Class Progress

Basics of Linux, gnuplot, C
Visualization of numerical data
Roots of nonlinear equations
(Midterm 1)
Solutions of systems of linear equations
Solutions of systems of nonlinear equations
Monte Carlo simulation
Interpolation of sparse data points
Numerical integration
(Midterm 2)
Solutions of ordinary differential equations

General Problem of Nonlinear Equation Roots

$$\text{Find } x \text{ for which } f(x) = 0$$

Examples:

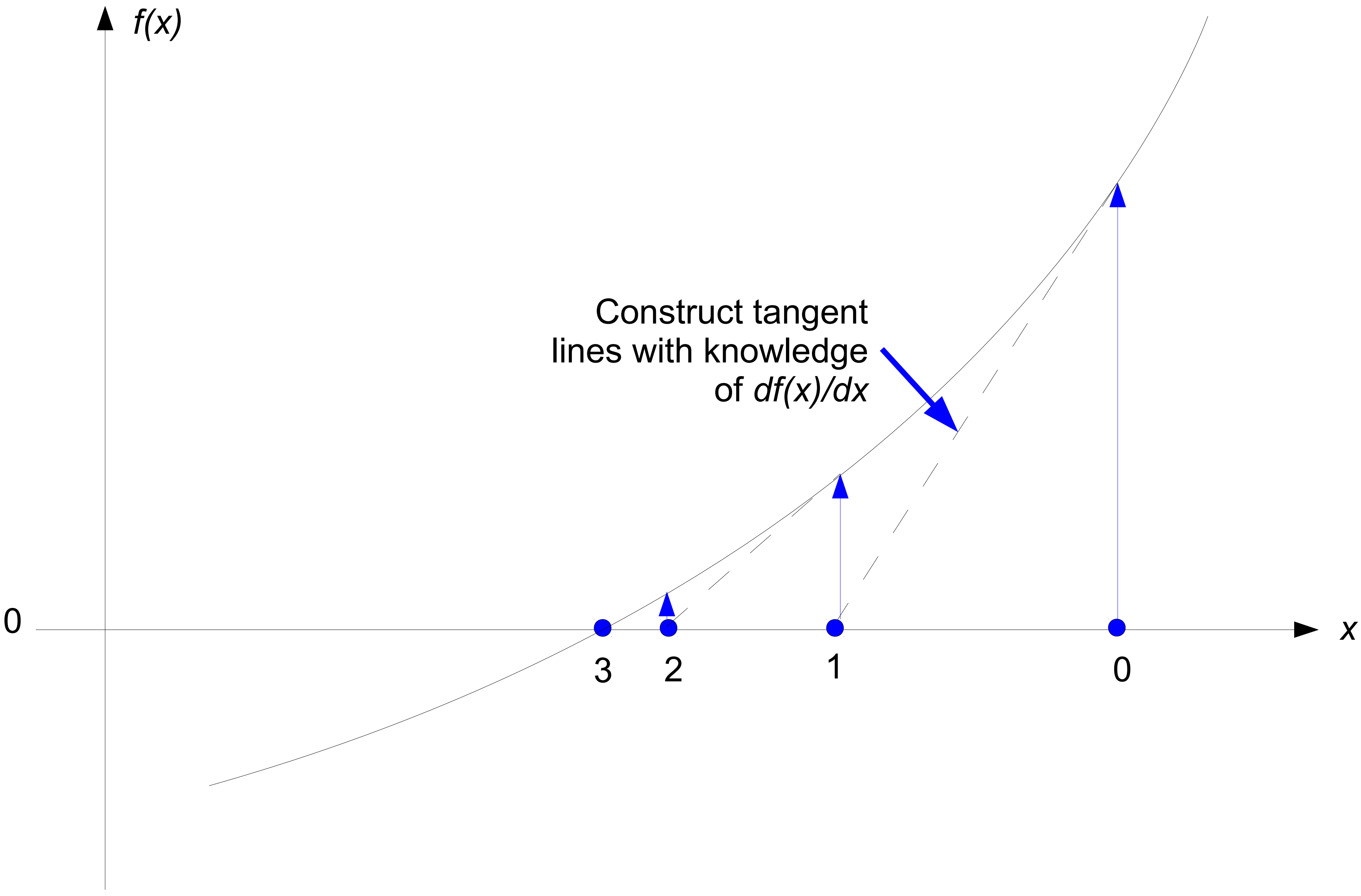
$$x^6 + 11x^5 - x^4 + 14x^3 - 23x^2 + 7x - 3 = 0$$

$$\frac{e^{k_1 x} + e^{-k_2 x}}{8} - \frac{1}{4} = 0$$

$$\sin(3x) - \cos(5x) - \sqrt{7x} + 13 = 0$$

Algorithms for finding roots are iterative, that is, they start at one or two initial guess points, x_0 or x_{-1} and x_0, and then iterate to give a series of successive values of x_i. Hopefully the sequence of x_i will converge on the root!

Newton-Raphson Algorithm for Root Finding



Newton-Raphson Algorithm

given $f(x)$ and an initial guess x_0

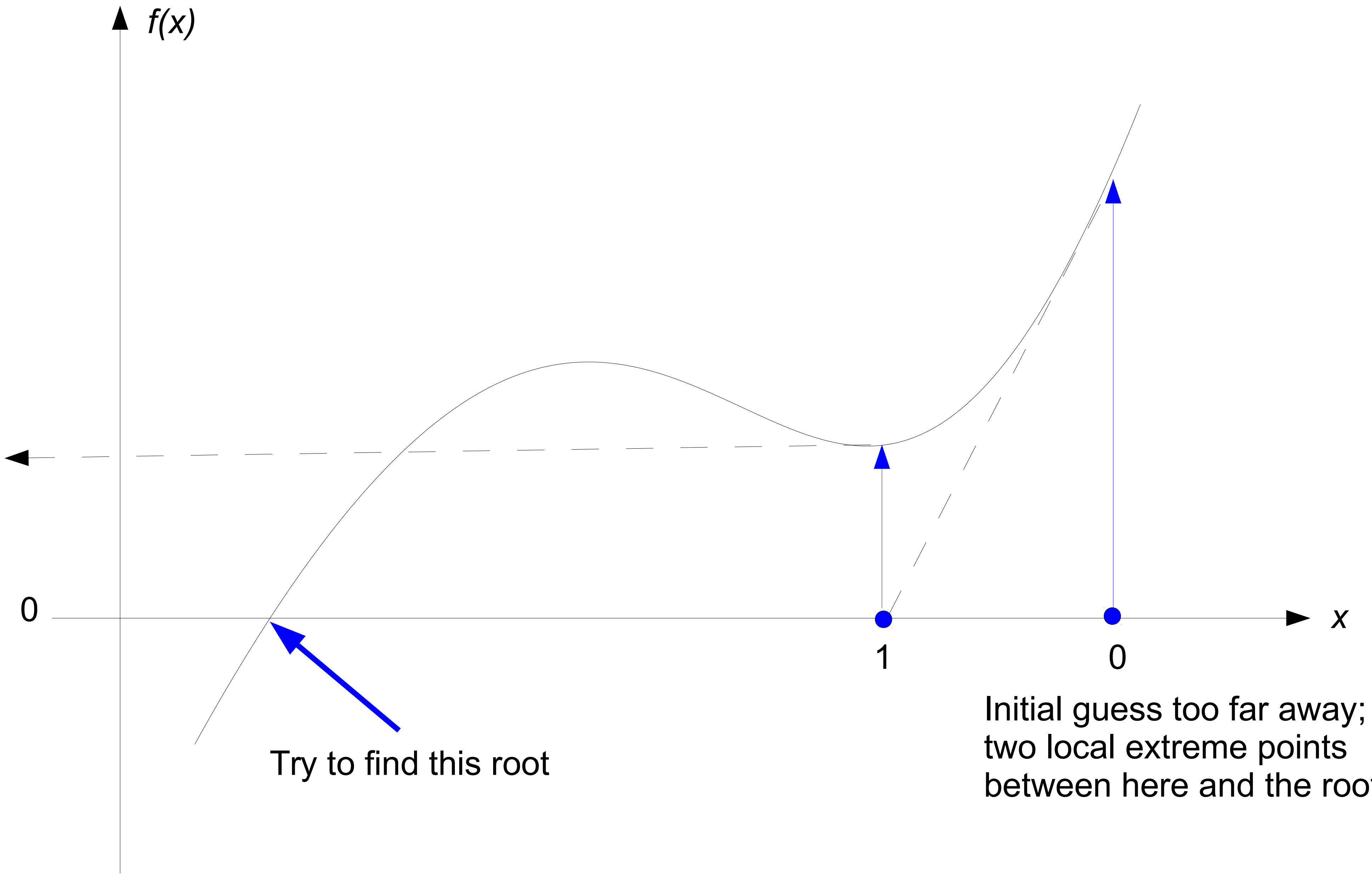
$$\frac{f(x_i)}{x_i - x_{i+1}} = f'(x_i)$$

where x_i is the trial value of x after iteration i

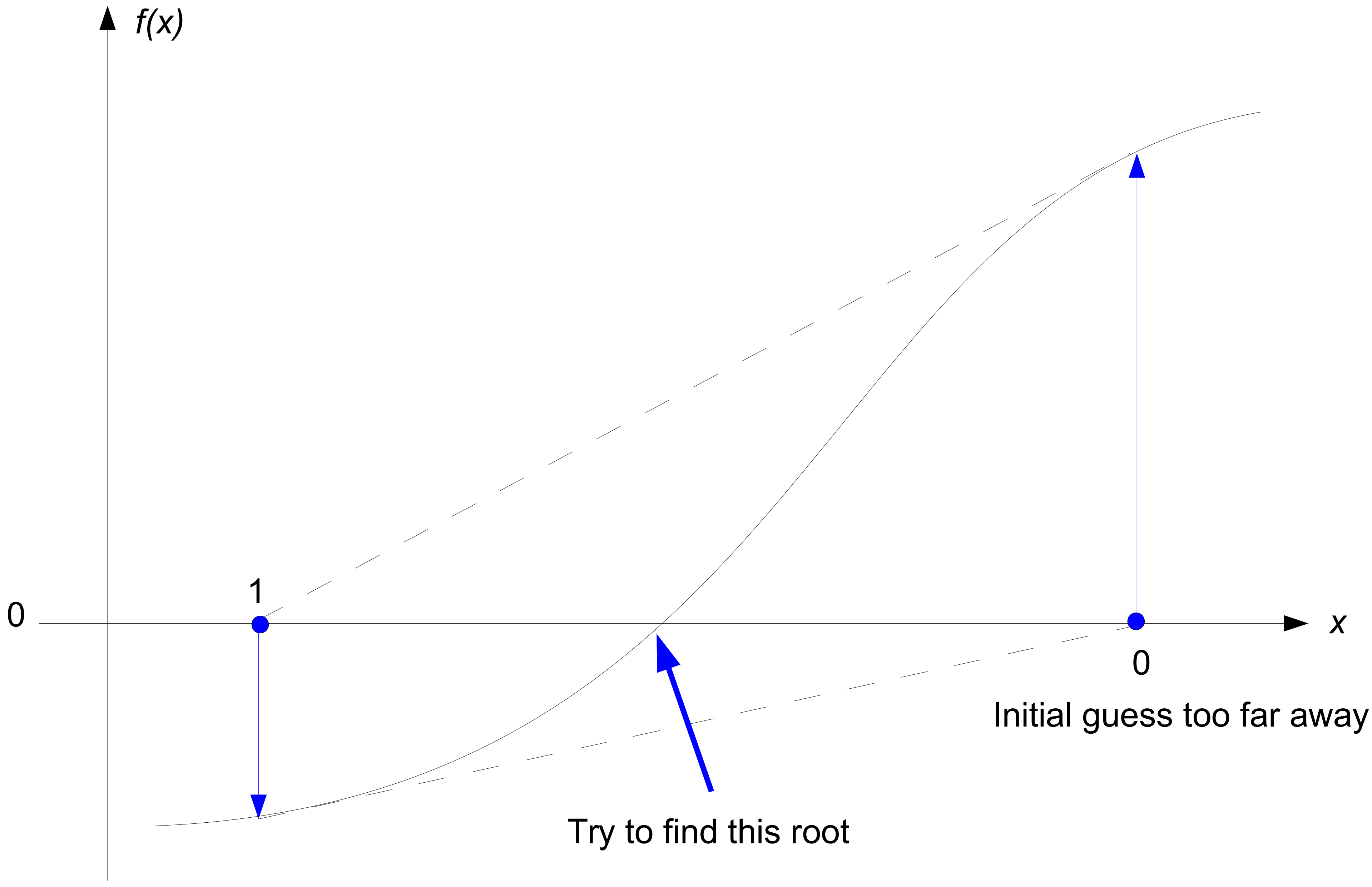
$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

But what about local extreme points when $f'(x_i)$ is near zero?

Divergence of Newton-Raphson Algorithm



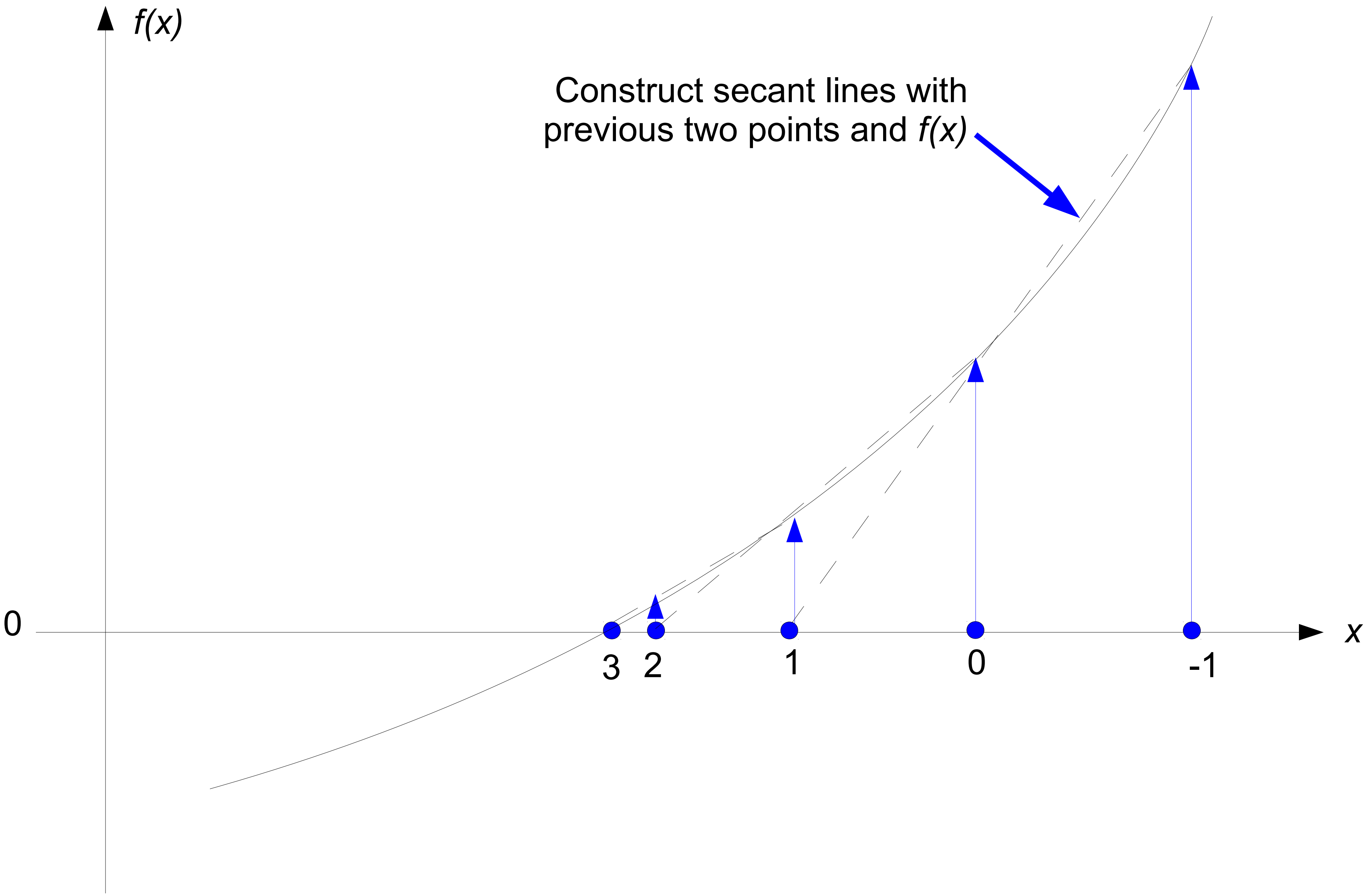
Oscillation of Newton-Raphson Algorithm



Newton-Raphson Algorithm

- Requires coding of both the function being solved and its derivative
- Requires any single point close to a root to start
- Minimal storage requirements for each iteration
- Usually will converge if starting point is sufficiently close to root, but also easily divergent for sharply nonlinear functions around local extrema
- Fastest convergence rate

Secant Algorithm for Root Finding



Secant Algorithm

given $f(x)$ and initial guesses x_0 and x_{-1}

$$\frac{x_{i-1} - x_i}{x_{i-1} - x_{i+1}} = \frac{f(x_{i-1}) - f(x_i)}{f(x_{i-1})}$$

where x_i is the trial value of x after iteration i

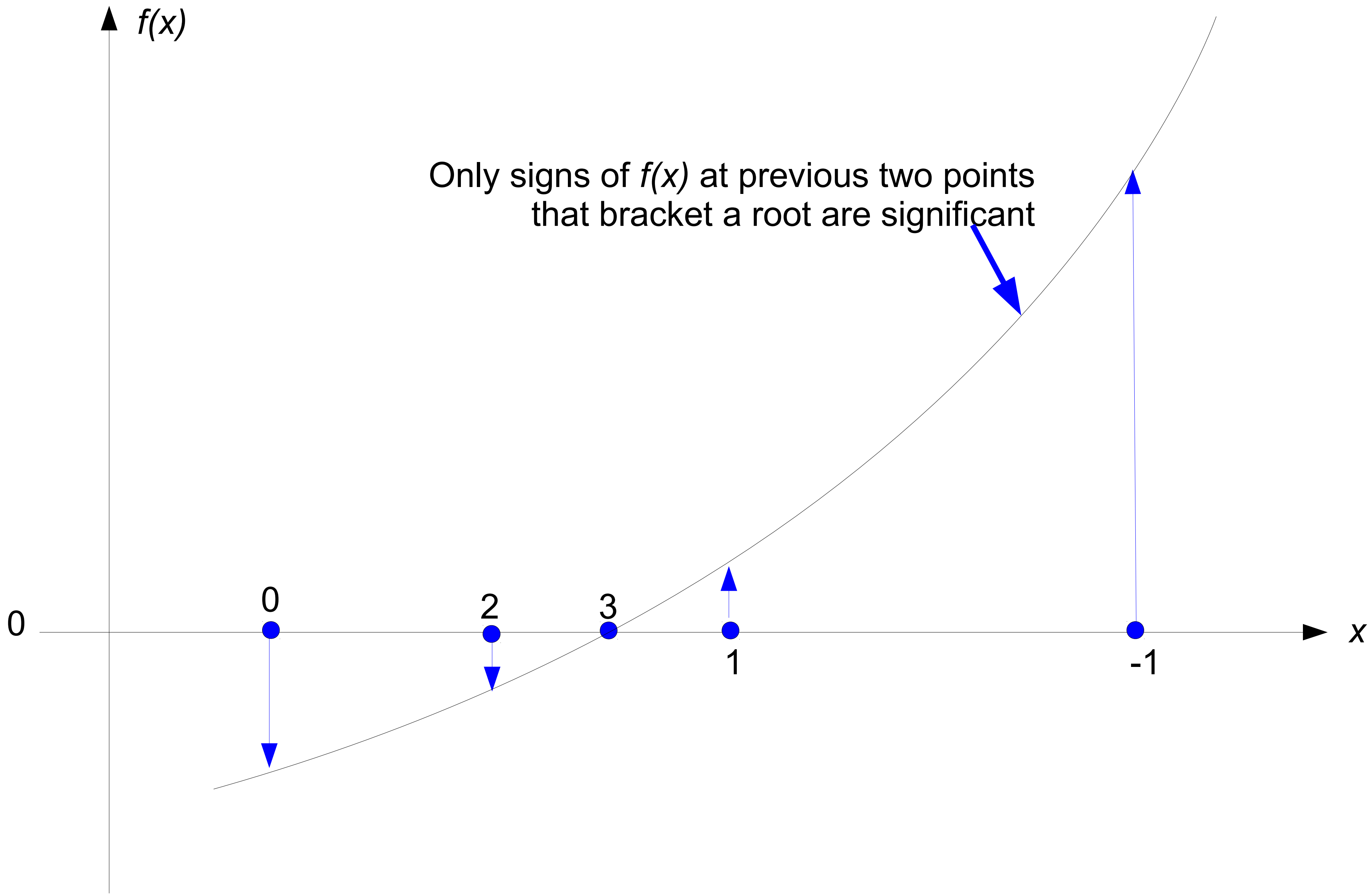
$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} \cdot f(x_i)$$

But what about when $f(x_{i-1}) - f(x_i)$ is near zero?

Secant Algorithm

- Requires any two points close to a root to start
- Coding of derivative not required
- Some small additional storage requirements over Newton-Raphson, significant when systems of nonlinear equations are to be solved
- Usually will converge if starting points are sufficiently close to root
- Sharply nonlinear functions, or starting points too far from root, can cause divergence
- Fast convergence rate; useful if function is expensive to evaluate

Bisection Algorithm for Root Finding



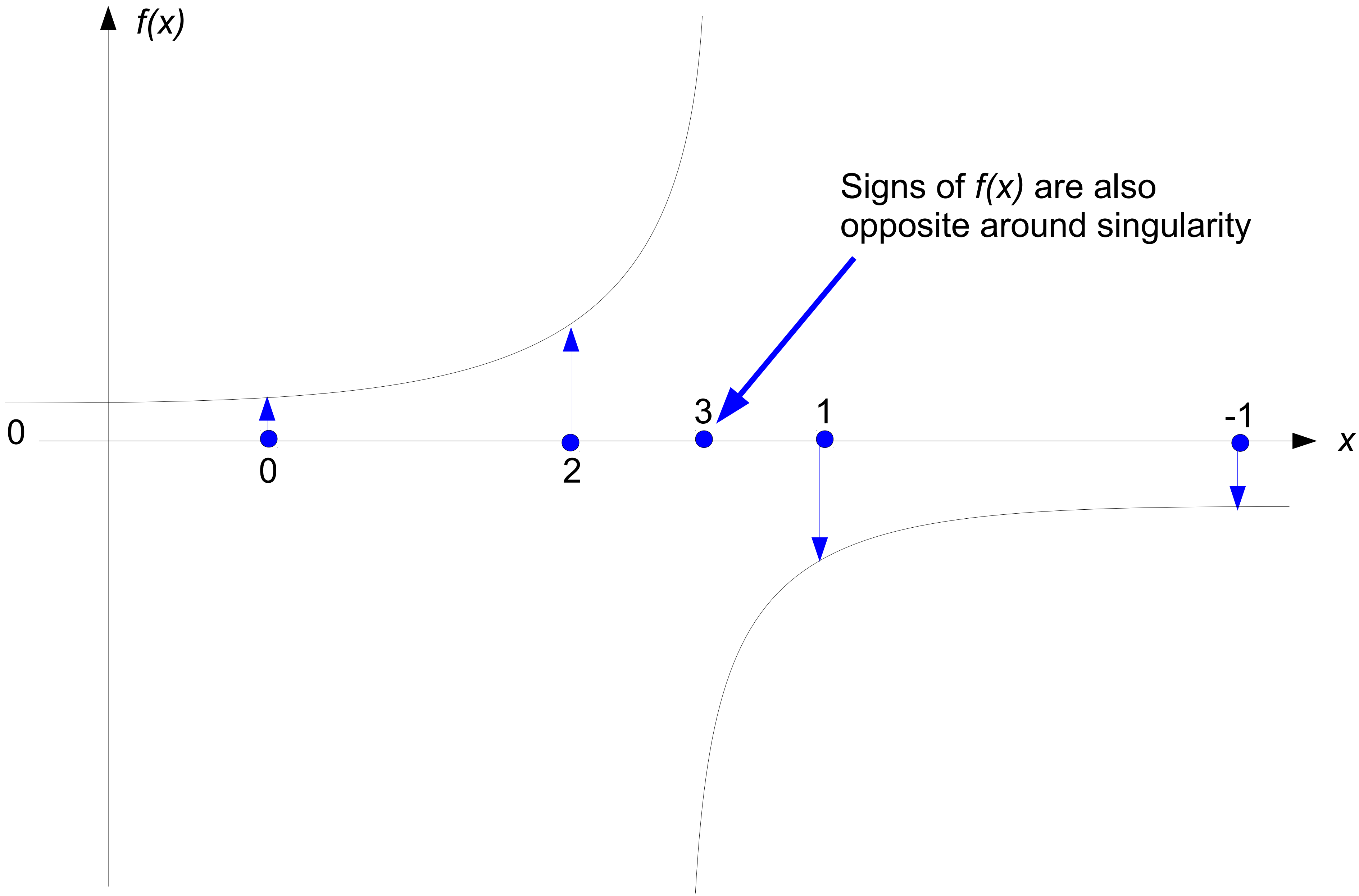
Bisection Algorithm

given $f(x)$ and initial guesses x_0 and x_{-1} that **must bracket the root**

$$x_{i+1} = \frac{x_i + x_{i-1}}{2}$$

where x_i is the trial value of x after iteration i
For the two points used in the next iteration, retain x_{i+1} and either x_i or x_{i-1} such that the root remains bracketed

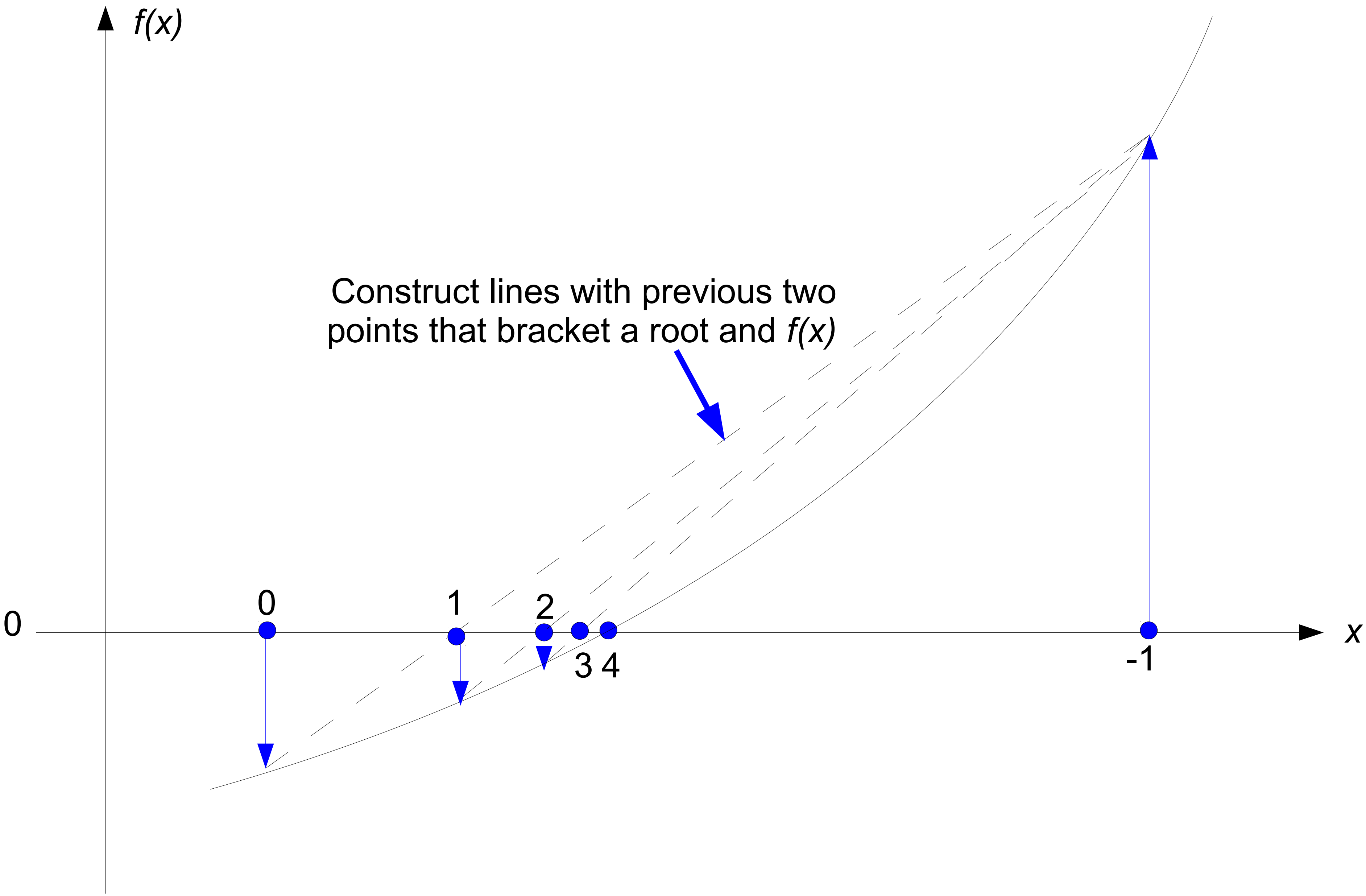
Bisection Algorithm Stuck at Singularities



Bisection Algorithm

- Requires two points that bracket a root to start, not necessarily very close to root
- Must converge on a root or a singularity, even for sharply nonlinear functions
- May be trapped by a singularity in the function
- Slowest convergence rate

Regula Falsi Algorithm for Root Finding



Regula Falsi Algorithm

given $f(x)$ and initial guesses x_0 and x_{-1} that **must bracket the root**

$$\frac{x_{i-1}-x_i}{x_{i-1}-x_{i+1}}=\frac{f(x_{i-1})-f(x_i)}{f(x_{i-1})}$$

where x_i is the trial value of x after iteration i

$$x_{i+1}=x_i-\frac{x_i-x_{i-1}}{f(x_i)-f(x_{i-1})}\cdot f(x_i)$$

Look familiar? This is the same expression as for the secant algorithm!
For the two points used in the next iteration, retain x_{i+1} and
either x_i or x_{i-1} such that the root remains bracketed
Another version of this expression that is computationally more balanced is

$$x_{i+1}=\frac{f(x_i)x_{i-1}-f(x_{i-1})x_i}{f(x_i)-f(x_{i-1})}$$

Regula Falsi Algorithm

- Requires two points that bracket a root to start
- New trial point is calculated identically with the Secant Algorithm
- As algorithm proceeds, the two active points maintain bracketing, so almost always will converge to a root
- Moderate convergence rate
- Good first choice algorithm for real world functions with unknown nonlinearity

Root Finding Algorithms

Different vehicles are appropriate for different road surfaces; different root finding algorithms are appropriate for different function “surfaces”

Robustness ← → *Convergence rate*

Bisection | Regula Falsi | Secant | Newton-Raphson

Convergence Rates of Root Finding Algorithms

given $f(x)$ where r is the desired root, so $f(r)=0$
and the sequence of x values determined by the algorithm is $x_{0,} x_{1,} x_{2,} \cdots, x_n$

For Bisection: $|x_{n+1}-r| \leqslant C|x_n-r|$
For Secant: $|x_{n+1}-r| \leqslant C|x_n-r|^{\alpha}$ where $\alpha \approx 1.62$
For Newton-Raphson: $|x_{n+1}-r| \leqslant C|x_n-r|^2$
For Regula Falsi: $|x_{n+1}-r| \leqslant C|x_n-r|^{\alpha}$ where $1 < \alpha < 1.62$

Convergence Rate of Newton-Raphson

New error $\left(x_{n+1}-r\right) = \left[x_n - \frac{f\left(x_n\right)}{f'\left(x_n\right)}\right] - r = \frac{\left(x_n - r\right) f'\left(x_n\right) - f\left(x_n\right)}{f'\left(x_n\right)}$

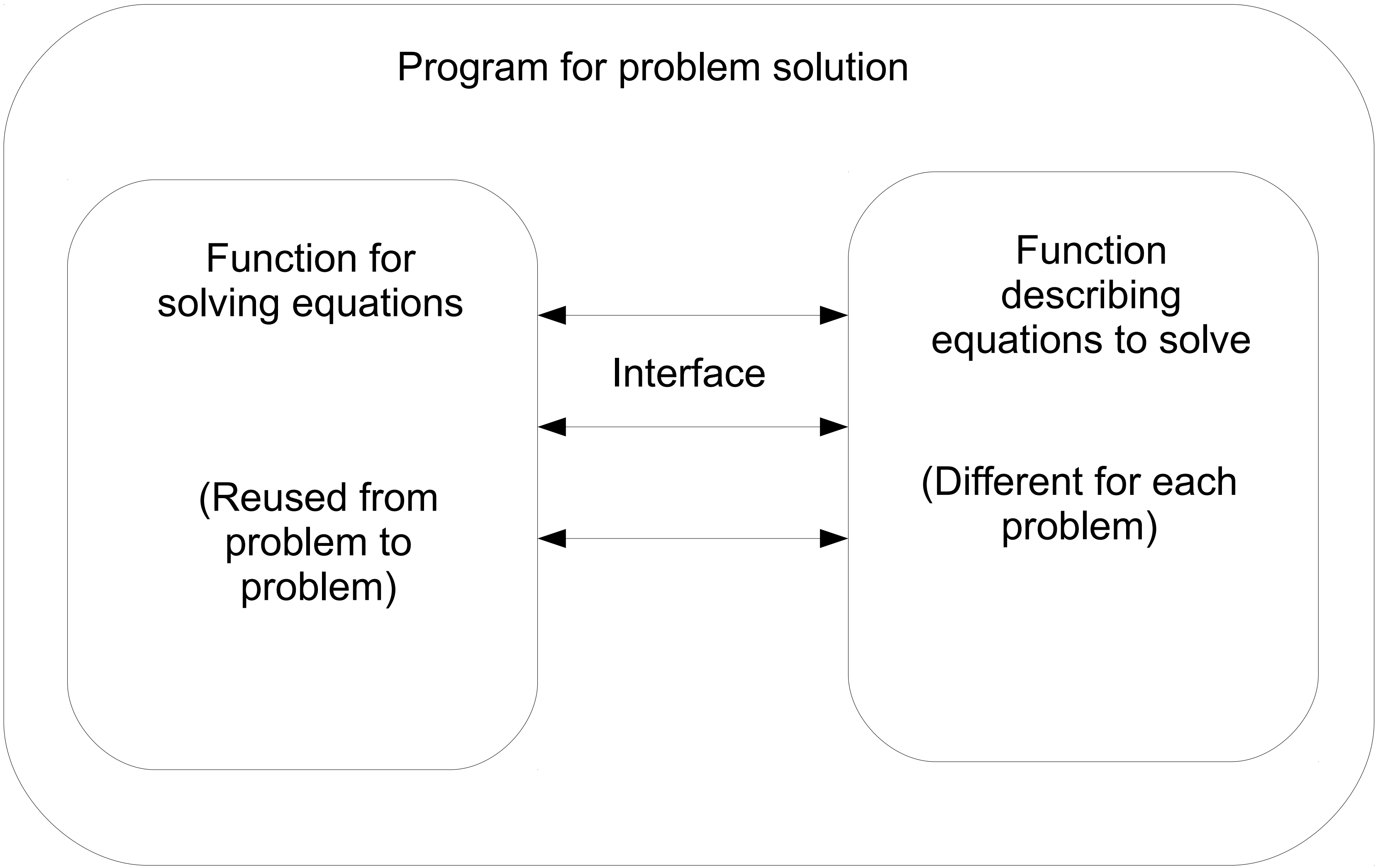
From Taylor series $f\left(r\right) = 0 = f\left(x_n\right) - \left(x_n - r\right) f'\left(x_n\right) + \frac{\left(x_n - r\right)^2}{2} f''\left(\xi_n\right)$

or $\left(x_n - r\right) f'\left(x_n\right) - f\left(x_n\right) = \frac{\left(x_n - r\right)^2}{2} f''\left(\xi_n\right)$

where ξ_n lies between r and x_n

plug into new error expression: $\left(x_{n+1} - r\right) = \frac{f''\left(\xi_n\right)}{2 f'\left(x_n\right)} \left(x_n - r\right)^2$

Reusing Program Code



Newton-Raphson Function in roots.c

```
int newton_raphson(double (*func)(double x),double (*deriv)(double x),double
x,int nmax,double tol,double min) {
  double fx,fp,d;
  int n;

  fx = (*func)(x);
  printf("0 %.8g %.8g\n",x,fx);
  n = 1;
  while (n <= nmax) {
    fp = (*deriv)(x);
    if (fabs(fp) < min) {
      fprintf(stderr,"newton_raphson: Derivative too small\n");
      return(2);
    }
    d =  fx / fp;
    x = x - d;
    fx = (*func)(x);
    printf("%d %.8g %.8g\n",n,x,fx);
    if (fabs(d) < tol) break;
    n++;
  }
  if (n > nmax) {
    fprintf(stderr,"newton_raphson: Iteration limit of %d reached\n",nmax);
    return(1);
  }
  return(0);
}
```

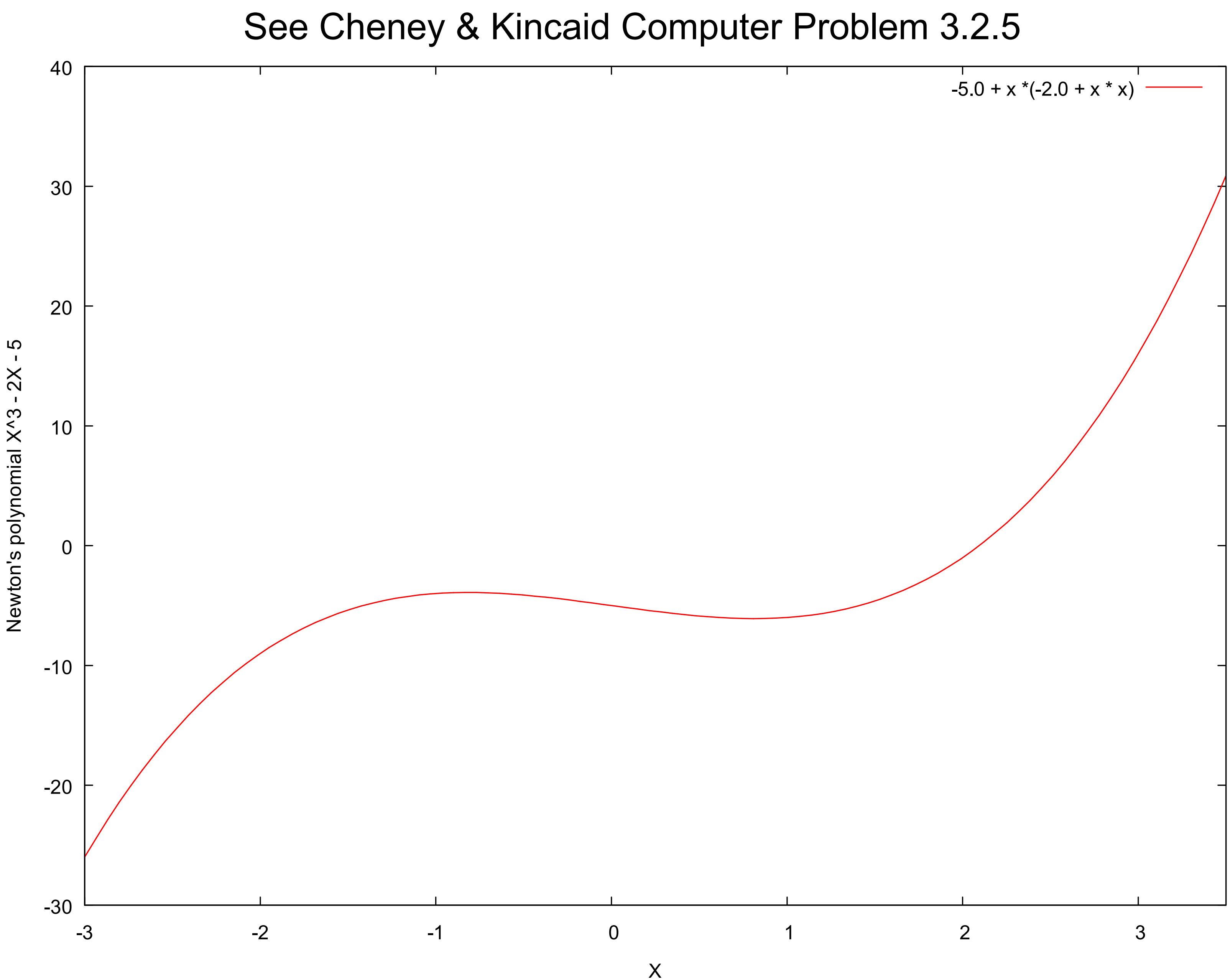
Bisection Function in roots.c

```
int bisection(double (*func)(double x),double a,double b,int nmax,double tol) {
  double fa,fb,c,fc,d;
  int n;

  fa = (*func)(a);
  fb = (*func)(b);
  if (fa * fb > 0.0) {
    fprintf(stderr,"bisection: Root not bracketed by initial points\n");
    return(2);
  }
  printf("0 %.8g %.8g\n",b,fb);
  printf("1 %.8g %.8g\n",a,fa);
  n = 2;
  while (n <= nmax) {
    d = 0.5 * (b - a);
    if (fabs(d) < tol) break;
    c = a + d;
    fc = (*func)(c);
    printf("%d %.8g %.8g\n",n,c,fc);
    if (fa * fc < 0.0) {
      b = c;
      fb = fc;
    }
    else {
      a = c;
      fa = fc;
    }
    n++;
  }
  if (n > nmax) {
    fprintf(stderr,"bisection: Iteration limit of %d reached\n",nmax);
    return(1);
  }
  return(0);
}
```

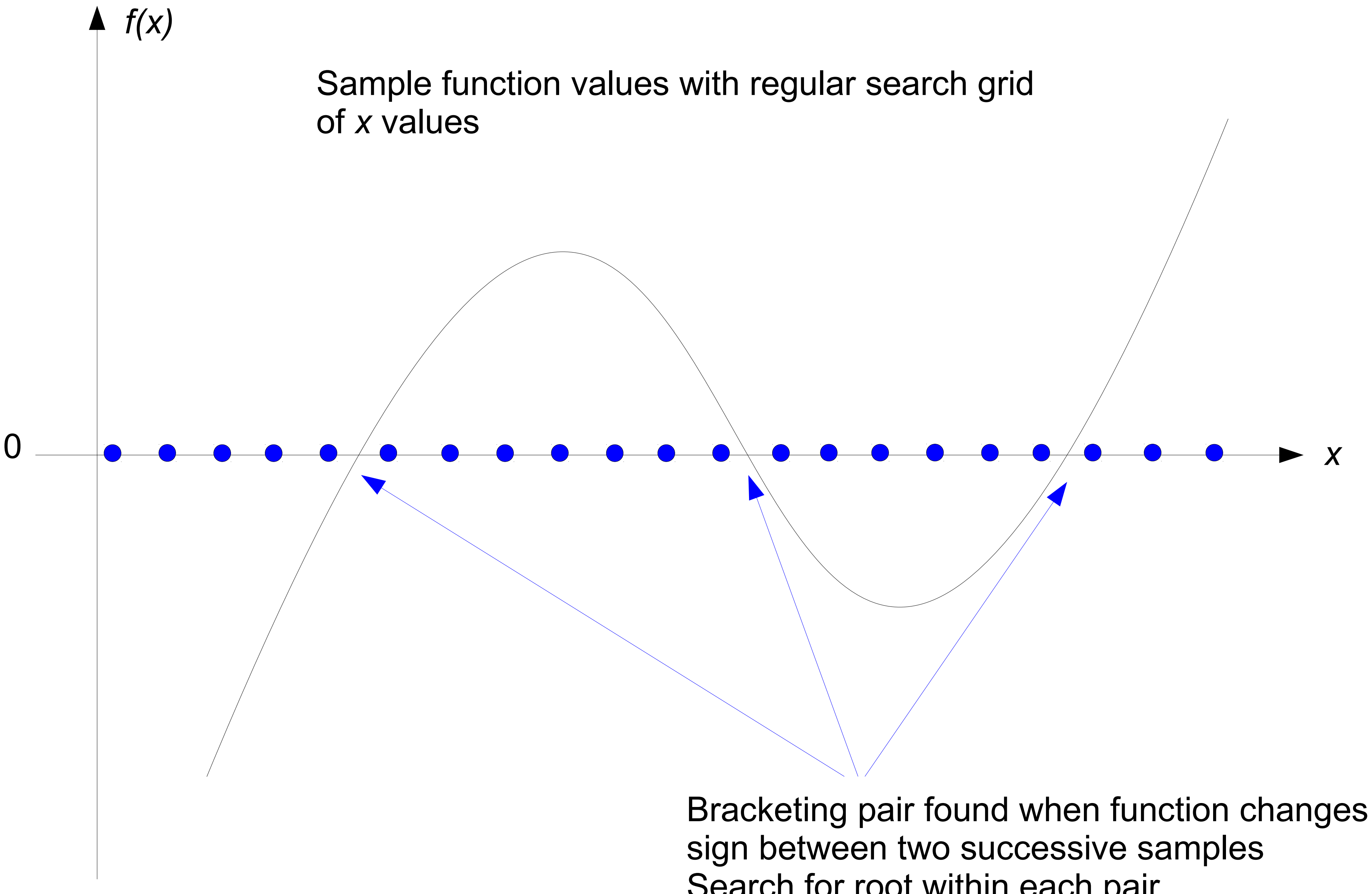
Header file roots.h

```
int newton_raphson(double (*func)(double x),double (*deriv)(double x),double
x,int nmax,double tol,double min);
int regula_falsi(double (*func)(double x),double a,double b,int nmax,double
tol);
int bisection(double (*func)(double x),double a,double b,int nmax,double tol);
```

Newton's Polynomial for Root Finding

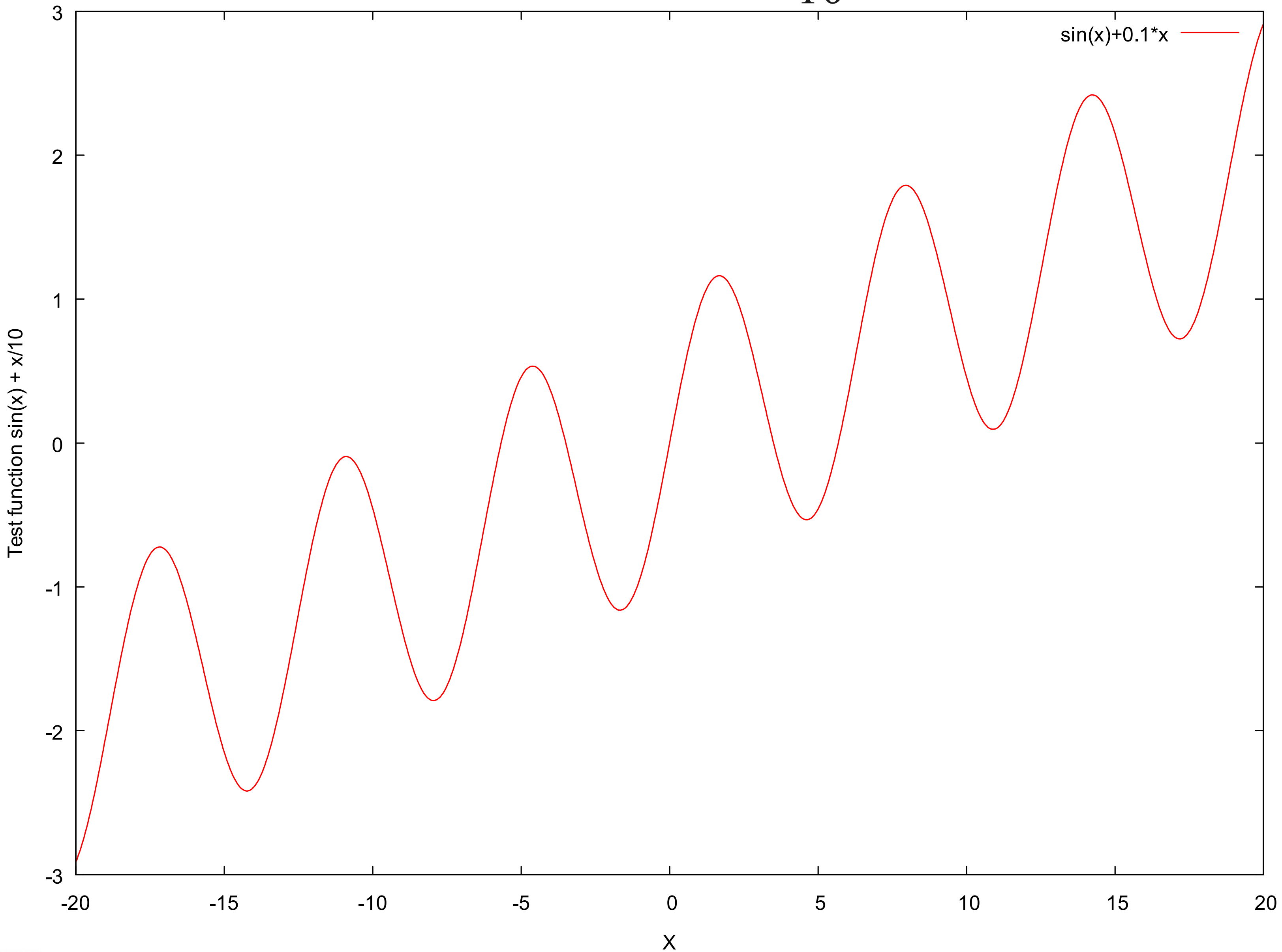


Finding Initial Pairs of Bracketing Points



Test Function for Root Finding

$$f(x) = \sin(x) + \frac{x}{10}$$



Bisection Functions in sweep_roots.c

```
int bisection(double (*func)(double x),double a,double b,int nmax,double tol) {
  ...      (same as in roots.c except without printf() calls)
}

int sweep_bisection(double (*func)(double x),double xstart,double xstop,double
xinc,int nmax,double tol) {
  double a,b,fa,fb;
  int retval;

  xstop = xstop + (xinc * 0.5);
  a = xstart;
  fa = (*func)(a);
  b = a + xinc;
  while (((xinc > 0.0) && (b < xstop)) || ((xinc < 0.0) && (b > xstop))) {
    fb = (*func)(b);
    if ((fa * fb) < 0.0) {   /*root bracketed, converge with bisection*/
      retval = bisection(func,a,b,nmax,tol);
      if (retval > 0) return(retval);
    }
    a = b;
    fa = fb;
    b = b + xinc;
  }
  return(0);
}
```

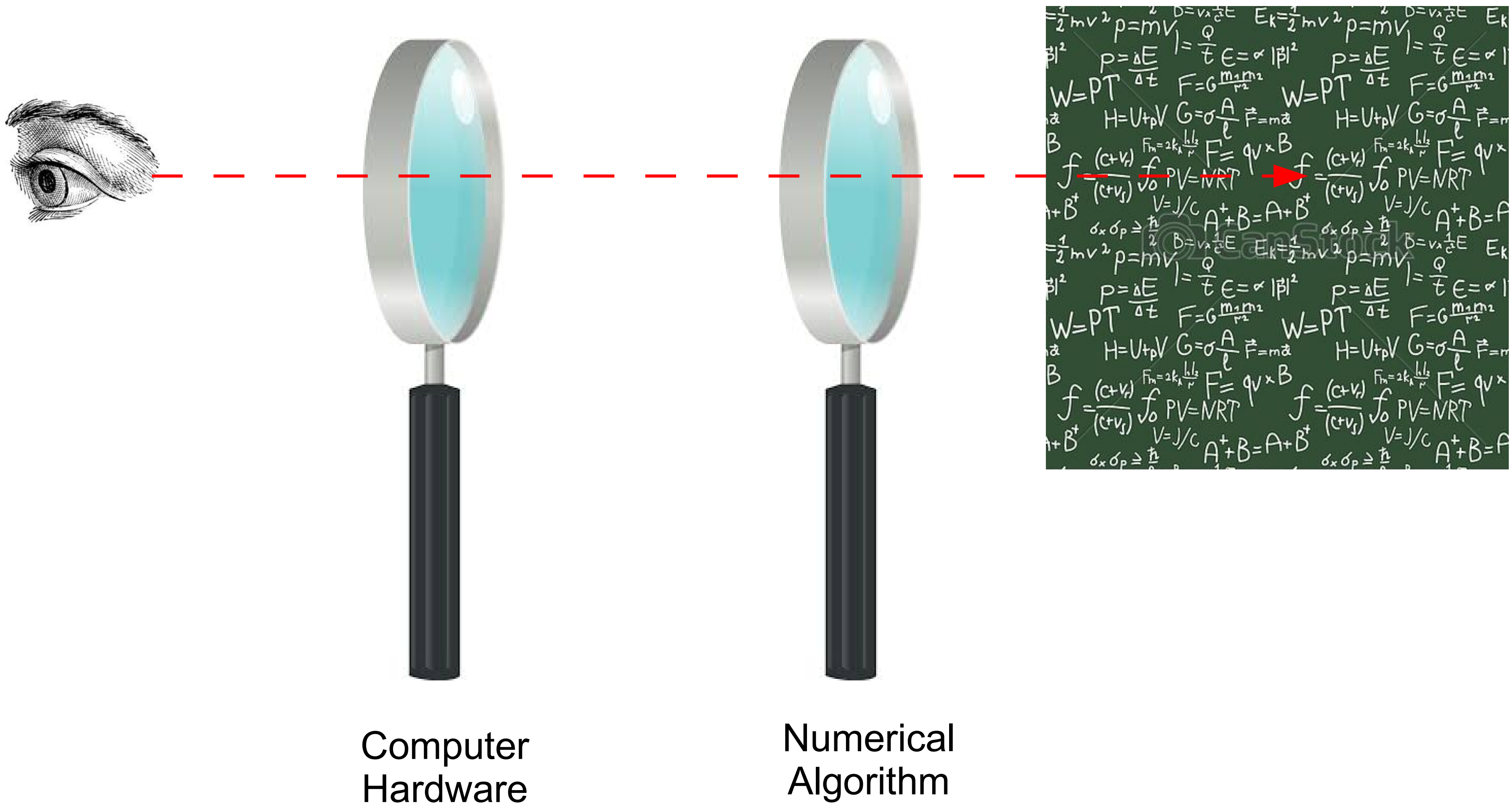
Newton-Raphson Functions in sweep_roots.c

```
int newton_raphson(double (*func)(double x),double (*deriv)(double x),double x,int
nmax,double tol,double min) {
  ...     (same as in roots.c except without printf() calls)
}

int sweep_newton_raphson(double (*func)(double x),double (*deriv)(double x),double
xstart,double xstop,double xinc,int nmax,double tol,double min) {
  double a,b,fa,fb;
  int retval;

  xstop = xstop + (xinc * 0.5);
  a = xstart;
  fa = (*func)(a);
  b = a + xinc;
  while (((xinc > 0.0) && (b < xstop)) || ((xinc < 0.0) && (b > xstop))) {
    fb = (*func)(b);
    if ((fa * fb) < 0.0) {   /*root bracketed, converge with newton_raphson*/
      if (fabs(fa) < fabs(fb)) {
        retval = newton_raphson(func,deriv,a,nmax,tol,min);
      }
      else {
        retval = newton_raphson(func,deriv,b,nmax,tol,min);
      }
      if (retval > 0) return(retval);
    }
    a = b;
    fa = fb;
    b = b + xinc;
  }
  return(0);
}
```

Header file sweep_roots.h

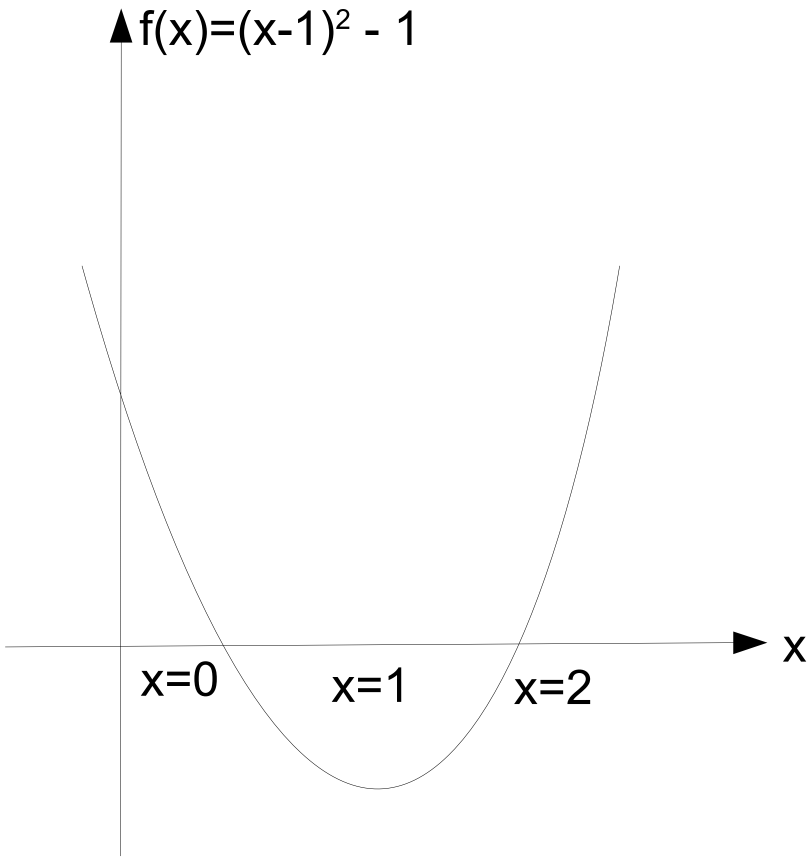
```
int sweep_newton_raphson(double (*func)(double x),double (*deriv)(double
x),double xstart,double xstop,double xinc,int nmax,double tol,double min);
int sweep_regula_falsi(double (*func)(double x),double xstart,double
xstop,double xinc,int nmax,double tol);
int sweep_bisection(double (*func)(double x),double xstart,double xstop,double
xinc,int nmax,double tol);
```

Solving Physics Problems Numerically



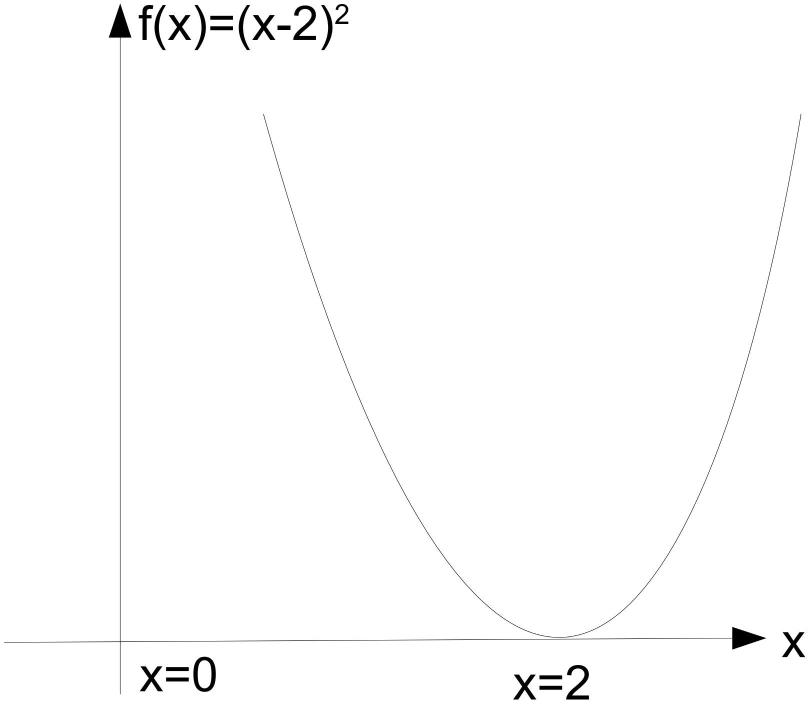
We have studied the limitations of the lens of computer hardware through which we must look for numerical results. What about the limitations of our numerical algorithms?

Behavior of N-R Algorithm at Multiple Roots



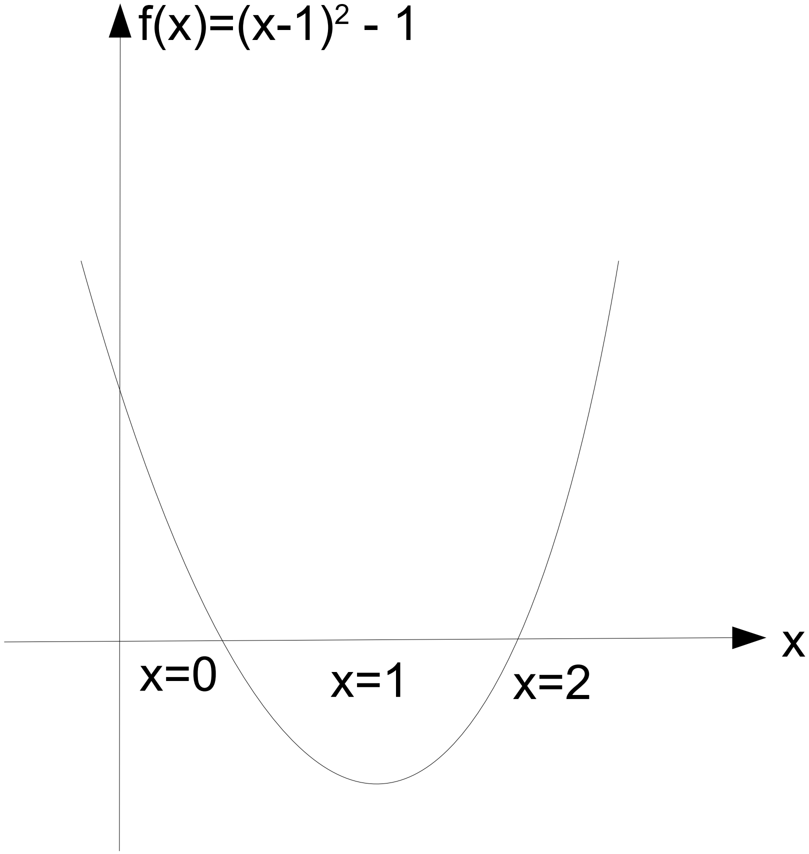

```
0 3 3
1 2.25 0.5625
2 2.025 0.050624999999999982
3 2.000304878048781 0.000609840481859375
4 2.000000046461147 9.292229690631253e-08
5 2.000000000000001 1.776356839400025e-15
6 2 0
```

Behavior of N-R Algorithm at Multiple Roots



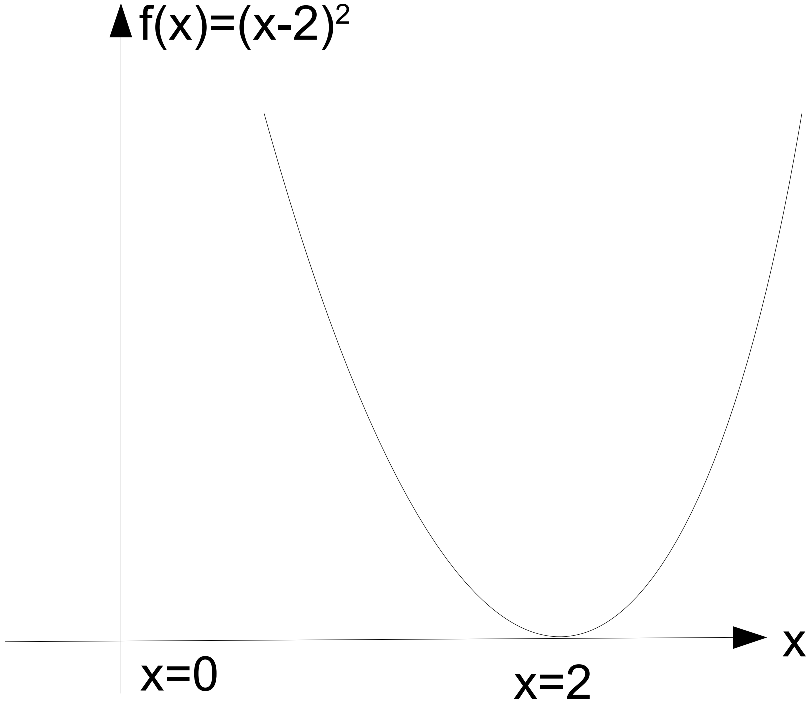

```
0 3 1
1 2.5 0.25
2 2.25 0.0625
3 2.125 0.015625
4 2.0625 0.00390625
5 2.03125 0.0009765625
6 2.015625 0.000244140625
7 2.0078125 6.103515625e-05
8 2.00390625 1.52587890625e-05
9 2.001953125 3.814697265625e-06
10 2.0009765625 9.5367431640625e-07
11 2.00048828125 2.384185791015625e-07
12 2.000244140625 5.960464477539062e-08
13 2.0001220703125 1.490116119384766e-08
14 2.00006103515625 3.725290298461914e-09
15 2.000030517578125 9.313225746154785e-10
16 2.000015258789062 2.328306436538696e-10
17 2.000007629394531 5.820766091346741e-11
18 2.000003814697266 1.455191522836685e-11
19 2.000001907348633 3.637978807091713e-12
20 2.000000953674316 9.094947017729282e-13
21 2.000000476837158 2.273736754432321e-13
22 2.000000238418579 5.684341886080801e-14
23 2.00000011920929 1.4210854715202e-14
24 2.000000059604645 3.552713678800501e-15
25 2.000000029802322 8.881784197001252e-16
26 2.000000014901161 2.220446049250313e-16
27 2.000000007450581 5.551115123125783e-17
28 2.00000000372529 1.387778780781446e-17
29 2.000000001862645 3.469446951953614e-18
30 2.000000000931323 8.673617379884035e-19
```

Behavior of Secant Algorithm at Multiple Roots




```
0 4 8
1 3 3
2 2.4 0.9599999999999997
3 2.117647058823529 0.2491349480968859
4 2.018691588785047 0.03773255306140229
5 2.00102933607823 0.002059731689221219
6 2.000009526032265 1.905215527485849e-05
7 2.000000004900199 9.80039785057011e-09
8 2.000000000000024 4.707345624410664e-14
```

Behavior of Secant Algorithm at Multiple Roots

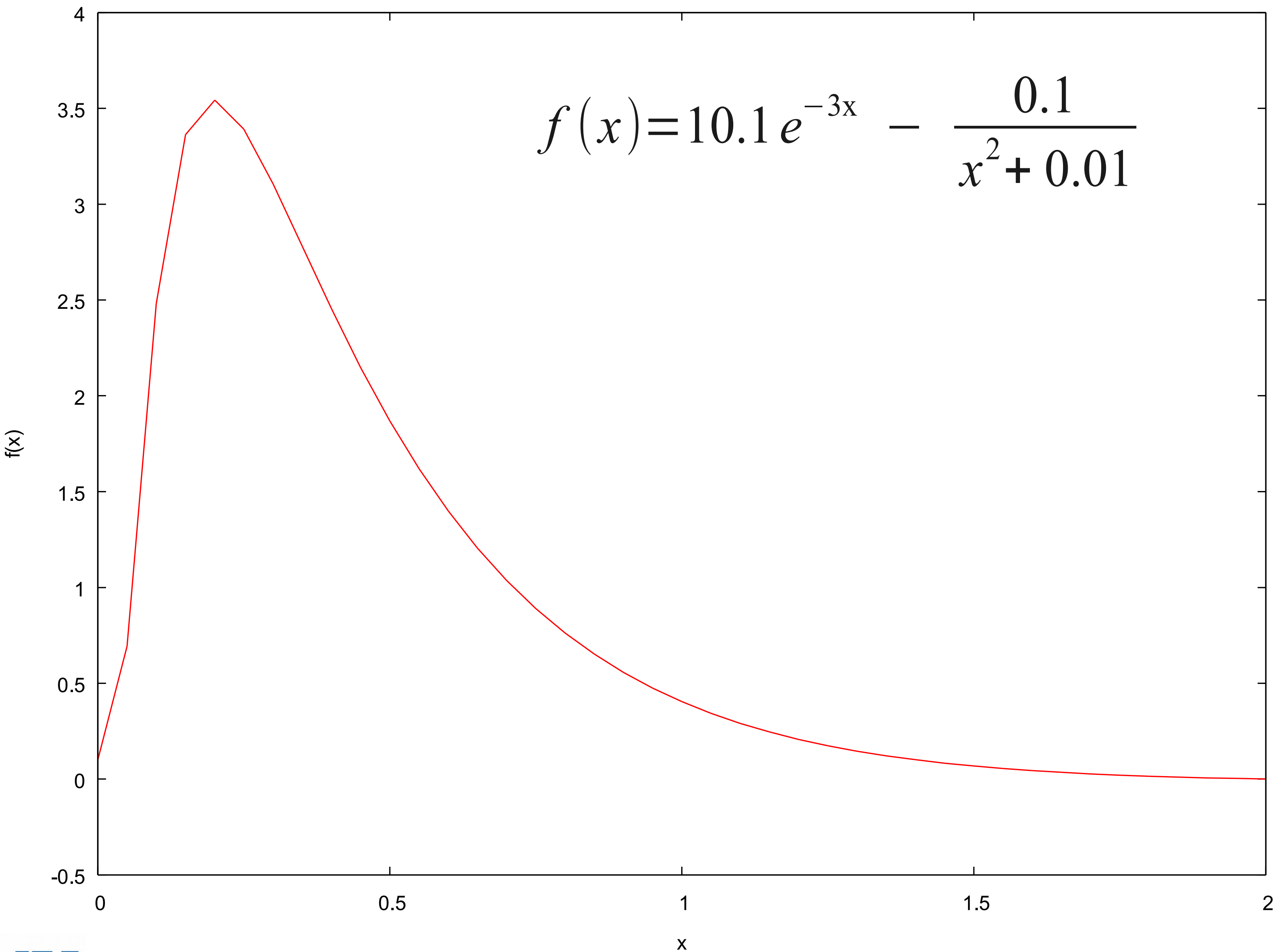



```
0 4 4
1 3 1
2 2.666666666666667 0.4444444444444443
3 2.4 0.1599999999999999
4 2.25 0.0625
5 2.153846153846154 0.02366863905325441
6 2.095238095238095 0.009070294784580518
7 2.058823529411765 0.003460207612456775
8 2.036363636363637 0.001322314049586794
9 2.02247191011236 0.0005049867440979699
10 2.013888888888889 0.0001929012345678999
11 2.008583690987125 7.367975096244472e-05
12 2.005305039787798 2.814344715012285e-05
13 2.00327868852459 1.074979844127896e-05
14 2.002026342451874 4.106063732267847e-06
15 2.001252348152787 1.568375895788087e-06
16 2.00077399380805 5.990664148993789e-07
17 2.000478354460655 2.288229900287754e-07
18 2.00029563932003 8.740260754764651e-08
19 2.000182715147086 3.338482497454056e-08
20 2.000112924171419 1.275186849068122e-08
21 2.000069790976027 4.870780334769662e-09
22 2.000043133195307 1.860472537393645e-09
23 2.00002665778074 7.106372739678463e-10
24 2.000016475414562 2.714392850029487e-10
25 2.000010182366178 1.036805809874421e-10
26 2.000006293048384 3.960245796558146e-11
27 2.000003889317794 1.512679290309838e-11
28 2.00000240373059 5.777920747782671e-12
29 2.000001485587204 2.206969341769987e-12
30 2.000000918143385 8.429872760069362e-13
31 2.000000567443819 3.219924872671856e-13
32 2.000000350699567 1.229901860977224e-13
33 2.000000216744252 4.697807072287988e-14
34 2.000000133955314 1.794402625504285e-14
35 2.000000082788937 6.854008169606801e-15
36 2.000000051166377 2.617998171864294e-15
37 2.00000003162256 9.999863089231212e-16
38 2.000000019543817 3.819607746096342e-16
39 2.000000012078743 1.458960299180093e-16
40 2.000000007465074 5.572732821864115e-17
41 2.000000004613669 2.128594576142821e-17
42 2.000000002851404 8.130507314050401e-18
43 2.000000001762265 3.105577932316062e-18
```

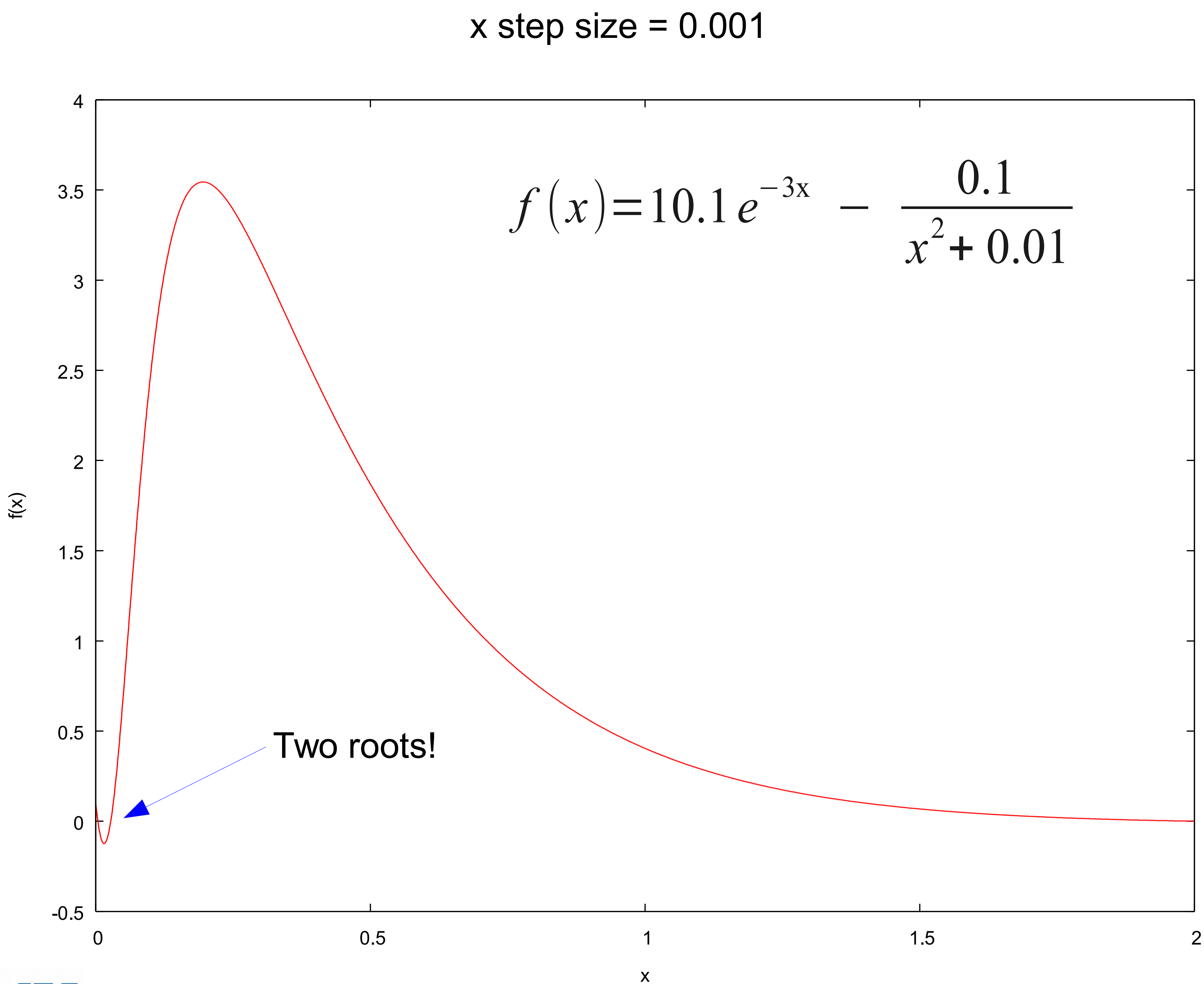
Pathological Function from Hopkins and Phillips

x step size = 0.05

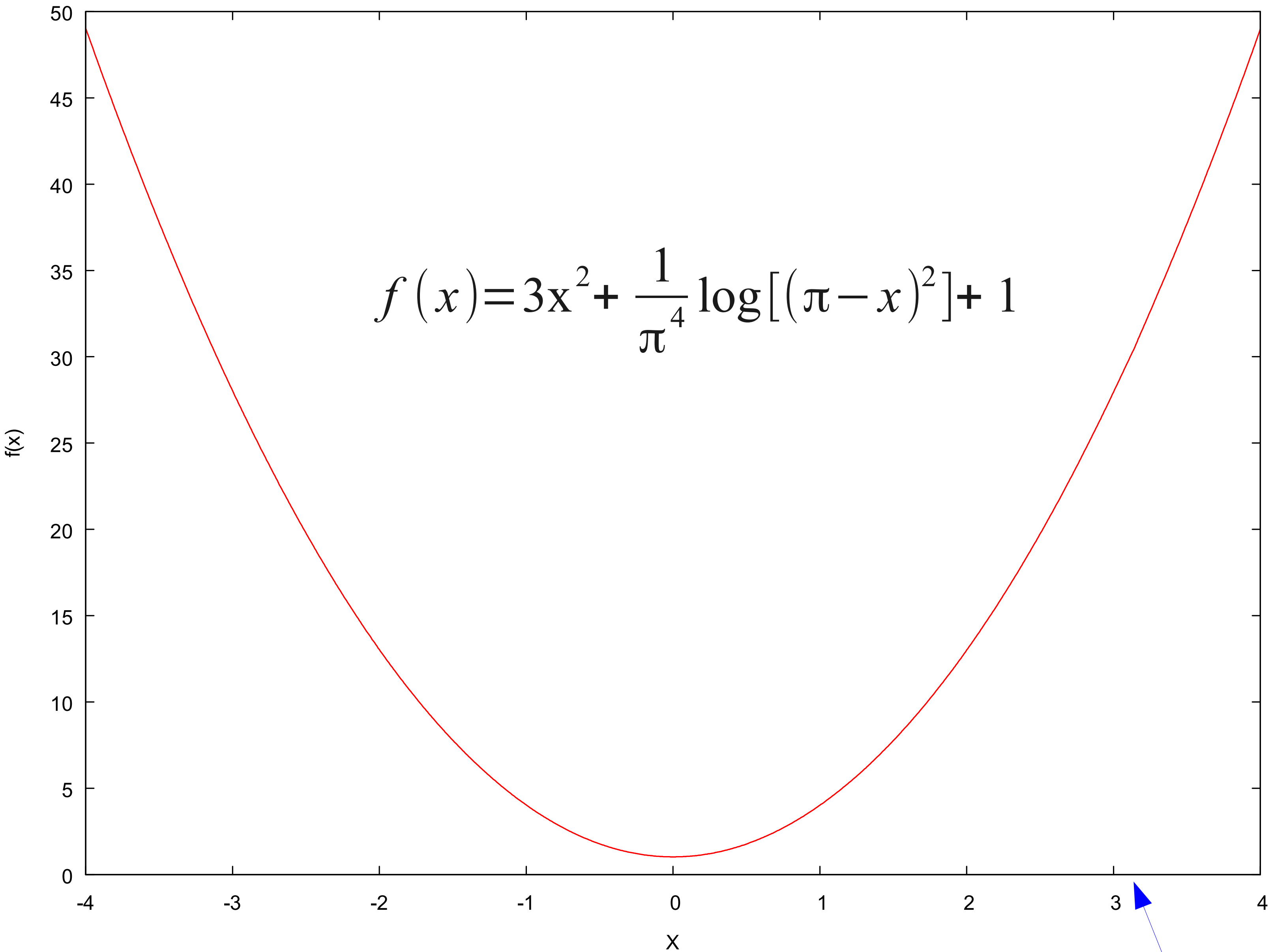
$$f(x) = 10.1\, e^{-3x} - \frac{0.1}{x^2 + 0.01}$$



Pathological Function from Hopkins and Phillips



Pathological Function Example from Numerical Recipes



$$f(x) = 3x^2 + \frac{1}{\pi^4}\log\left[(\pi - x)^2\right] + 1$$

Actually has two roots at $\sim\pi \pm 10^{-648}$

Pathological Function Code

Note use of 'long double' data type

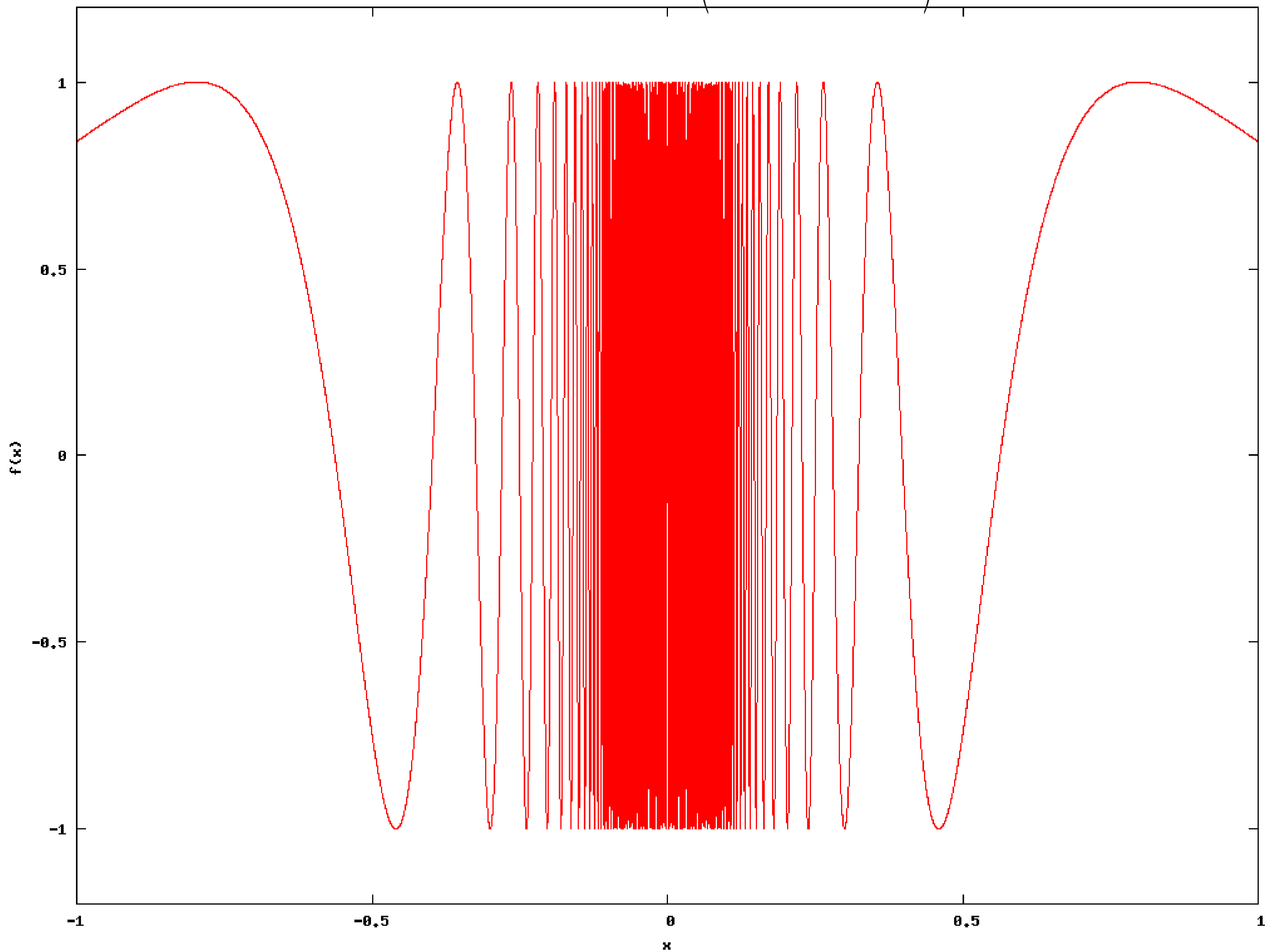
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159265358979323846264338327950288L

int main() {
  long double d,x,f;
  d = -10.0e-647L;
  while (d < 10.005e-647L) {
    x = PI + d;
    f = 3.0L*x*x + logl(d*d)/(PI*PI*PI*PI) + 1.0L;
    printf("d = %.6Le   f = %.6Le\n",d,f);
    d = d + 0.01e-647L;
  }
  exit(0);
}
```

Pathological Function Code Output

```
d = -1.000000e-646   f = 6.813257e-02
d = -9.990000e-647   f = 6.811203e-02
d = -9.980000e-647   f = 6.809146e-02
  ...
d = -4.000000e-648   f = 2.042725e-03
d = -3.900000e-648   f = 1.522900e-03
d = -3.800000e-648   f = 9.895727e-04
d = -3.700000e-648   f = 4.420212e-04
d = -3.600000e-648   f = -1.205336e-04
d = -3.500000e-648   f = -6.989371e-04
d = -3.400000e-648   f = -1.294108e-03
d = -3.300000e-648   f = -1.907048e-03
  ...
d = 3.200000e-648   f = -2.538851e-03
d = 3.300000e-648   f = -1.907048e-03
d = 3.400000e-648   f = -1.294108e-03
d = 3.500000e-648   f = -6.989371e-04
d = 3.600000e-648   f = -1.205336e-04
d = 3.700000e-648   f = 4.420212e-04
d = 3.800000e-648   f = 9.895727e-04
d = 3.900000e-648   f = 1.522900e-03
d = 4.000000e-648   f = 2.042725e-03
 ...
```

Pathological Function with a Huge Number of Roots

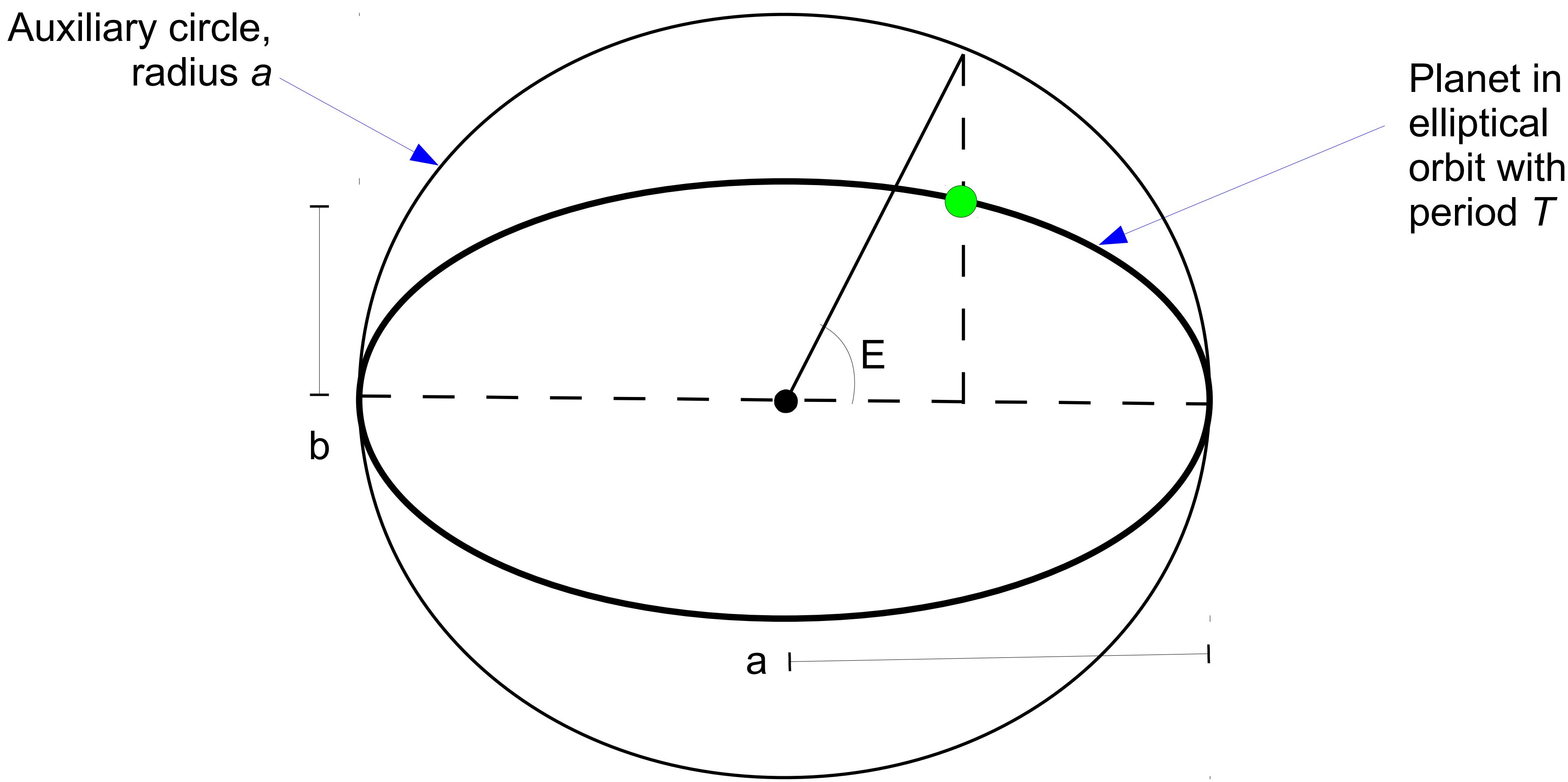
$$f(x) = \sin\left(\frac{1}{x^2 + 0.001}\right)$$



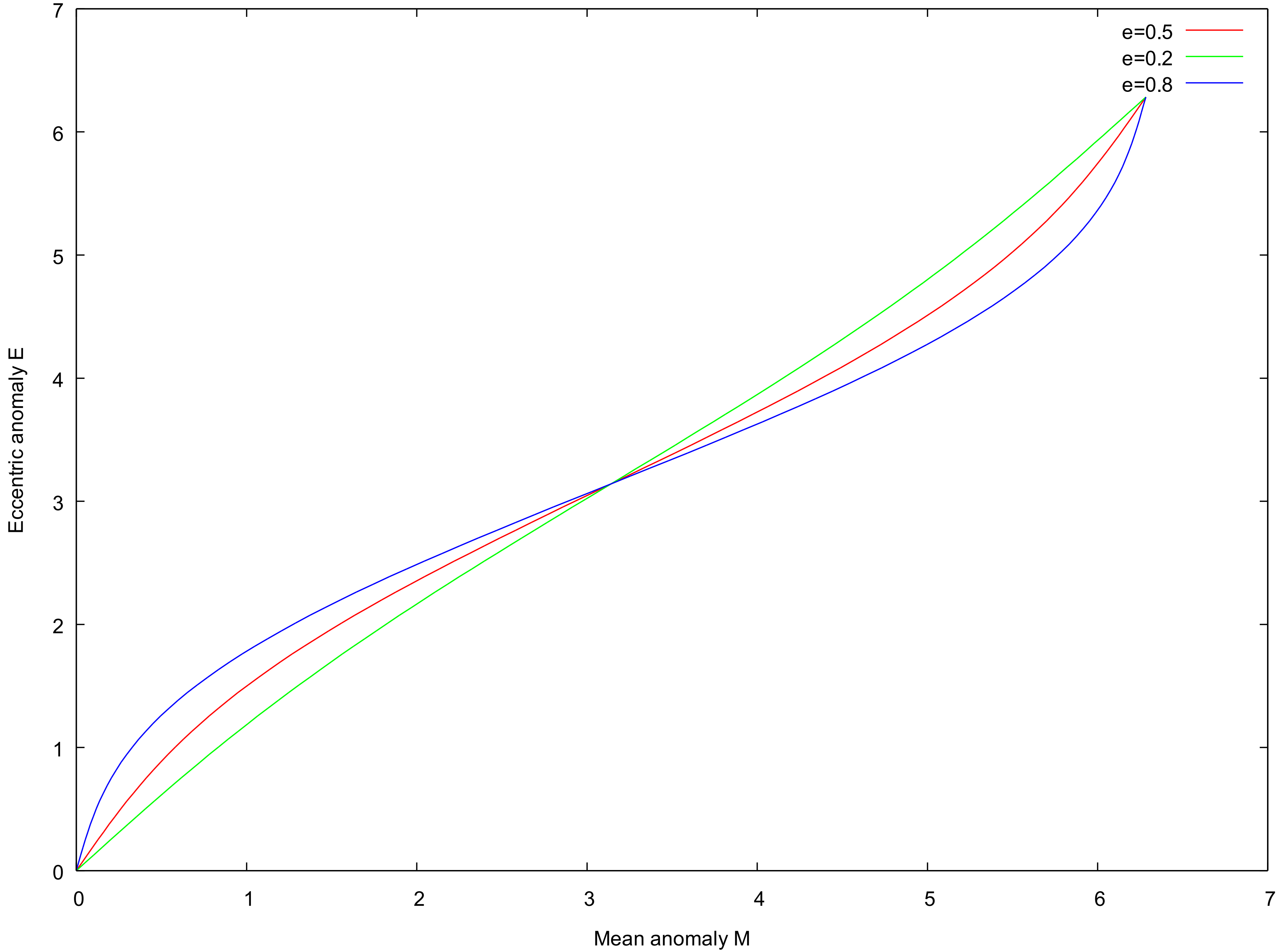
Kepler's Equation

$$M = E - \epsilon \cdot \sin(E)$$

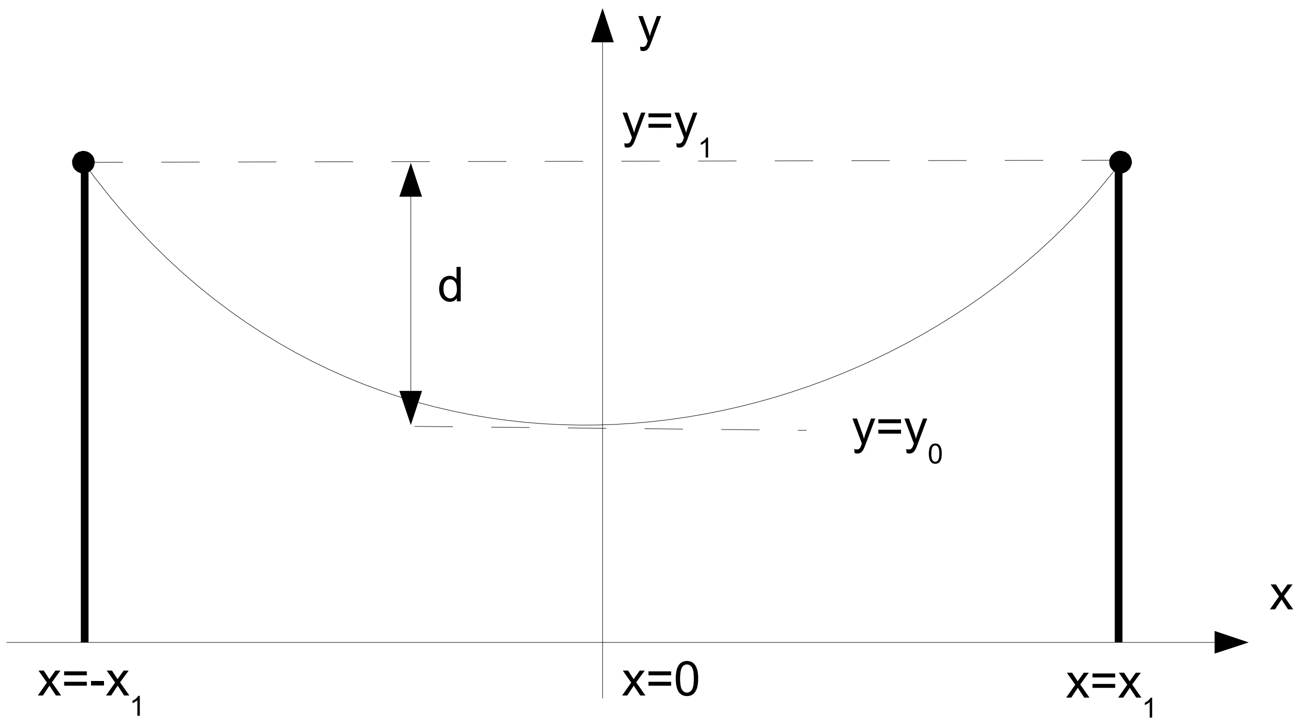
$$M = \text{Mean anomaly} = 2\pi \frac{t}{T} \quad E = \text{Eccentric anomaly} \quad \epsilon = \text{Eccentricity} = \sqrt{\left(1 - \frac{b^2}{a^2}\right)}$$



Kepler's Equation Solutions



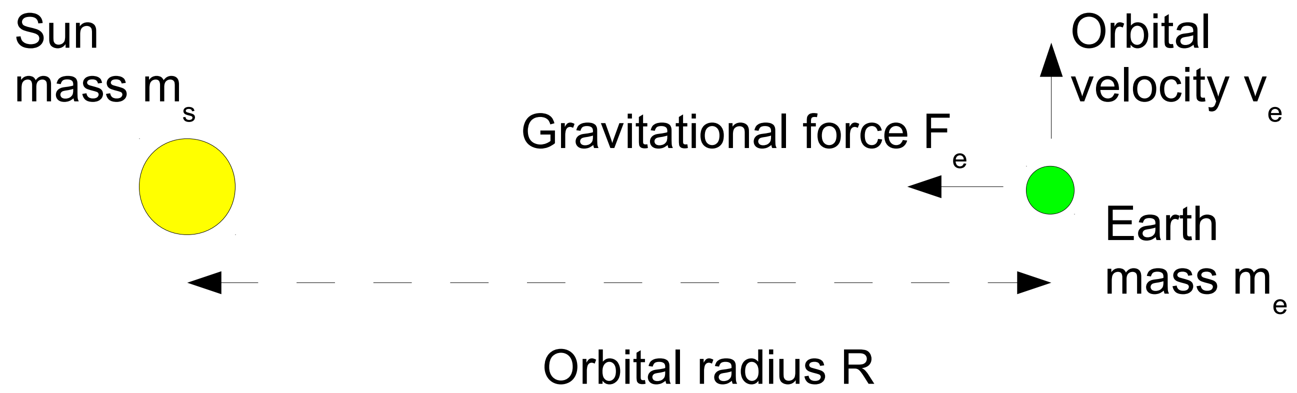
Catenary Profile of a Suspended Line



$$\text{elevation} \quad y(x) = \lambda \cosh\left(\frac{x}{\lambda}\right) - \lambda + y_0$$

$$\text{length} \quad L = 2\lambda \sinh\left(\frac{x_1}{\lambda}\right)$$

Sun-Earth Orbital System

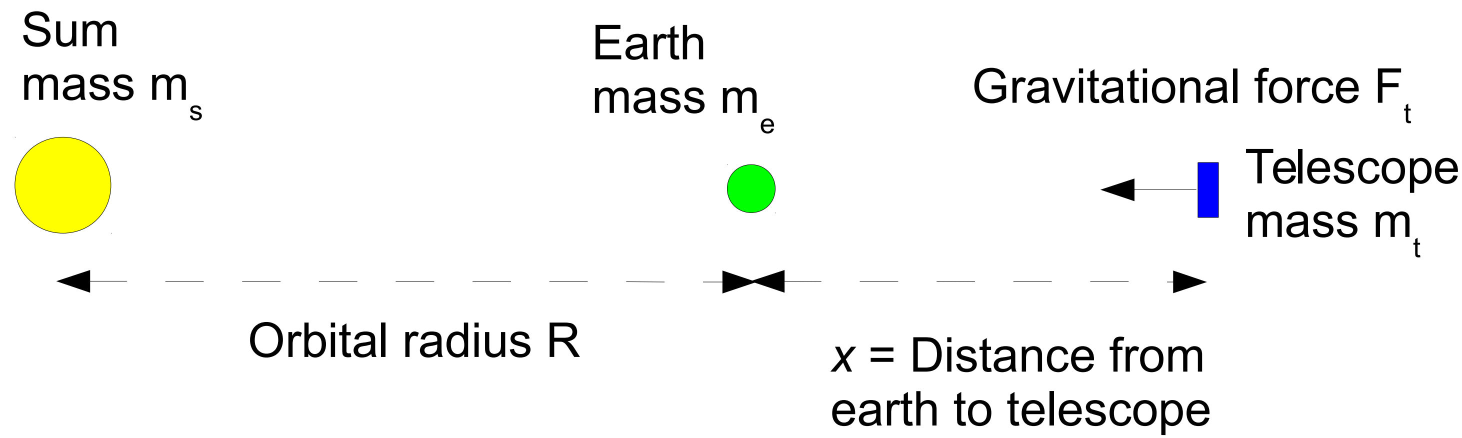


$$F_e = G\frac{m_s m_e}{R^2}$$

$$F_e = \frac{m_e v_e^2}{R} = \frac{4\pi^2 R m_e}{T_e^2}$$

$$T_e^2 = \frac{4\pi^2 R^3}{G m_s}$$

Sun-Earth Orbital System with JWST

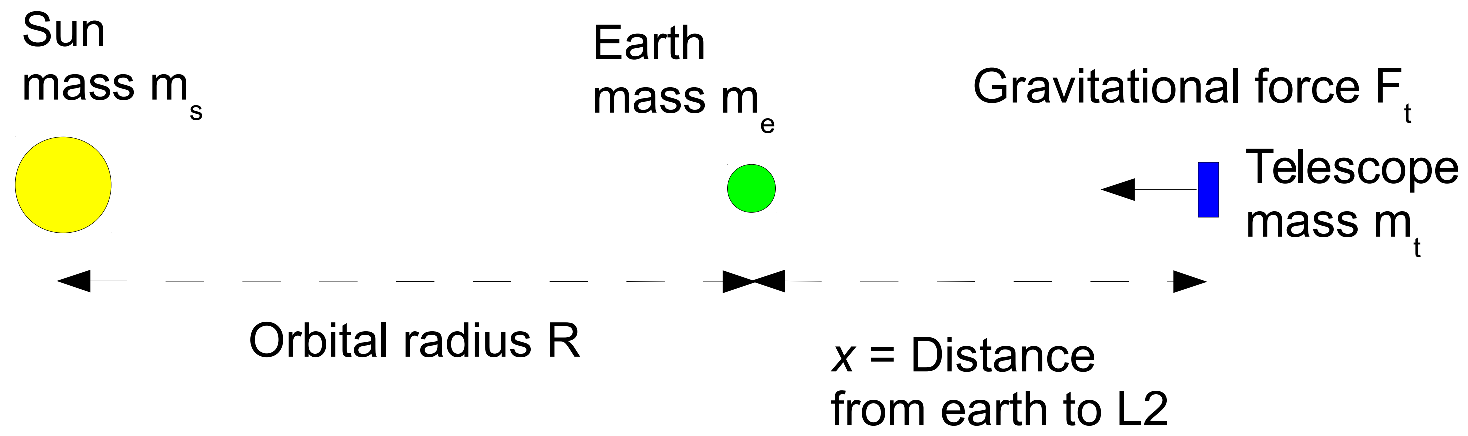


$$F_t = G\frac{m_s m_t}{(R+x)^2} + G\frac{m_e m_t}{x^2}$$

$$F_t = \frac{4\pi^2 (R+x) m_t}{T_t^2}$$

$$T_t^2 = \frac{4\pi^2 (R+x)}{\frac{G m_s}{(R+x)^2} + \frac{G m_e}{x^2}}$$

Sun-Earth Orbital System with JWST at L2



$$\text{At L2 point } T_e^2 = T_t^2$$

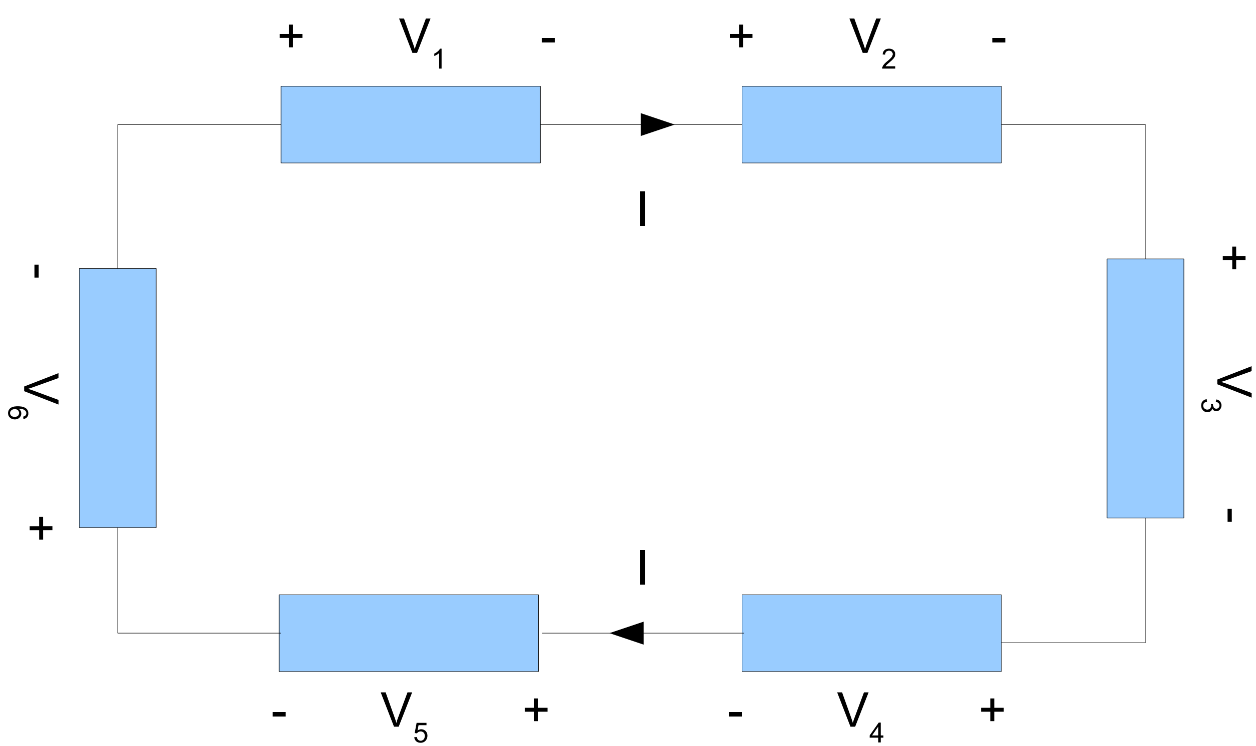
$$\frac{4\pi^2 R^3}{G m_s} = \frac{4\pi^2 (R+x)}{\dfrac{G m_s}{(R+x)^2} + \dfrac{G m_e}{x^2}}$$

Kirchoff's Voltage Law

Electrical circuit elements in a loop carrying current I

If all the reference polarities of all potential drops across loop elements are in the same loop direction, then the sum of all potential drops around the loop = 0

With the relationship between I and V_j for each circuit element, the current I can be found by solving for the root of an equation



$$\sum V_j = 0$$

Example of Kirchoff's Voltage Law

$$V_{PS} + V_R + V_{LED} = 0$$

$$V_R = I \cdot R$$

Note: Polarity of element voltage drop is determined by reference direction around loop

$$V_{PS} = -5\text{V}$$

$$V_{LED} = \frac{kT}{q} \cdot \log\left(\frac{I}{I_s}\right)$$