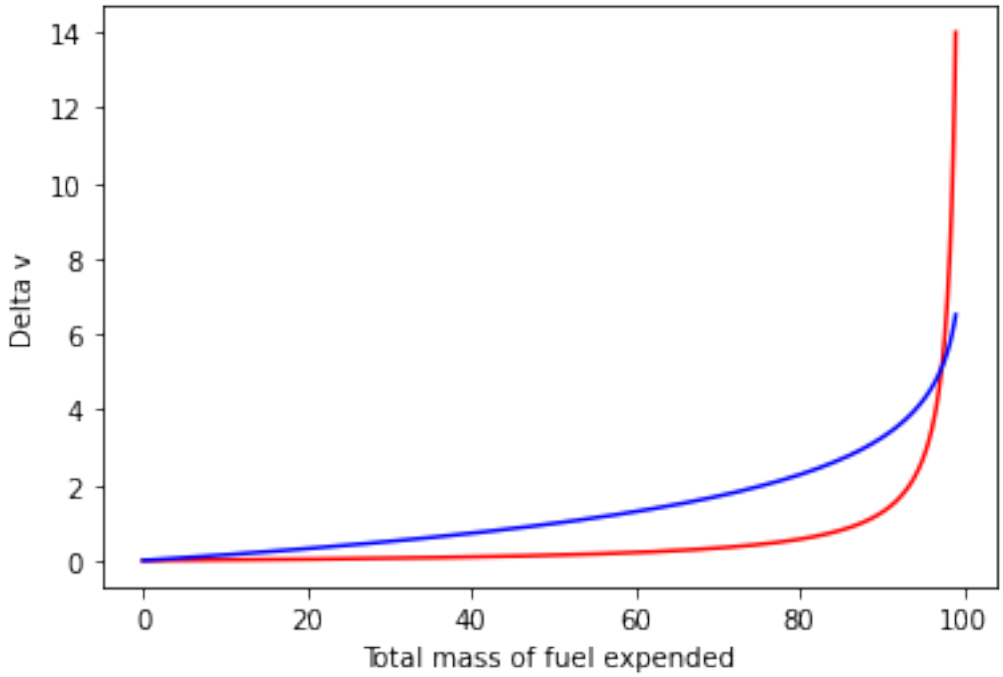
[84]:
```
plt.plot([0.1 * i for i in range(91)], [deltav_optimal(10.0, 1.0, 1.0, 10.0 - 0.
 ↪1 * i) for i in range(91)], 'r-')
plt.plot([0.1 * i for i in range(91)], [deltav_constant(10.0, 1.0, 1.0, 10.0 -␣
 ↪0.1 * i) for i in range(91)], 'b-')
plt.xlabel("Total mass of fuel expended")
plt.ylabel("Delta v")
```

[84]:
```
Text(0, 0.5, 'Delta v')
```



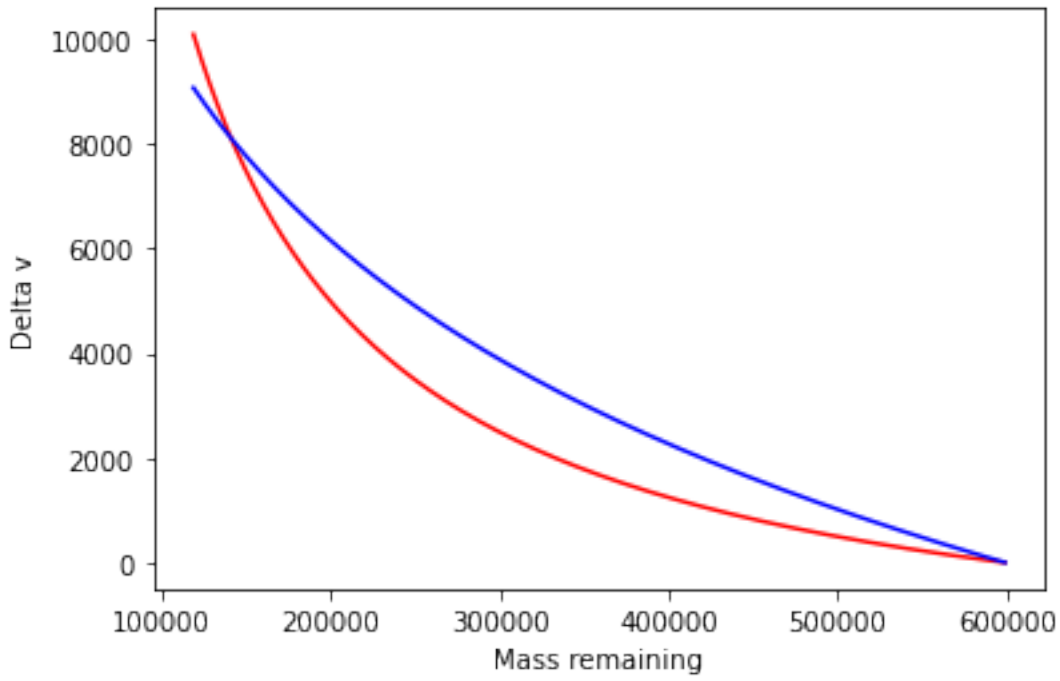
[85]:
```
## That's for a fuel-payload ratio of 10 to 1.  What happens when it's 100 to 1?

plt.plot([0.1 * i for i in range(991)], [deltav_optimal(100.0, 1.0, 1.0, 100.0␣
 ↪- 0.1 * i) for i in range(991)], 'r-')
plt.plot([0.1 * i for i in range(991)], [deltav_constant(100.0, 1.0, 1.0, 100.0␣
 ↪- 0.1 * i) for i in range(991)], 'b-')
plt.xlabel("Total mass of fuel expended")
plt.ylabel("Delta v")
```

[85]:
```
Text(0, 0.5, 'Delta v')
```



[86]:
```
## Let's try with some physically plausible numbers

h_energy_density = 141.8e6 / 9  # 141.8e6 is from wikipedia, dividing by 18 is␣
 ↪because we need to carry oxygen
# we need to carry 16 mass for every 2 mass of H2

saturn_v_booster_mass = 2.3e6
saturn_v_stage_2_mass = 0.48e6
saturn_v_stage_3_mass = 0.119e6

m_i_sv = saturn_v_stage_3_mass + saturn_v_stage_2_mass
m_f_sv = saturn_v_stage_3_mass

masses_sv = [m_f_sv + 0.001 * i * saturn_v_stage_2_mass for i in range(1001)]

plt.plot(masses_sv, [deltav_optimal(m_i_sv, m_f_sv, h_energy_density, m) for m␣
 ↪in masses_sv], 'r-')
plt.plot(masses_sv, [deltav_constant(m_i_sv, m_f_sv, h_energy_density, m) for m␣
 ↪in masses_sv], 'b-')
plt.xlabel("Mass remaining")
plt.ylabel("Delta v")
print(str([deltav_optimal(m_i_sv, m_f_sv, h_energy_density),␣
 ↪deltav_constant(m_i_sv, m_f_sv, h_energy_density)]))
```

```
[10092.208262381673, 9072.152221635844]
```



So under real-world plausible mass ratios, optimal vs constant v_e doesn't matter *that* much!

But an extra km/s would still be useful if it could be achieved.

Later we'll look into how feasible that might be, and how much energy we'd need to store in a rocketry scenario like this.

But first we'll compare these simulation numbers to the actual Δv from the rocket this example was based on.

[87]:
```
# Let's compare to the actual Saturn-V second stage delta v
vf_kmh = 25181
vi_ms = 2756
delta_v_m_per_s = (1000/3600) * vf_kmh - vi_ms
print("Achieved delta v from actual Saturn V stage 2 was " +␣
 ↪str(delta_v_m_per_s))
```

```
Achieved delta v from actual Saturn V stage 2 was 4238.722222222223
```

Huh, so our upper bound from the raw Tsiolkovsky equation wasn't that bad.

6.1 Comparison to billiard style collisions

A natural question that might arise now is how this "optimal" schedule for $v_e(m)$ as a function of the mass remaining compares to a hypothetical "billiard ball" collision, in which the entire exhaust is ejected at once as a rigid body in a way that preserves total system momentum, and adds a kinetic energy of $R_e(m_i - m_f)$.

We calculate that below.

Here we let v_r be the final velocity magnitude of the rocket, and abuse notation to use v_e to denote the final velocity magnitude of the ejected exhaust (as if it were a rigid body)

Total energy in terms of R_e and masses

$$E = R_e(m_i - m_f)$$

Momentum balance

$$m_f v_r = (m_i - m_f)v_e$$

Kinetic energy equals total energy

$$\frac{1}{2}(m_i - m_f)v_e^2 + \frac{1}{2}m_f v_r^2 = R_e(m_i - m_f)$$

Gather terms by masses

$$m_f v_r^2 = (m_i - m_f)(2R_e - v_e^2)$$

$$v_r = \sqrt{\frac{m_i - m_f}{m_f}}\sqrt{2R_e - v_e^2}$$

which we can plug back in to the momentum balance equation for

$$(m_i - m_f)v_e = m_f\sqrt{\frac{m_i - m_f}{m_f}}\sqrt{2R_e - v_e^2}$$

$$\sqrt{\frac{m_i - m_f}{m_f}}v_e = \sqrt{2R_e - v_e^2}$$

Well, now we have to square things again to get anything done

$$(\frac{m_i - m_f}{m_f})v_e^2 + v_e^2 = 2R_e$$

$$(\frac{m_i}{m_f})v_e^2 = 2R_e$$

$$v_e = \sqrt{\frac{2R_e m_f}{m_i}}$$

$$v_r = \frac{m_i - m_f}{m_f}\sqrt{\frac{2R_e m_f}{m_i}}$$

$$v_r = (m_i - m_f)\sqrt{\frac{2R_e}{m_i m_f}}$$

which is exactly what our optimal schedule of releasing mass and energy gave us.

Well, that's both relieving, and somewhat disappointing. It ties everything together in a satisfying way, but raises the question of whether starting from the billiard style collision would have been a better way to reach this optimal schedule

We also have an expression for $v_e(m)$ in terms of the remaining mass amd $v_r(m)$. What happens if we subtract these?

Recall

$$v_e(m) = \frac{\sqrt{2R_e m_i m_f}}{m}$$

$$v_r(m) = \sqrt{2R_e m_i m_f}(\frac{1}{m} - \frac{1}{m_i})$$

If we subtract these we get

$$v_e(m) - v_r(m) = \frac{\sqrt{2R_e m_i m_f}}{m_i} = \sqrt{\frac{2R_e m_f}{m_i}}$$

which is exactly the exhaust velocity from our rigid-body billiard collision thought experiment.

This, also, makes perfect sense.

7 How *much* energy does this require storing?

Well, the amount of energy released when the mass remaining is m_r is $ R_e (m_i - m_r) $

At the same time, the amount released via exhaust is

$$\int_{m_r}^{m_i} \frac{1}{2} v_e(m)^2 dm$$

where

$$v_e(m) = \frac{\sqrt{2R_e m_i m_f}}{m}$$

which gives us

$$\int_{m_r}^{m_i} \frac{1}{2} \frac{2R_e m_i m_f}{m^2} = \int_{m_r}^{m_i} \frac{R_e m_i m_f}{m^2} dm$$

or

$$(R_e m_i m_f)(\frac{1}{m_r} - \frac{1}{m_i})$$

This makes the difference between energy released and energy expended

$$R_e((m_i - m_r) - m_i m_f(\frac{1}{m_r} - \frac{1}{m_i}))$$

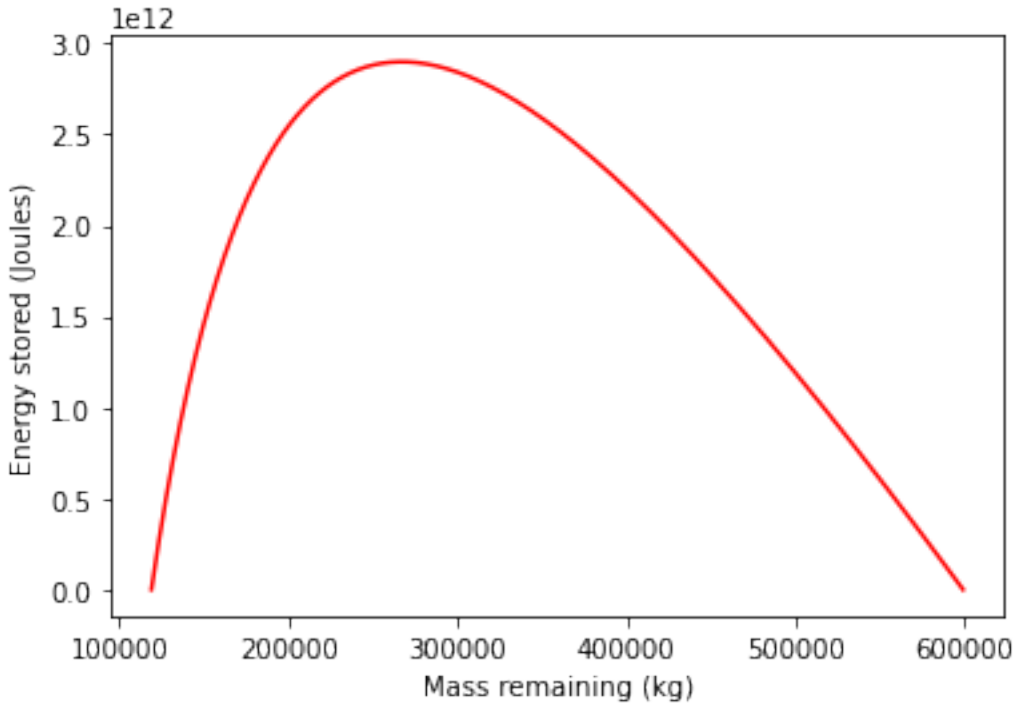
or

$$R_e((m_i - m_r) - (\frac{m_i m_f}{m_r} - m_f) = R_e(m_i + m_f - m_r - \frac{m_i m_f}{m_r})$$

[88]:
```
def energy_stored_optimal(m_i, m_f, Re, m_curr = None):
    """
    m_curr allows us to plan a burn schedule for a certain amount of fuel, and␣
 ↪calculate velocity
    at any point along that burn trajectory
    """
    if m_curr is None:
        m_curr = m_f
    return Re * (m_i + m_f - m_curr - (m_i * m_f) / m_curr)
```

[89]:
```
### Now let's plot this for our saturn V second stage


plt.plot(masses_sv, [energy_stored_optimal(m_i_sv, m_f_sv, h_energy_density, m)␣
 ↪for m in masses_sv], 'r-')
plt.xlabel("Mass remaining (kg)")
plt.ylabel("Energy stored (Joules)")
```

[89]:
```
Text(0, 0.5, 'Energy stored (Joules)')
```

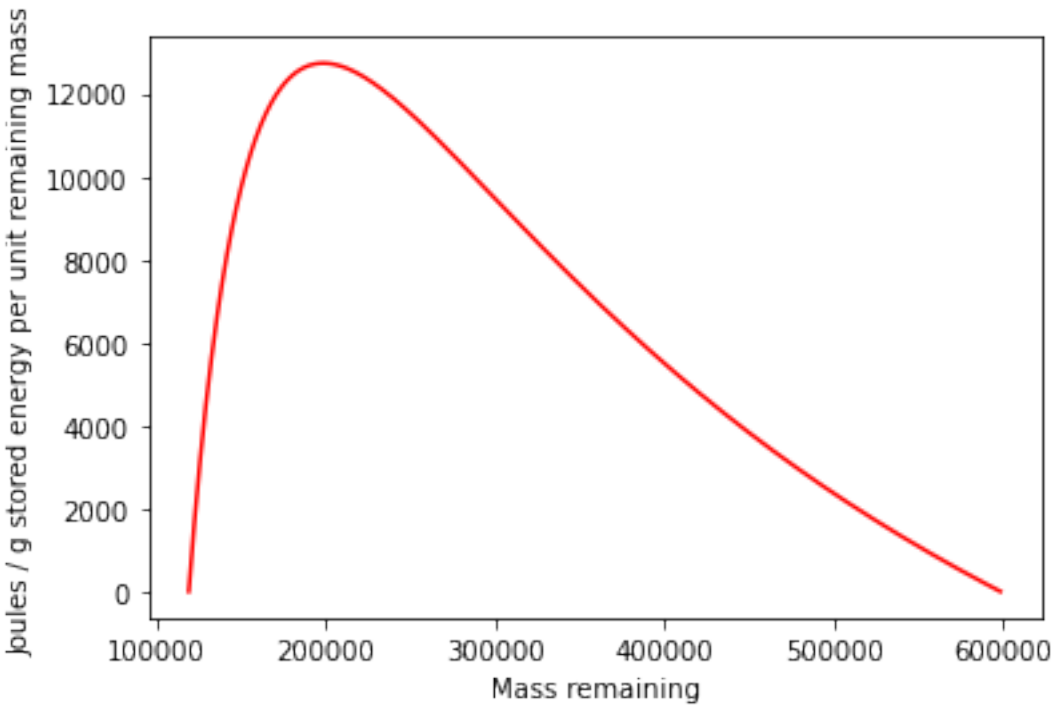


Wow, that's a lot of energy! What happens if we assume all of it is stored as heat in the remaining fuel?

We can look at what the resulting Joules/g stored heat energy would be (bearing in mind that Oxygen has a specific heat of very close to 1 Joule / (g degree Kelvin), although I am completely unsure of what kind of range that number holds over).

[90]:
```
### Now let's plot joules / g for our saturn V second stage


plt.plot(masses_sv, [energy_stored_optimal(m_i_sv, m_f_sv, h_energy_density, m)/
 ↪(1000.0 * m) for m in masses_sv], 'r-')
plt.xlabel("Mass remaining")
plt.ylabel("Joules / g stored energy per unit remaining mass")
```

[90]:
```
Text(0, 0.5, 'Joules / g stored energy per unit remaining mass')
```



12000 huh? I have no idea how plausible that is. If those Joles / g translate 1-to-1 into degrees Kelvin, that would be twice as hot as the surface of the sun. That's pretty hot.

8 Inverting our view

So far, we've been looking at how much more Δv we could get from a fixed amount of fuel by changing the schedule at which we release mass and energy.

What if we look at the problem the other way around : what if we want to look at how little fuel we can get away with for a fixed Δv and payload mass?

To do this we'll first write a crude numerical method to help us numerically invert our `delta_v_optimal` and `delta_v_constant` functions that we defined above.

[91]:
```
def find_zero(f, low, hi):
    if f(low) > 0:
        return find_zero(lambda x: -f(x), low, hi)
    while f(hi) < 0.0:
        hi = low + 2.0 * (hi - low)
    while abs(hi - low) > 1.0e-6 * abs(max(hi, low)):
        mid = 0.5 * (hi + low)
        if f(mid) < 0.0:
            low = mid
        else:
            hi = mid
```

```
    return 0.5 * (hi + low)

def inverse(f, val):
    return find_zero(lambda x: f(x) - val, 0.0, val)
```

[92]:
```
inverse(exp, 1.5)
```

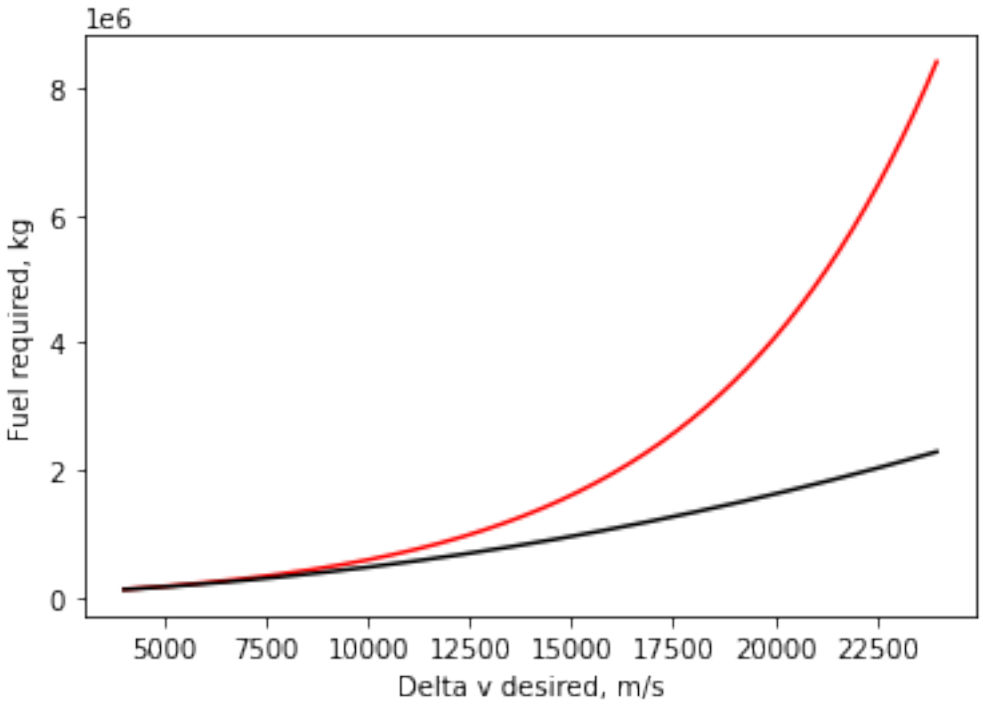
[92]:
```
0.4054650664329529
```

[93]:
```
exp(inverse(exp, 1.5))
```

[93]:
```
1.499999937487184
```

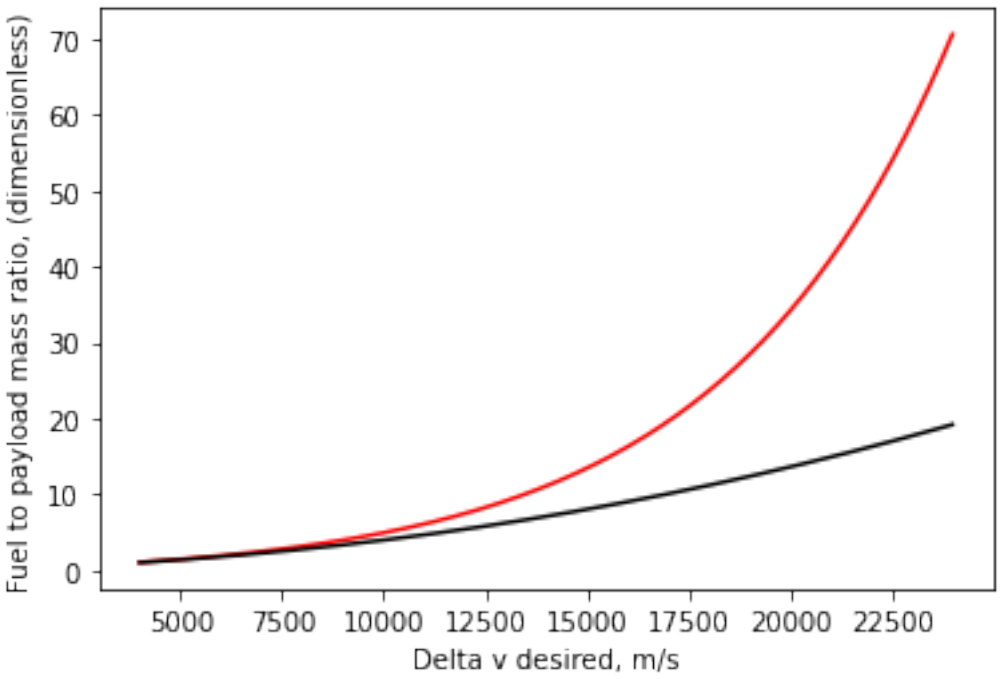
[94]:
```
delta_vs = [4000 + 20.0 * i for i in range(1000)]

const_v_e_masses = [inverse(lambda m: deltav_constant(m + m_f_sv, m_f_sv,␣
 ↪h_energy_density), v) for v in delta_vs]
opt_v_e_masses = [inverse(lambda m: deltav_optimal(m + m_f_sv, m_f_sv,␣
 ↪h_energy_density), v) for v in delta_vs]

plt.plot(delta_vs, const_v_e_masses, 'r-')
plt.plot(delta_vs, opt_v_e_masses, 'k-')

plt.xlabel("Delta v desired, m/s")
plt.ylabel("Fuel required, kg")
```

[94]:
```
Text(0, 0.5, 'Fuel required, kg')
```



[95]:
```
# Or, if you're more interested in fuel to payload mass ratios

delta_vs = [4000 + 20.0 * i for i in range(1000)]

const_v_e_mass_ratios = [m/m_f_sv for m in const_v_e_masses]
opt_v_e_mass_ratios = [m/m_f_sv for m in opt_v_e_masses]

plt.plot(delta_vs, const_v_e_mass_ratios, 'r-')
plt.plot(delta_vs, opt_v_e_mass_ratios, 'k-')

plt.xlabel("Delta v desired, m/s")
plt.ylabel("Fuel to payload mass ratio, (dimensionless)")
```

[95]:
```
Text(0, 0.5, 'Fuel to payload mass ratio, (dimensionless)')
```



As a computer programmer, having been trained to care about the asymptotic efficiency of processes, I naturally find these plots intriguing. But am I right to do so?

For one thing, asymptotics tend to only matter when analyzing functions of variables along which one expects to scale. And these two functions start to peel away at Δv beyond that required for most interplanetary travel within the solar system.

More importantly, though, the stored energy requirements for savings at the more extreme end might be increasingly prohibitive. Let's analyze them!

Recall that stored energy, when m_r kg of mass remains, can be expressed as

$$R_e(m_i + m_f - m_r - \frac{m_i m_f}{m_r})$$

We want to find where the derivative of this over mass remaining reaches zero for $m_i < m_r < m_f$

$$\frac{d}{dm}\frac{1}{m_r}(m_i + m_f - m_r - \frac{m_i m_f}{m_r}) = 0$$

$$\frac{d}{dm}\frac{m_i + m_f}{m_r} - 1 - \frac{m_i m_f}{m_r^2} = 0$$

$$\frac{-m_i - m_f}{m_r^2} + \frac{2m_i m_f}{m_r^3} = 0$$

$$\frac{2m_i m_f}{m_r} = m_i + m_f$$

$$m_r = \frac{2m_i m_f}{m_i + m_f}$$

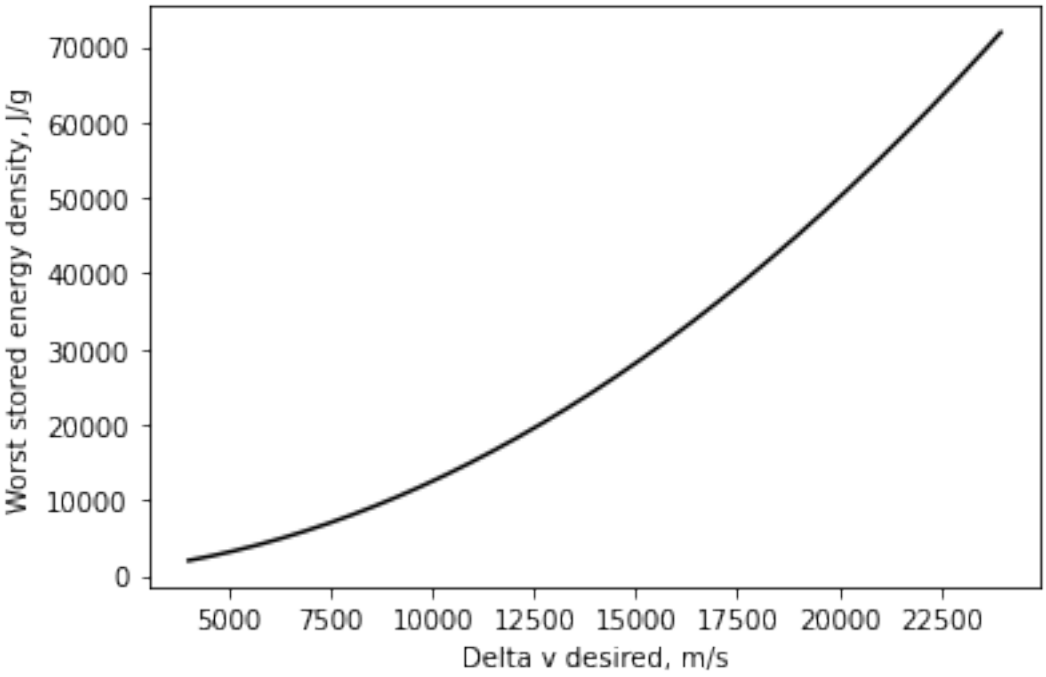
[96]:
```
# So, let's write a method to compute this worst-case point

def mass_of_worst_case_energy(m_i, m_f):
    return (2.0 * m_i * m_f)/(m_i + m_f)

def worst_case_energy_density(m_i, m_f, R_e):
    """ returns answer in Joules / g """
    m = mass_of_worst_case_energy(m_i, m_f)
    return energy_stored_optimal(m_i, m_f, R_e, m) / (1000.0 * m)
```

[97]:
```
# energy_stored_optimal(m_i_sv, m_f_sv, h_energy_density, m)/(1000.0 * m)

# Now let's check it with our Saturn V second stage example

worst_case_energy_density(m_i_sv, m_f_sv, h_energy_density)
```

[97]:
```
12731.58345141062
```

[98]:
```
# That checks out, let's plot it!

delta_vs = [4000 + 20.0 * i for i in range(1000)]

densities = [worst_case_energy_density(m + m_f_sv, m_f_sv, h_energy_density)␣
 ↪for m in opt_v_e_masses]
plt.plot(delta_vs, densities, 'k-')

plt.xlabel("Delta v desired, m/s")
plt.ylabel("Worst stored energy density, J/g")
```

[98]:
```
Text(0, 0.5, 'Worst stored energy density, J/g')
```



Those are some pretty steep energy densities required near the end. Probably not achievable by merely adding heat to the remaining fuel.

[ ]: