ECE3340

Polynomials, Zeroes and Poles, Roots and Contours

PROF. HAN Q. LE

- We have seen example of Laplace transformed circuits; where the transfer function can be described as a polynomial rational
- The system behavior is determined by the **zeroes** and **poles** of the transfer function, i. e. the **roots** of the numerator and denominator **polynomials**

- Similarly, many problems in science/engineering are to find solutions to equations:

$$left\ hand\ side(x) = right\ hand\ side(x)$$

$$left\ hand\ side(x) - right\ hand\ side(x) = 0$$

- This is known as "**find zero**" or "**find root**" of an equation.
- A system of equations has many equations and many unknowns to be solved.

How to compute roots, zeroes, polynomials and all that…numerically

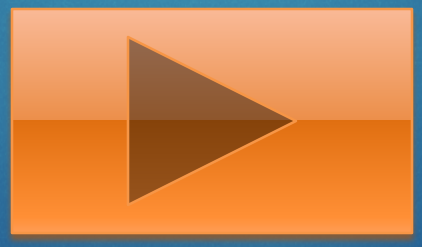
- Polynomials (***now that's the easy part***): Horner's method (or some similar variation) is well know and straightforward for minimizing computation steps and preserving precision.
- Finding roots: very common in scientific/engineering problems. It is more general than just for polynomials but also for functions requiring complex computation (most real world application functions cannot be expressed analytically in closed form but only via numerical computation).
- Extensive, rigorous algorithms have been developed. The algorithms can range from very simple all the way to complex and very involved mathematically (especially for complex roots).
- Widely available software have been developed with tried and true robust, efficient algorithms for these needs.

What does this mean?

- Do I need to know the nitty-gritty details of these algorithms?
  - Not essential, but it is good to have a conceptual understanding.
- Do I need to write my own codes of finding roots?
  - Only for coding exercise. Better to use many readily available codes (free in public domain). Some commercial software have more sophisticated algorithms to handle difficult cases (such as highly oscillatory functions).
- So, I don't really need to learn anything about these, but just use commercial software?
  - Not quite. There are certain things about potential errors or precision in challenging cases that we should be aware and can't just use the software blindly.

Horner's method & illustration

- Numerically calculate polynomial with minimized steps and memory (important only in the old days of computer).
- Implicitly implemented in advanced software by the order of computation via bracketing. (e. g. x (x (x(a x+b)+c))+d …)
  - a purpose of higher intelligence software is to eliminate trivial, tedious and repetitive do-loops (higher command such as Nest)
- Polynomial handling utilities are implemented in many advanced software (Mathematica and MATLAB).



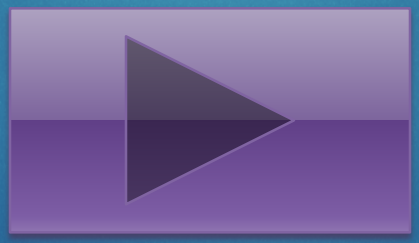
Common algorithms for finding roots of real function

- Finding zeroes of a polynomial is a subclass of general root finding. Specific algorithms for complex polynomial roots have been developed (and still being developed): “solve” vs. “find root”
- General root finding has a long history (not exhaustive):
  - Bisection search method & False-Position method
  - Fixed-Point iteration (limited applications)
  - Newton-Raphson & Secant method (popular)
    - Quadratic interpolation/extrapolation method
  - Brent’s method (root bracketing, also common for flexibility)
  - Extension to multiple roots with methods above
- How efficient each method is depends on the specific problems
- All scientific/engineering software have comparable root finding functions, e. g. fzero (MATLAB), FindRoot (Mathematica).

Root finding illustrations

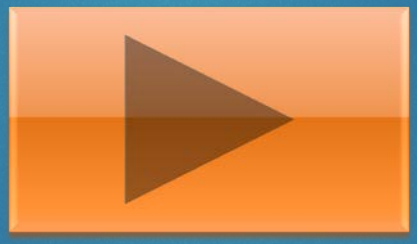
- Bisection & False position

- Newton-Raphson & Secant



Root finding illustrations

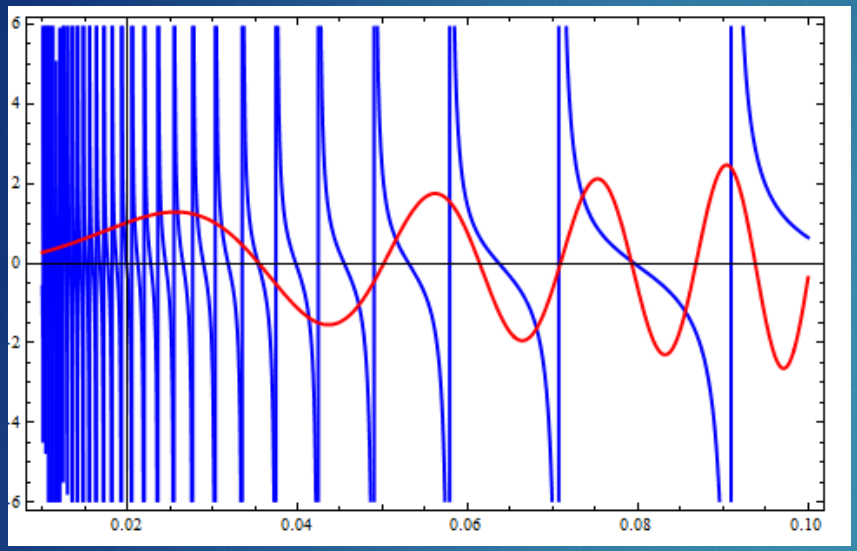
- Example of adaptive & flexible algorithm: FindRoot function in Mathematica, (including with Brent's root bracketing method).
- Use set precision option and set accuracy goal

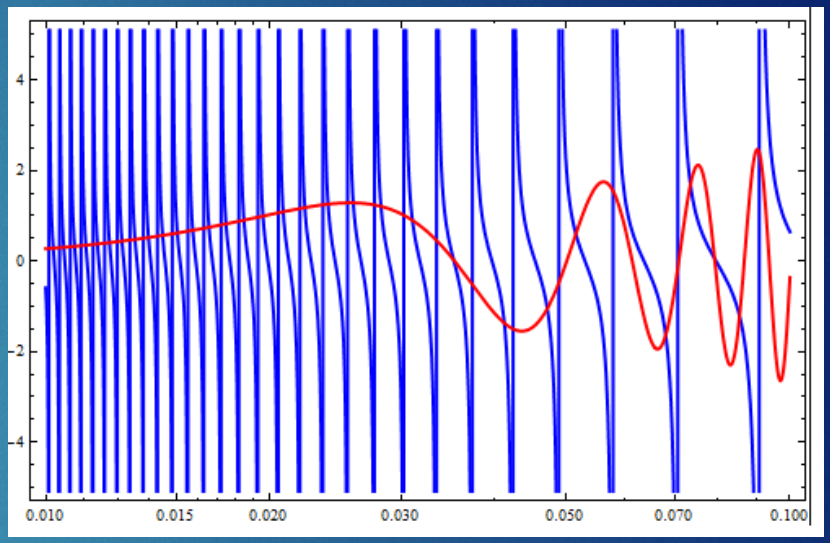


Bottom line question: should I use commercial root finding functions or code my own?

- Everyone (*once in the their life*) wrote a FORTRAN, BASIC, or C++ function for finding roots. It's for exercise, but commercial software have been thoroughly debugged and tested.
- Most software allow custom precision and accuracy. (*trade-off of precision/accuracy and computation time*).
- Software that compute zeroes of well-known analytic functions are also available (e. g. Bessel, Airy, Zeta zeroes in Mathematica).
- It's good to be aware how various algorithms work, but coding one for own use is ONLY for some unique case and unique need.
- Hence, for most practical purpose, one only needs to understand how to use various software.
- It is the set-up of the problem that is critical, not the software itself: Set up the problem in such a way that common algorithms work best, such as avoiding singularity, highly oscillatory behavior, etc.

Overarching considerations in finding roots

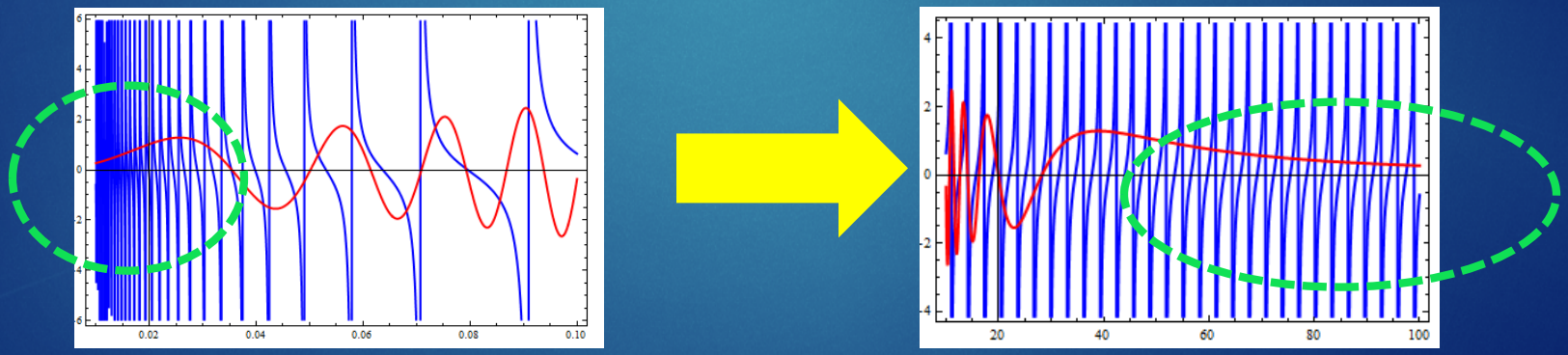




- Must have a general understanding of the behavior of the function(s):
  - does it have root(s) within certain known or expected range?
  - if many roots, how are the roots distributed?
  - are there singularities near the roots? or is the function highly oscillatory near the roots?

Overarching considerations in finding roots

- If the function is not well-behaved near the roots, transform the problem to obtain a well-behaved function: this is actually often the essence of a problem.
- Once properly transformed, various algorithms discussed above can be applied and they generally work quite well.
- Bottom line: one should not find roots "blindly." Must analyze to understand the function and know what to expect for the roots.

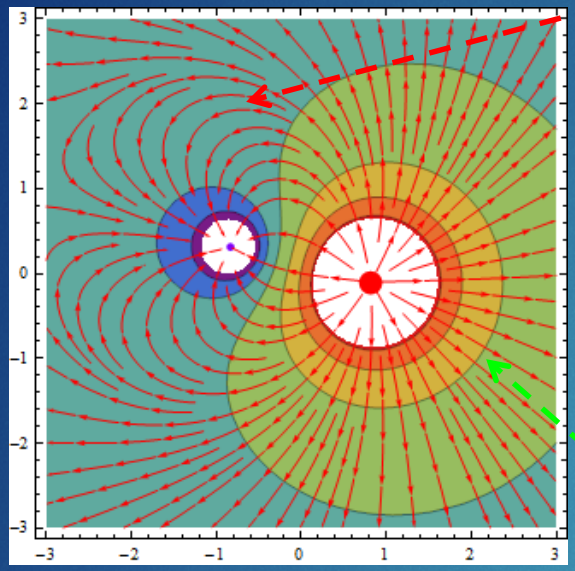


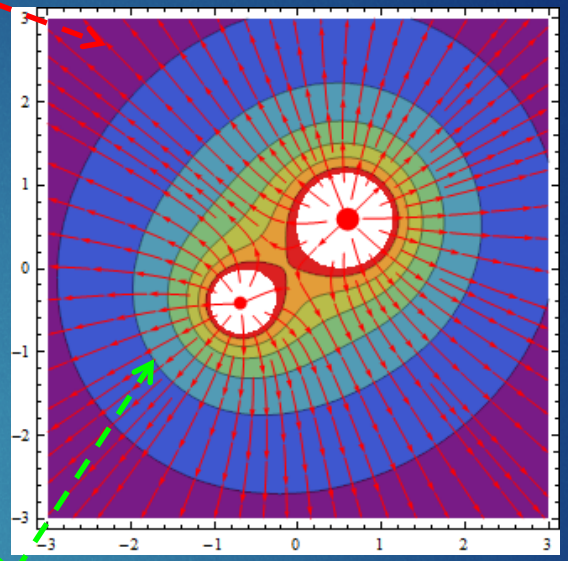
Transform the problem such that the roots are properly distributed, bracketable, and can be found

Classwork: example of oscillatory multiple root function

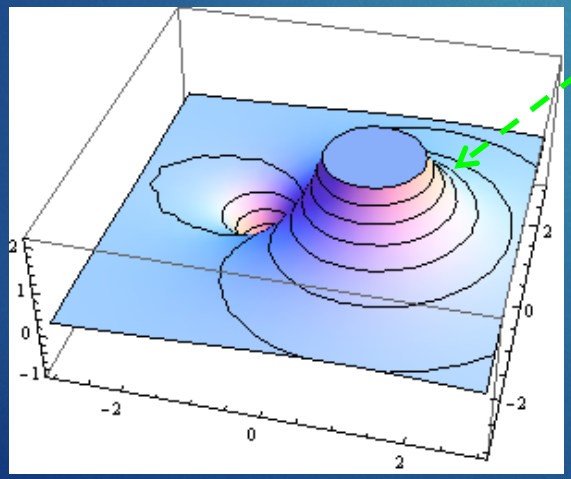
Contours, zeroes and poles

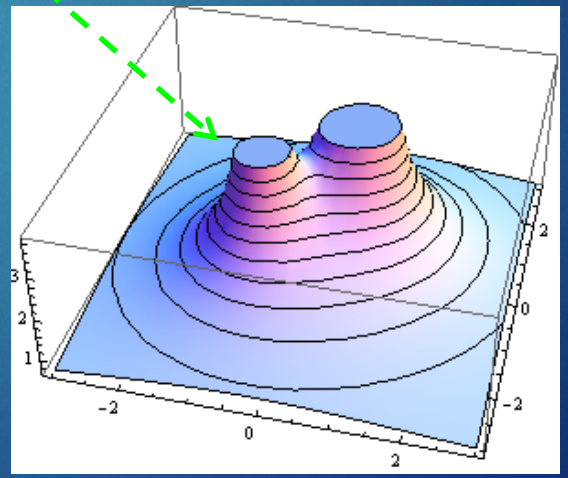
Electrostatic field lines





Equipotential surface



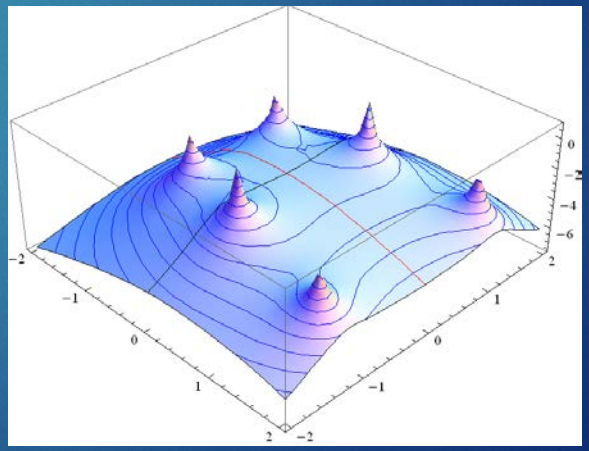


Contour is essentially partial find-root:

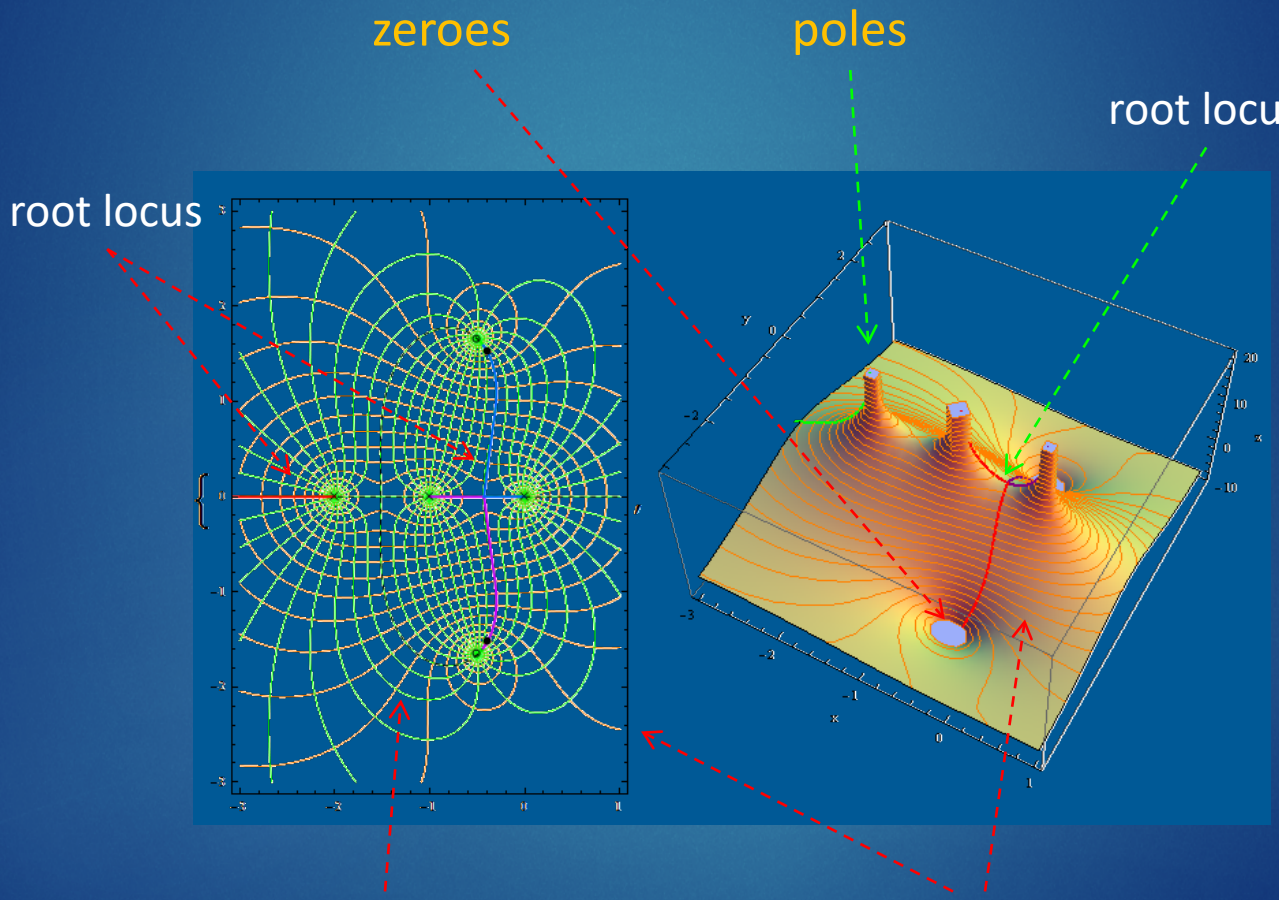
$$f(x, y, z, \ldots)=0$$

- Generate a search mesh
- Calculate points, e. g. {x,y} that satisfy the equation (find root) near a node.
- Join points and interpolate if necessary to obtain a smooth curve.

See exercise in generating contours for complex root problems.



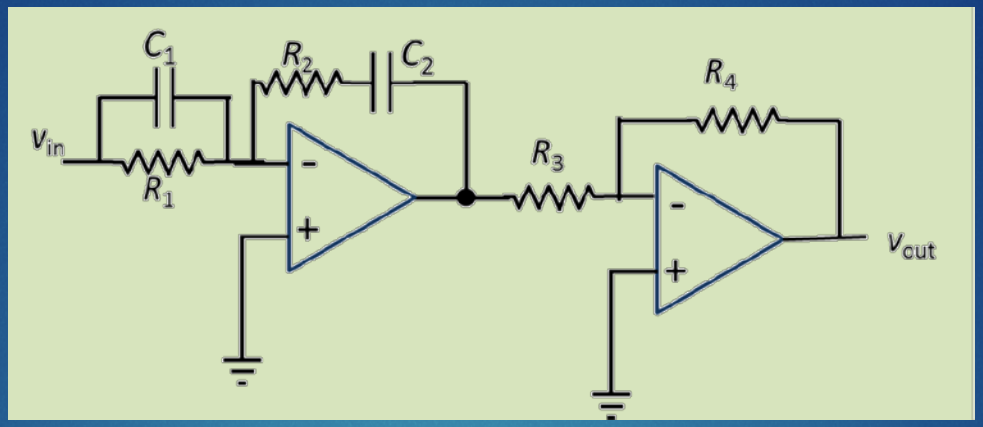
Example application: analysis of Laplace transformed open loop transfer function



https://www.youtube.com/watch?v=w2itwFJCgFQ&ebc=ANyPxKobWyIQ0HVSJi_RnKRqwE7pPVQMxBRwJxURdSPwgl1cUESlHONKy_La0GpTg0DwlFiUnYVXb8SsV-ROE6k0CYZQ6ZK4dQ

https://www.youtube.com/watch?v=cLHXgAKLxuo

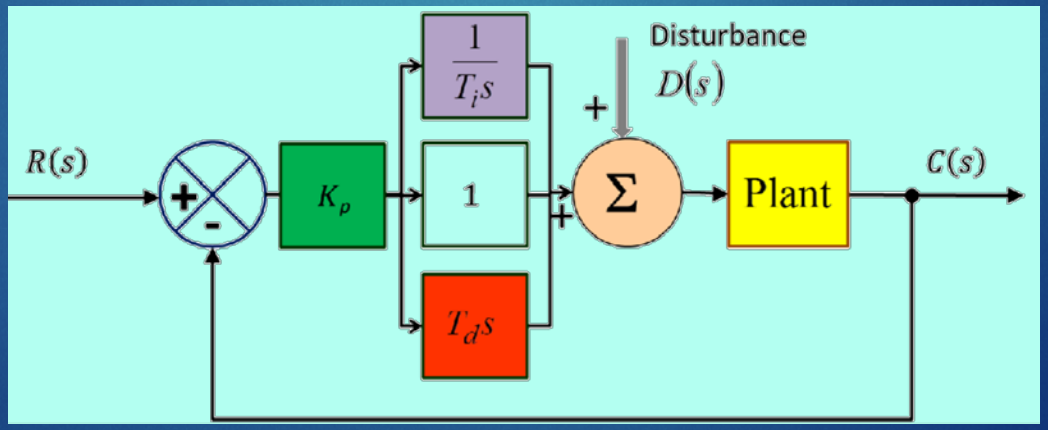
https://www.youtube.com/watch?v=kOEihjOwMAI

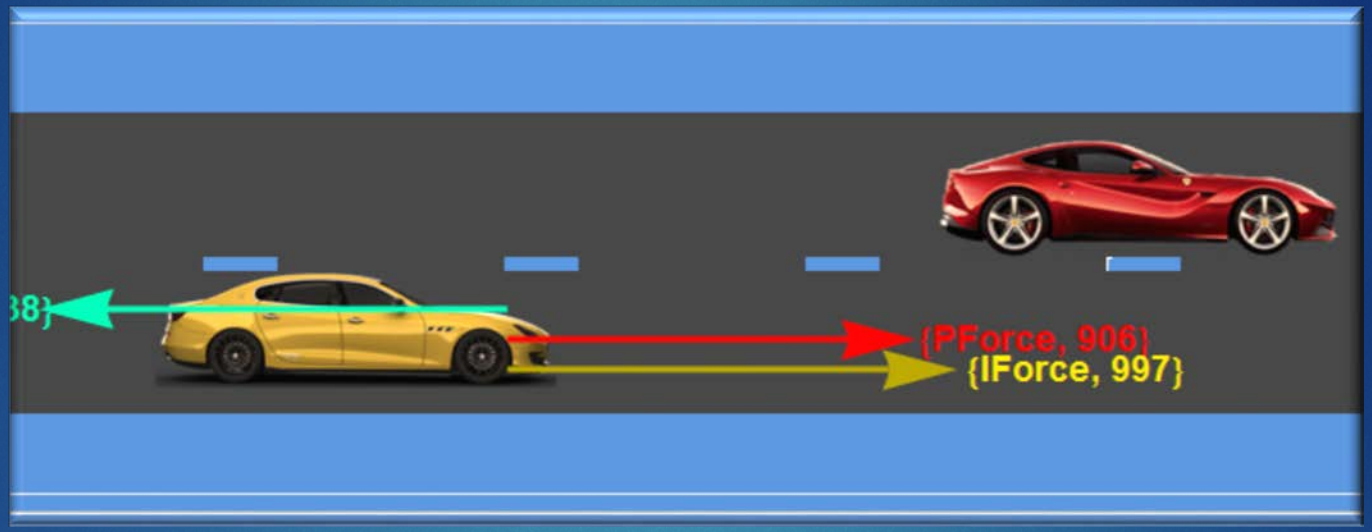


An analog PID controller

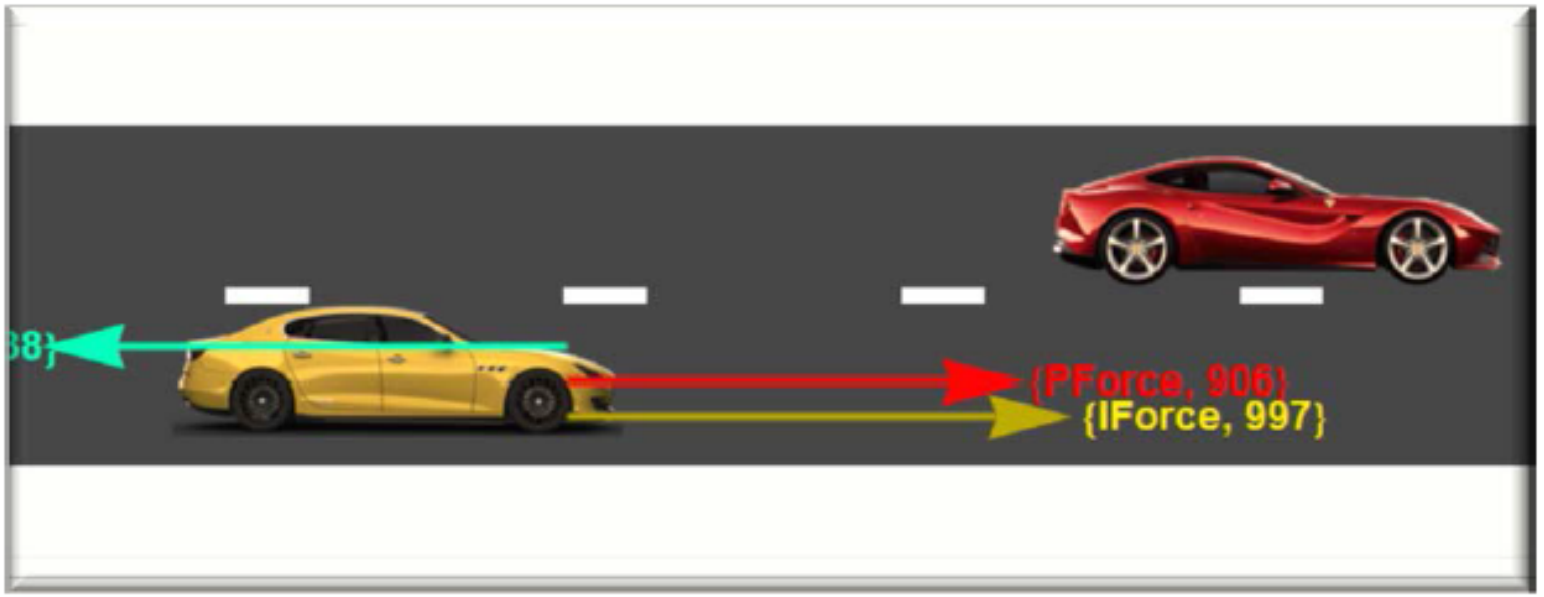
$V_o/V_i = K\,(1+t1/t2+1/(t2\,s)+t1\,s)$

$G[s]=K_p+K_i/s+K_d\,s$




38}
{PForce, 906}
{IForce, 997}


PDF



You are happily cruising at 25 m/s (~ 56 m/hr) when suddenly a car zooms from behind and passes you. Glancing sideway as he/she is passing, you suddenly recognize that person … (fill in your choice:

a. someone who owes you a ton of money and has disappeared in real life and on facebook for a long time
b. the girl/guy of your dream that you once missed and has been looking for since.
c. fill in your favorite celeb: Lindsay Lohan, Justin Bieber,… whoever
d. the one who snatched your iPhone last week
e. whatever … but you have a reason to catch up with that car.

You happen to be in control of a self-drive Google car, so you push the button of a PID controller that is based on distance and speed to make it to catch up with the other car. It takes you a full 2 seconds after the other car passed to react and start the chase. The other car is going at 40 m/s (~90 m/hr). Below is the result of the chase.

Open loop transfer function

Zeros: zeroes of numerator

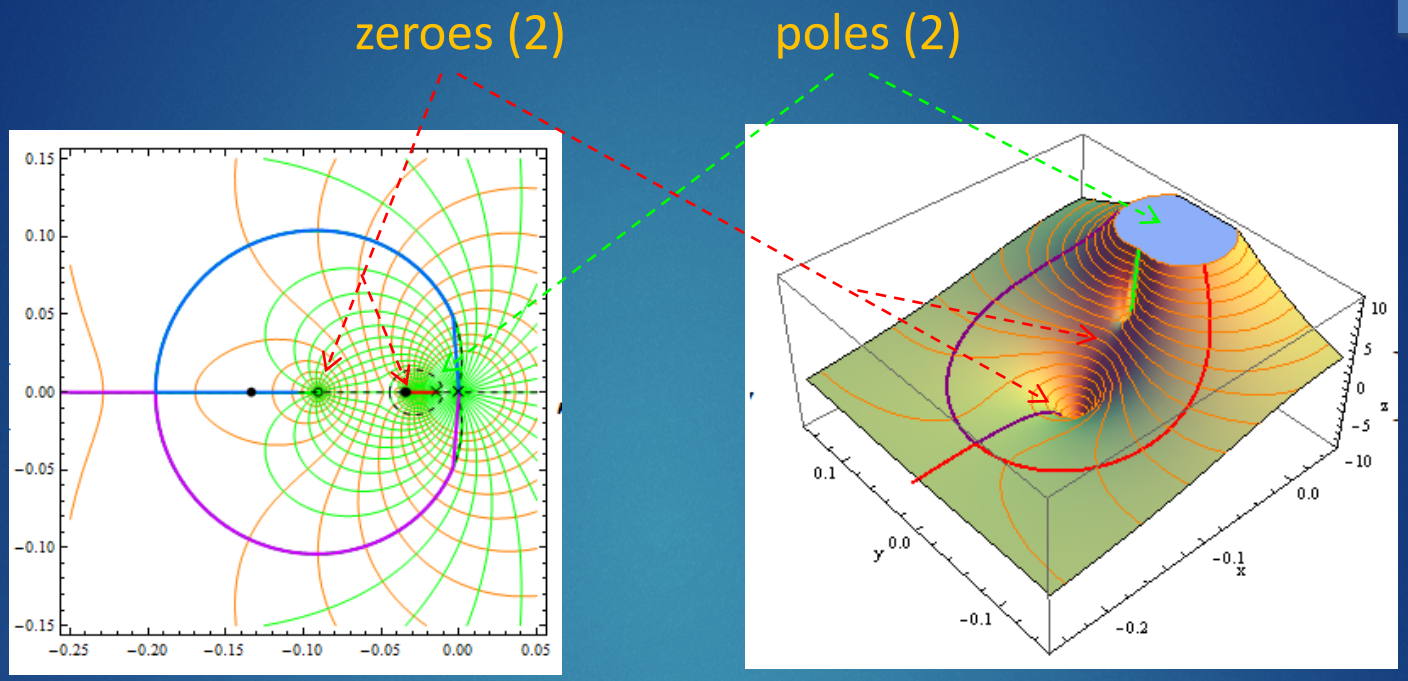
$$\mathrm{Kp}\left(1+\frac{\kappa \mathrm{I}}{s}+\mathrm{Td}\,s\right)\frac{1}{s\,(m\,s+b)}=\mathrm{Kp}\,\frac{(\kappa \mathrm{I}+s\,(1+\mathrm{Td}\,s))}{s^2\,(m\,s+b)}$$

P I D

Car (mass m) with air drag coeff. b

Poles: zeroes of denominator (s=0(2) and s=-b/m)

Root Locus Analysis and Design



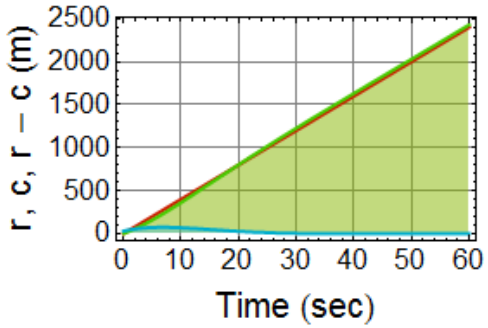
- Zeroes and poles: limiting points of root locus
  - Pole in this case is a property of the car & drag coefficient
  - Zeroes of the PID controller (relative to poles) determine root locus.
- Root locus: for designing response function (speed vs. damping, etc.)
- Contours: analysis of amplitude (constant gain curve) and phase.

  Phase contours of steepest descent (π-phase) between zeroes and poles are root loci.

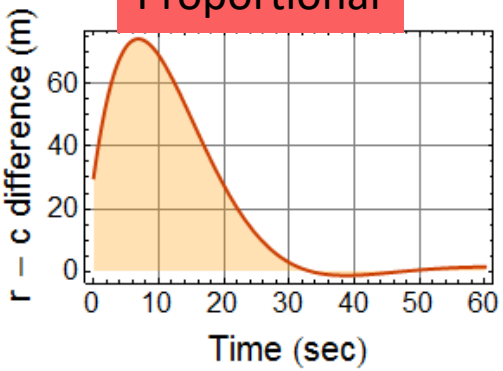
v0 Kp xI Td

speed reaction time resistance tmax

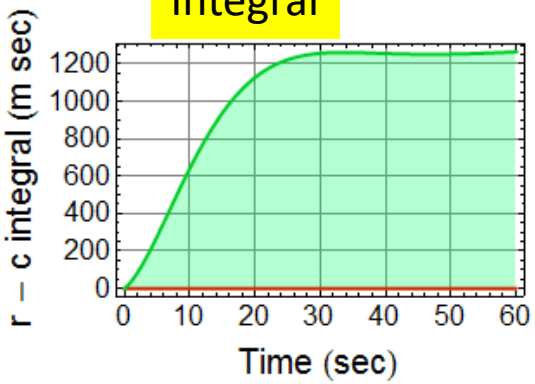
Position control



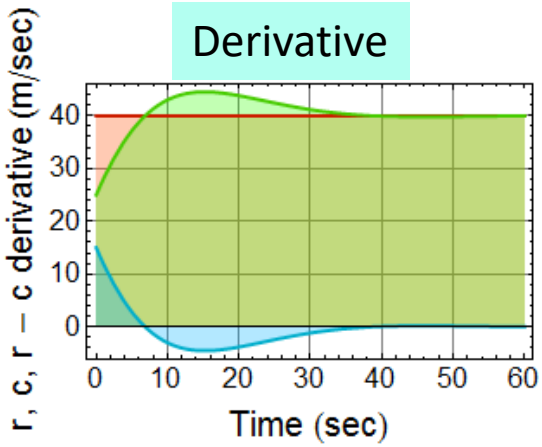
Proportional

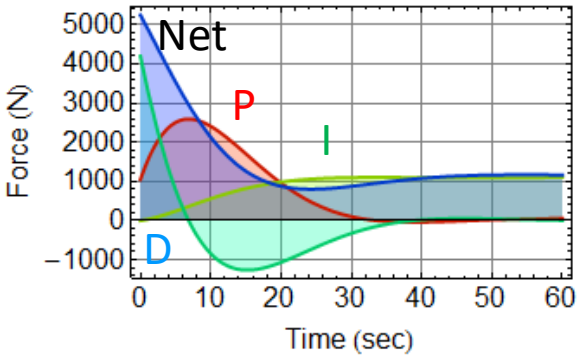


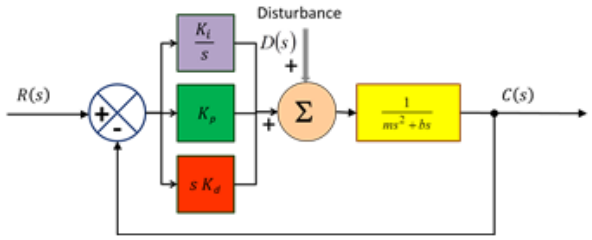
Integral



Derivative



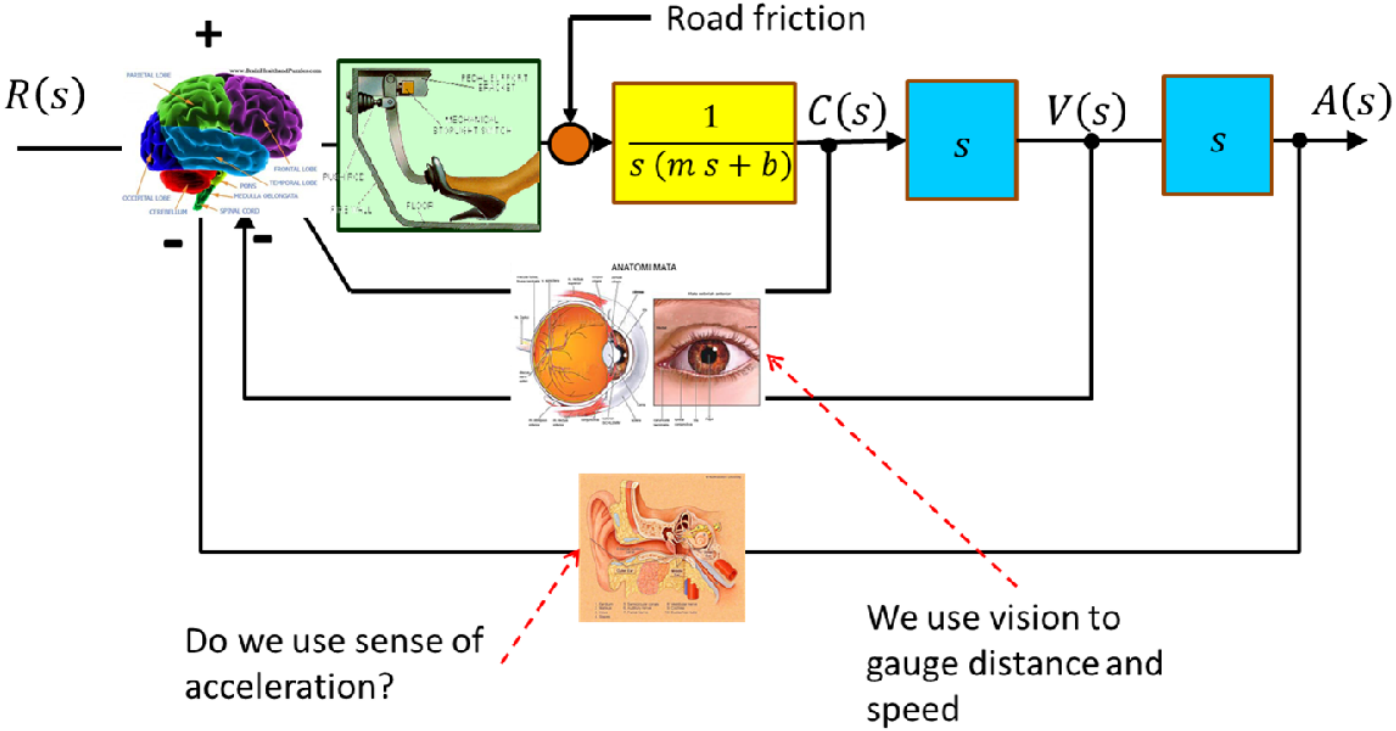




Human brain natural controller: example: braking

While we don't know how the brain compute...

We can observe what it tells the foot to do: which is the foot-brake function $F_b(x, v, a)$



Summary (takeaways)

- Root finding, polynomial zeroes and poles, and contours are common and ubiquitous in scientific engineering problems.
- Extensive algorithms have been developed and implemented in high-level application software utilities.
- However, most software packages are developed for general use, applicable to well-defined, well-behaved problems.
- The essence of computation involved these things is to formulate the problem such that these algorithms work best and yield desired accurate, precise results.

know how to use them smartly