ECS 130: HW 4:

Problem 1 (15 pts). Let n be the smallest positive integer that ***cannot* be represented exactly** by a floating point number system as defined in class.

a) Determine a formula for n in terms of the number system's defining parameters (base β, precision p, and $[e_{\min}, e_{\max}]$). Briefly explain your reasoning. You can assume $p \leq e_{\max}$, which ends up meaning that you can ***ignore the possibility of overflow***: the first unrepresentable integer counting up from zero will be smaller than the largest representable floating point number!

b) Compute the value of n for both IEEE 754 single-precision and IEEE 754 double-precision.

Problem 1:

a) To determine a floating point formula, let us use the assumption $p \leq e_{max}$.
we know we have a base B, precision p & $[e_{min}, e_{max}]$

Notice fraction × base^{exponent}
(fraction = significand)

I know [eminimum, emaximum]
$e_{range} = e_{max} - e_{minimum} + 1$ → I am trying to compute total value of exponents
significands = B^{p-1} → this is because range [1, B) and B^p possible representations

Final form: $B^{p-1} \times (e_{max} - e_{min} + 1)$
↳ I ultimately multiplied both because (# of significand) × (exponent range) will give us correct formula.

b) compute the value n for IEEE 754 for single-precision and IEEE 754 double.
↳ we know:
- B = 2
- p = 24 (so 23 bits)
- $e_{min} = -126$, $e_{max} = 127$ ($\approx 10^{-38}$ to 10^{38})

$e = \hat{e} - 127$
for $1 \leq \hat{e} \leq 254$

Using
$e = e_{max} - e_{min} + 1$
$= 127 - (-126) + 1$
$= 254$

Now plugging into

$n = B^{p-1} \cdot (e_{max} - e_{min} + 1)$
$n = 2^{23} \cdot 254$
↳ $8,388,608 \times 254$
$n = 2,129,821,632$ for single-precision.

Now, for double-precision

Base $B = 2$

precision $p = 53$ bits

$e_{range} = 1023 - (-1022) + 1 = 2046$

# of significands $= B^{p-1} = 2^{53-1} = 2^{52}$

$n = 2^{52} \cdot 2046$

$n = 9,216,108,092,989,460,992$

c) complete the program

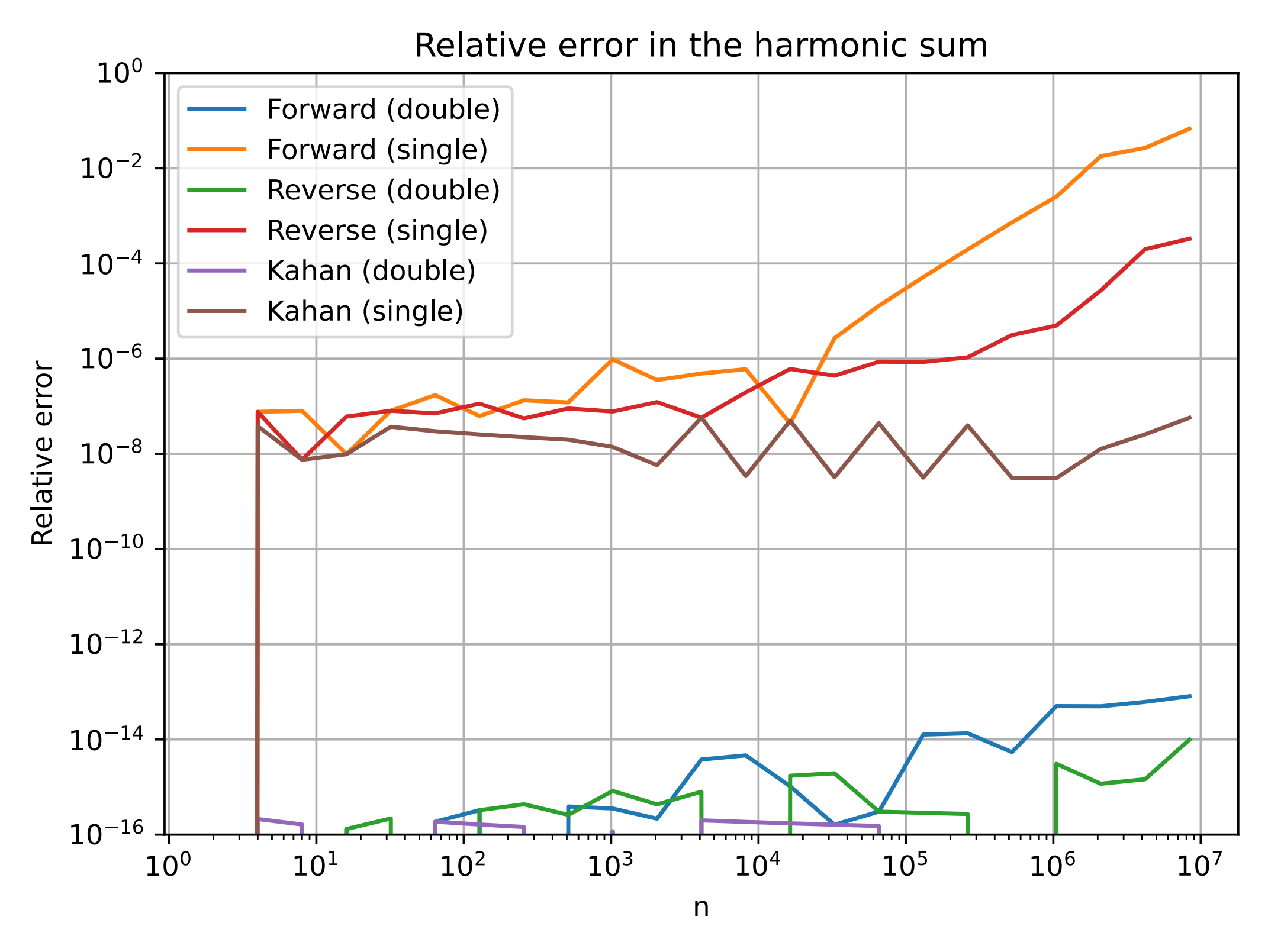
↳ I think the problems that are above n in $(n-8, n+8)$ are represented at a different precision that is why np.float64 contains exactly this range. Numbers BELOW are @ np.float32

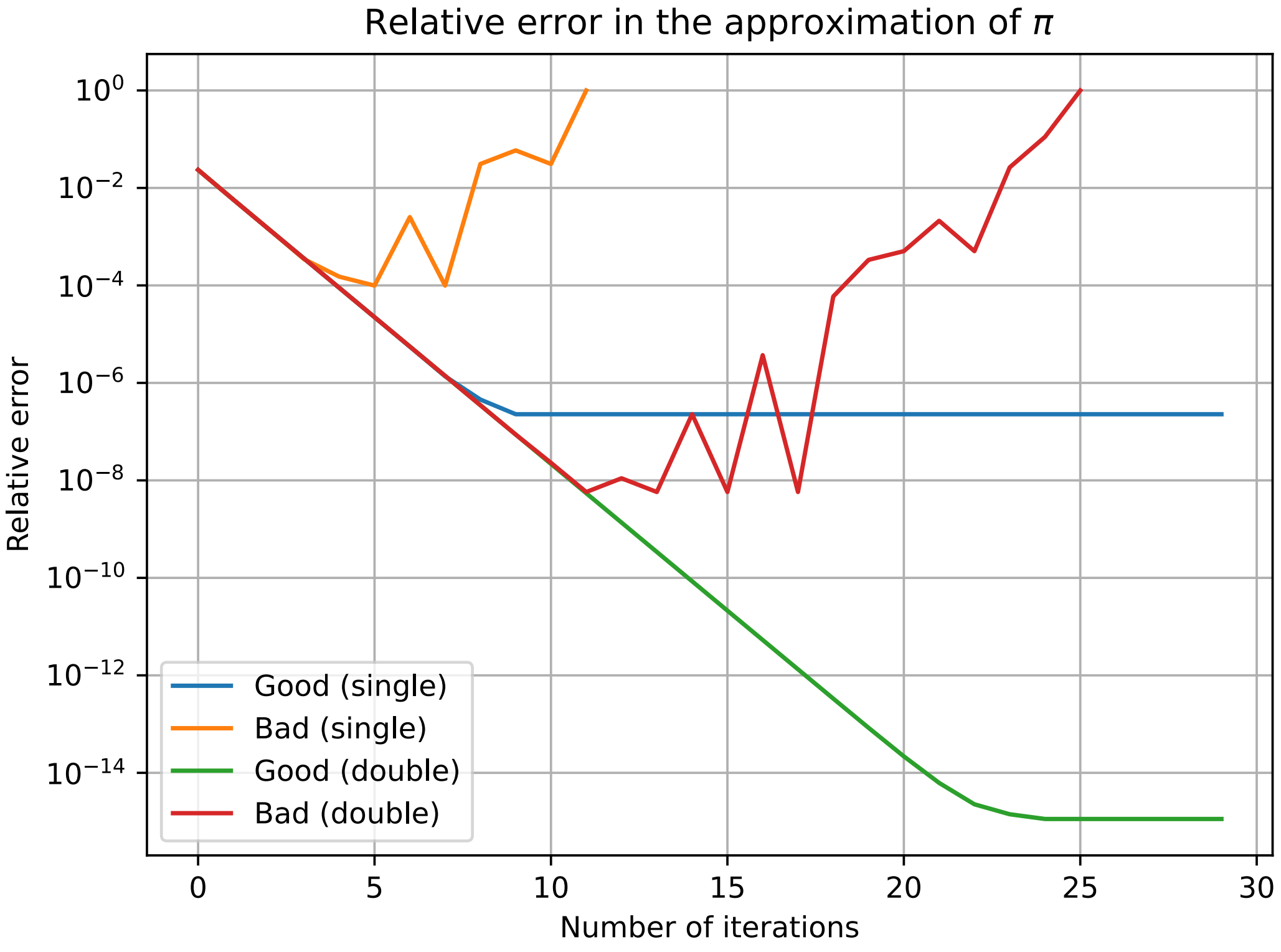
Problem 2:

Forward: The double precision has minimal errors as it continues lingering at 10^{-16} to 10^{-14}. Single precision has significantly more as it has almost the largest error.

Reverse: The double prec. for reverse has less than forward double, but not as little as kahan. Single precision has much more errors than double.

kahan: The single has the LEAST errors of all singles, while kahan double HAS LEAST OF ALL.


Relative error in the harmonic sum
Forward (double)
Forward (single)
Reverse (double)
Reverse (single)
Kahan (double)
Kahan (single)
Relative error
n
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
10^{7}


Relative error in the approximation of π
Relative error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
0
5
10
15
20
25
30
Number of iterations
Good (single)
Bad (single)
Good (double)
Bad (double)

Problem 3:

3b) derive mathematical

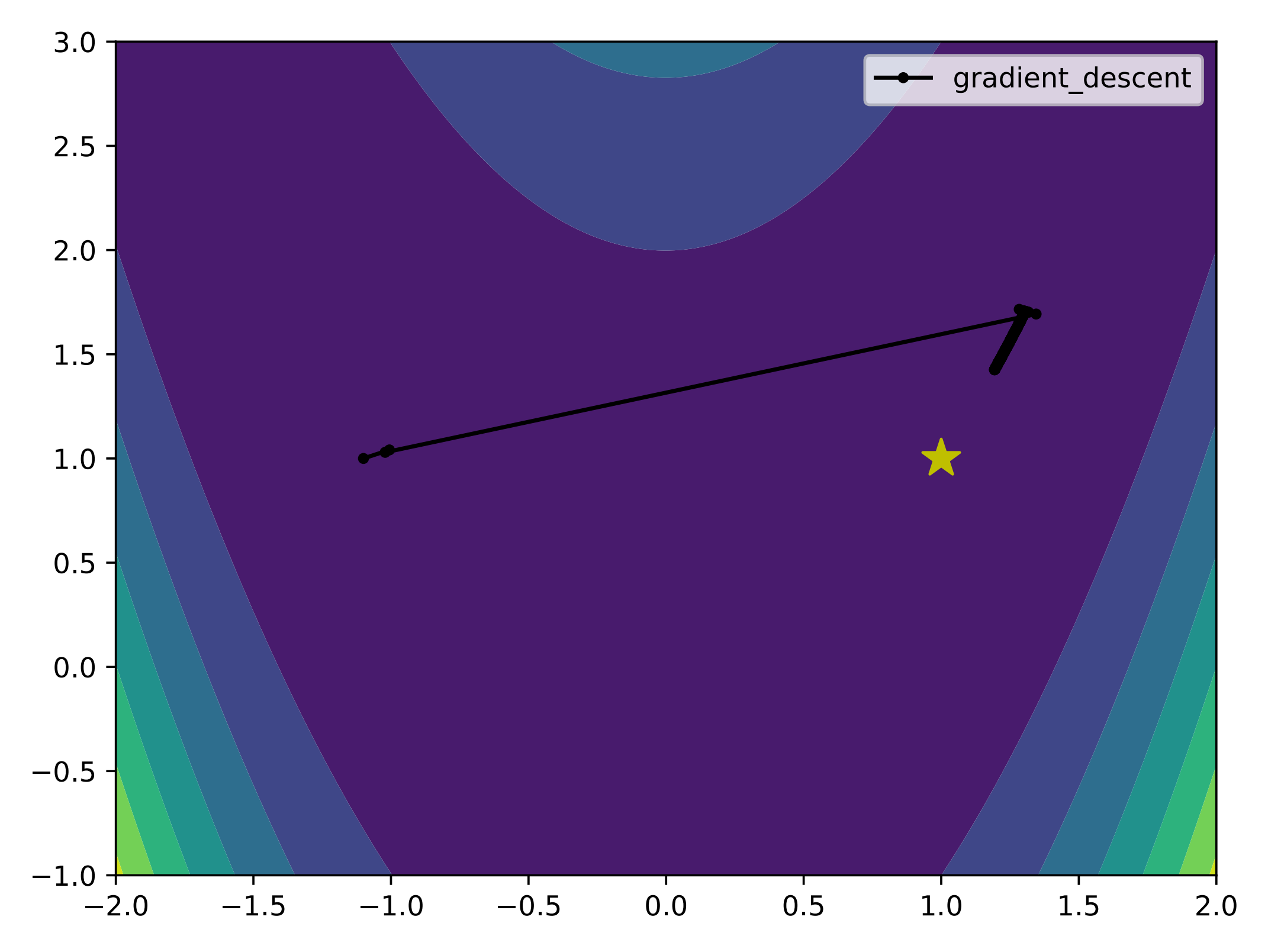
$$\frac{\sqrt{1+t^2} - t \cdot \sqrt{1+t^2} + 1}{\sqrt{1+t^2}}$$

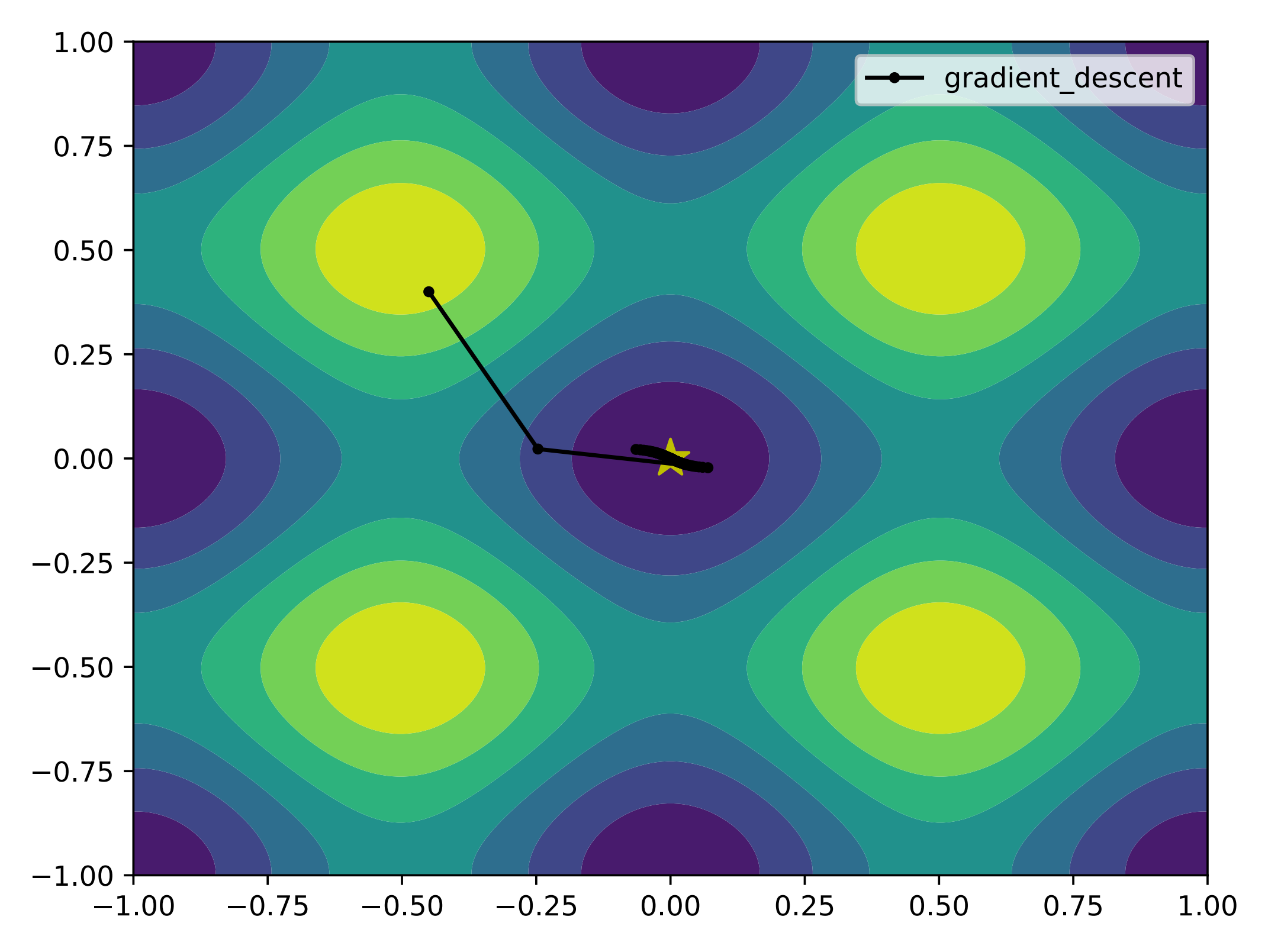
↳ $$\frac{1}{\sqrt{1+t^2} + t}$$

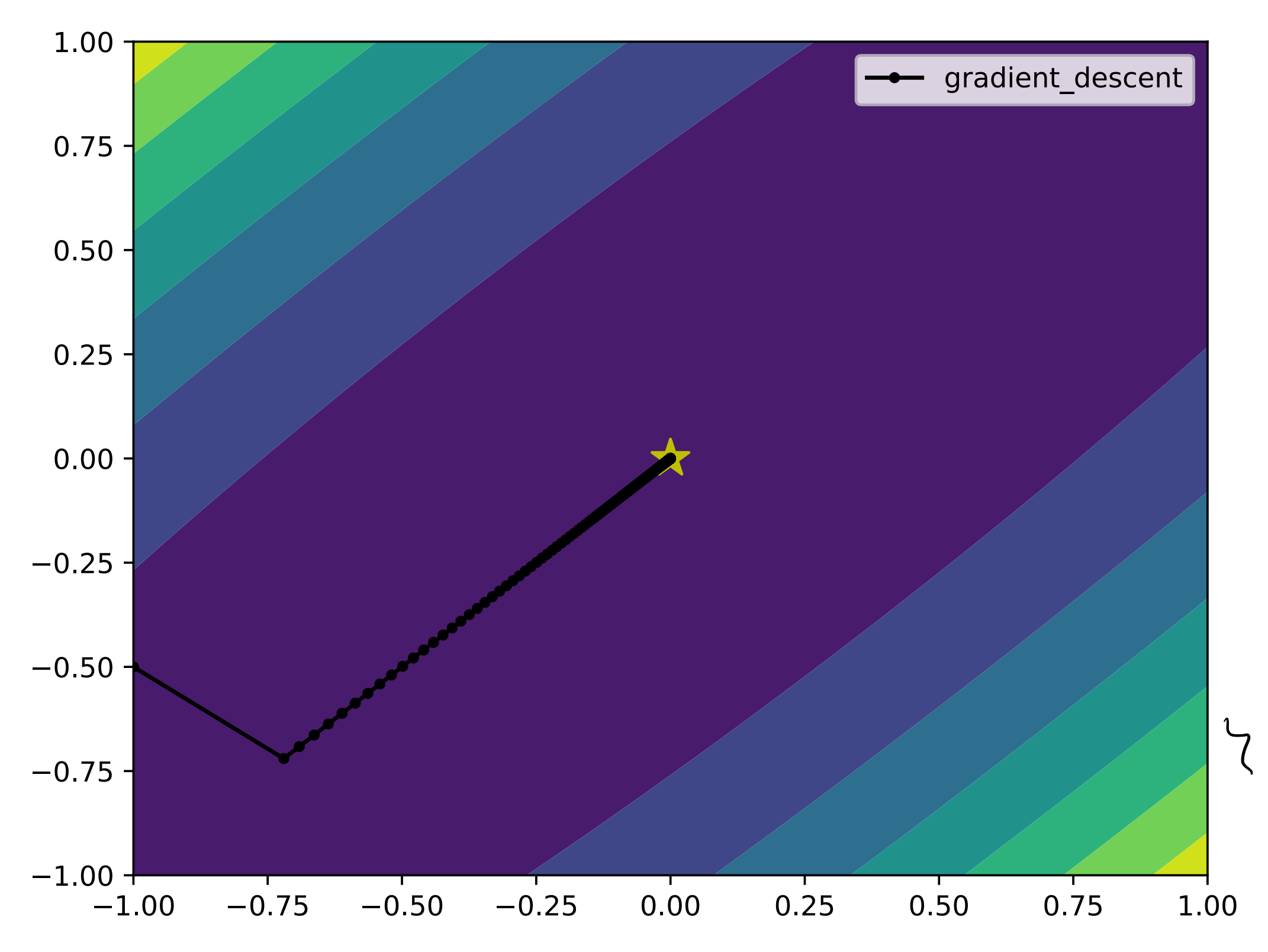
3c) so, I included the pi_error.pdf, I think good(double) had LOWEST error as it was closest to 10^{-14}. Bad(double) had next relative error as it lay at 10^{-9} but continued to increment.

Next, good single had a closer relative error on graph but had a lesser one b/c it was able to go smoothly. Bad(single) had a pretty high error and continued to increment

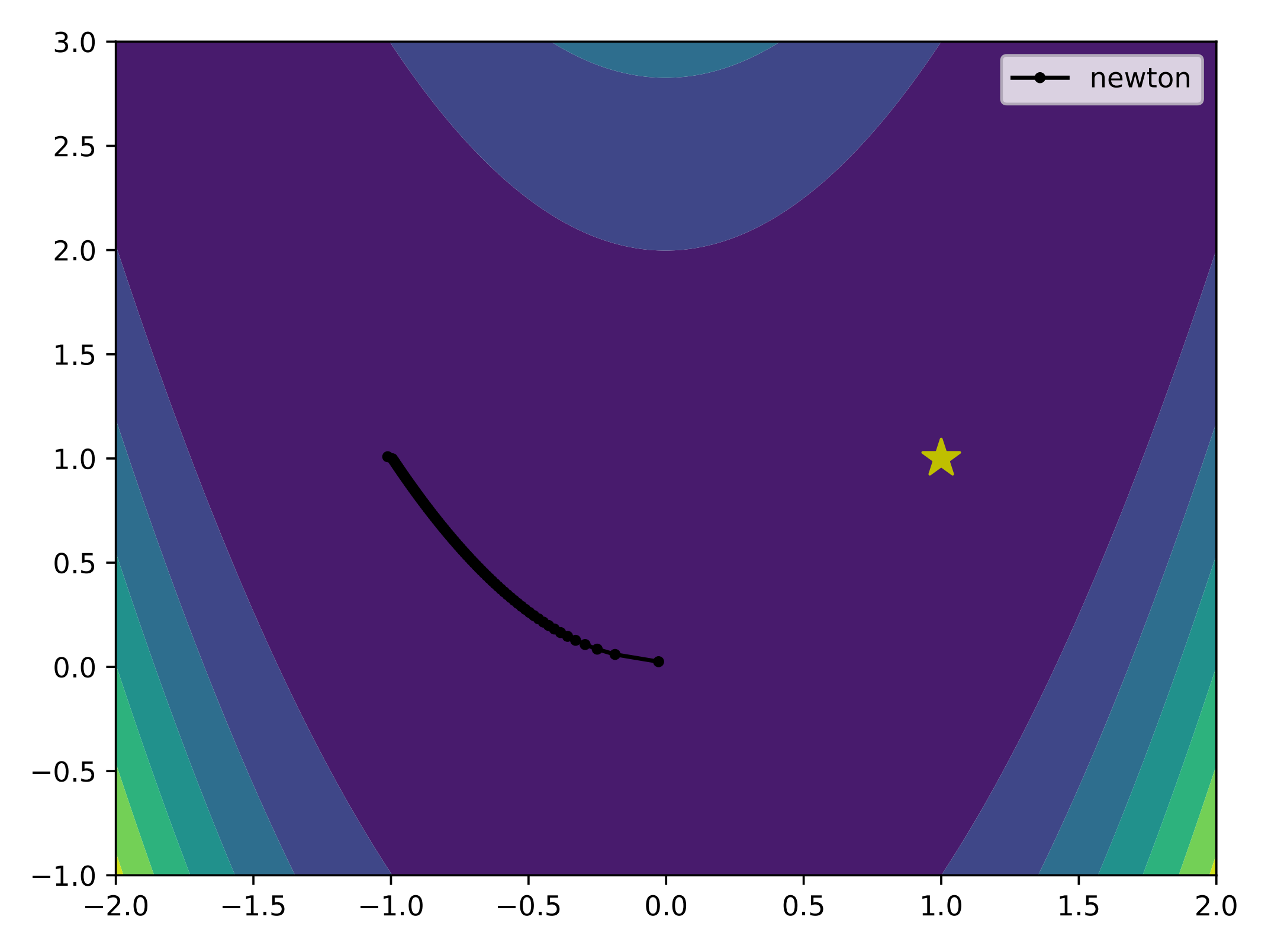
Rosenbrock, rastrigin, manyas descent_gradient.pdf:

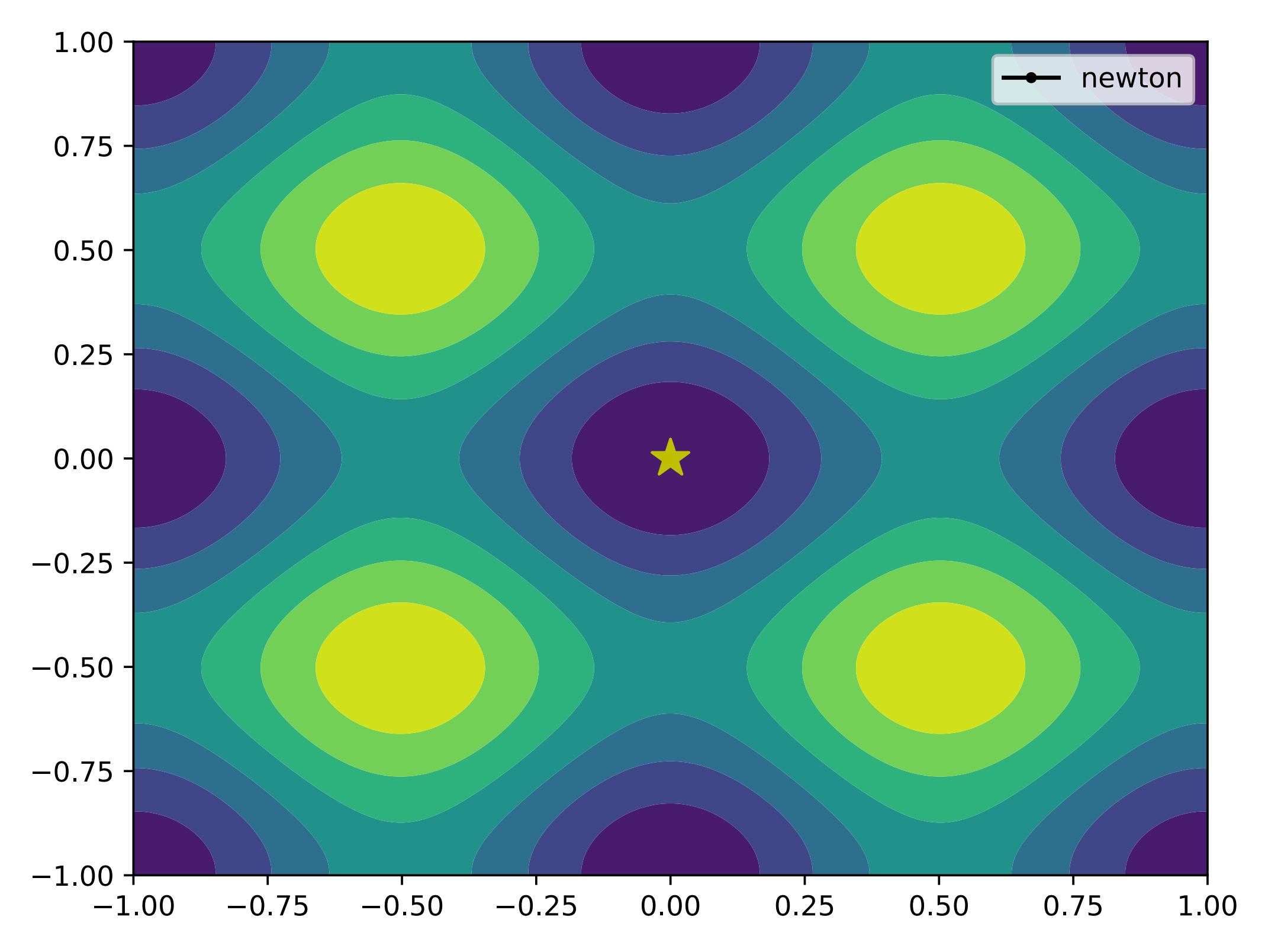

gradient_descent
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

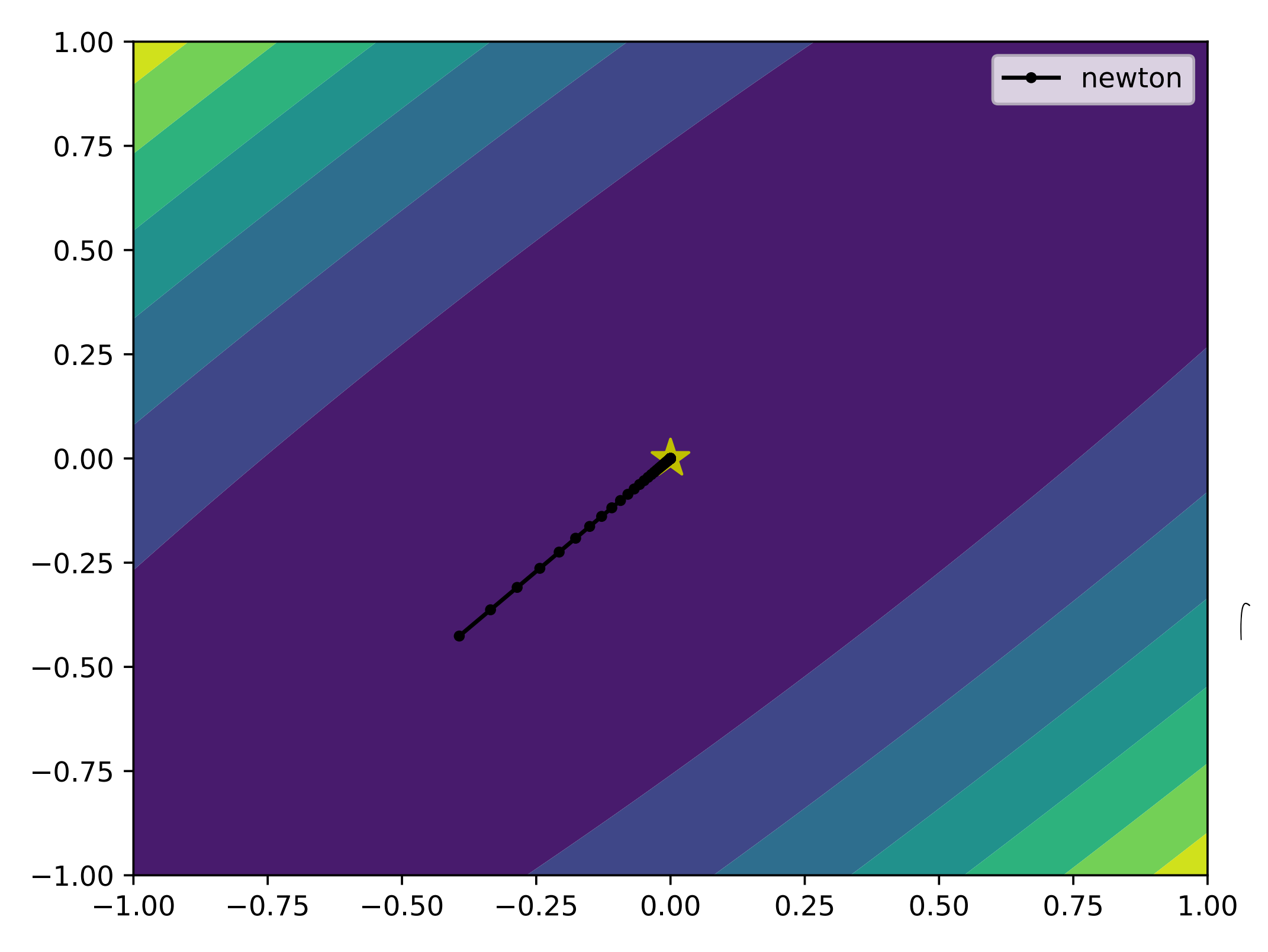

gradient_descent
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


gradient_descent
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Problem 6:
When I ran newton, for some reason it worked the best on matyas and I think this is because it produced such great values / plots with the implementation of Cholesky. I think Newton's worked worst w/ rosenbrock, bc it produced so many Nans

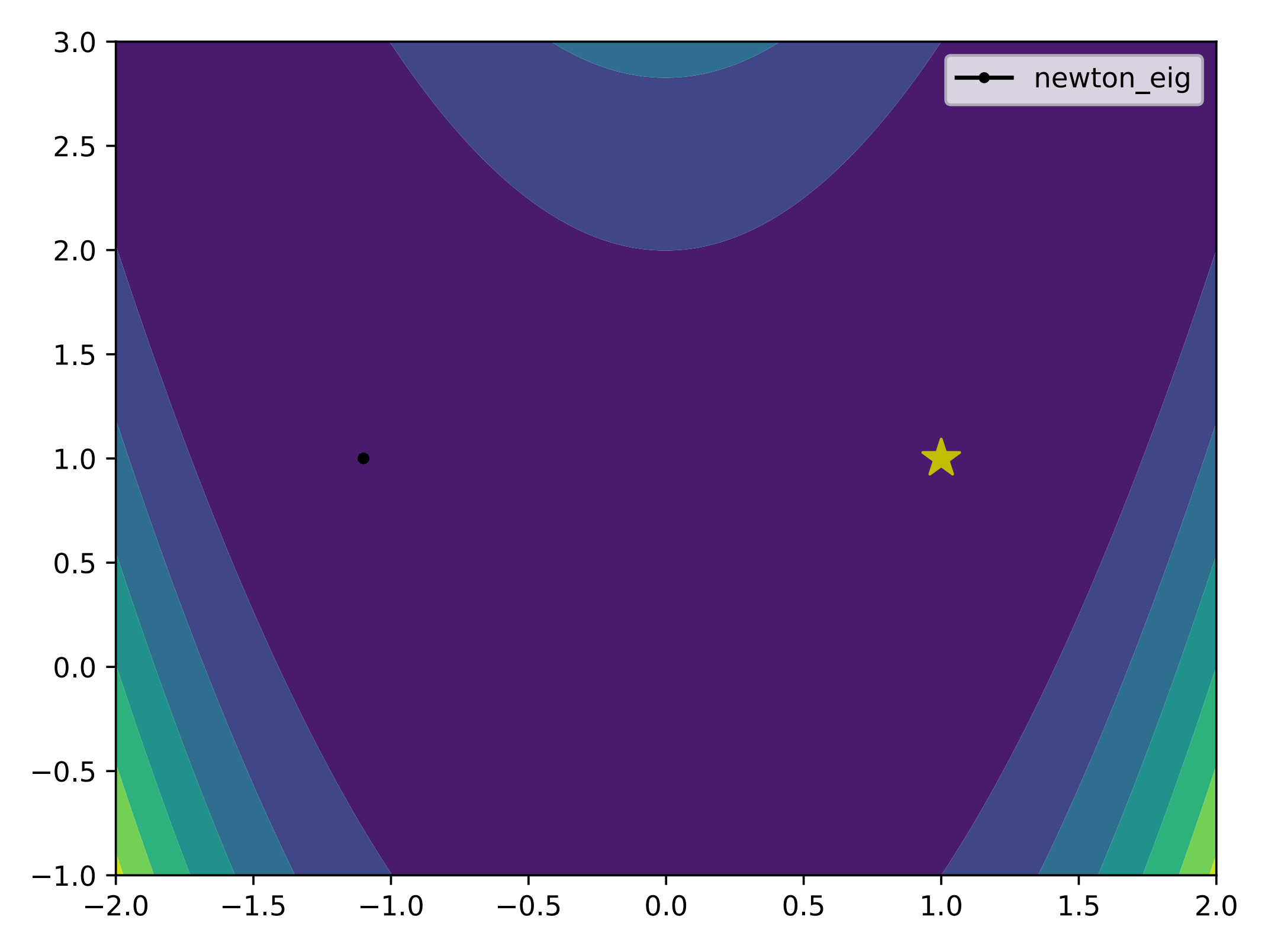

newton
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


newton
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


newton
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Problem 7:

↳ It actually created more errors for me here :(it only worked under Rosenbrock. I think this is b/c of its lower-order


newton_eig
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0