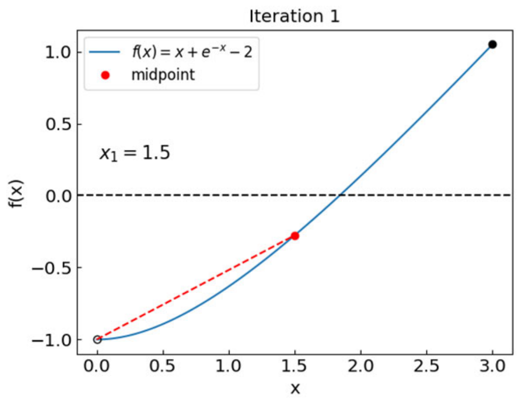
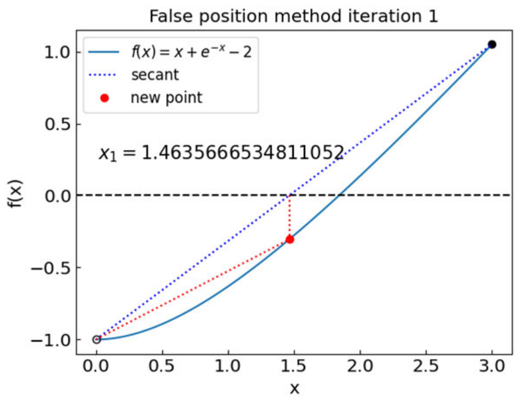
THE UNIVERSITY of TENNESSEE KNOXVILLE

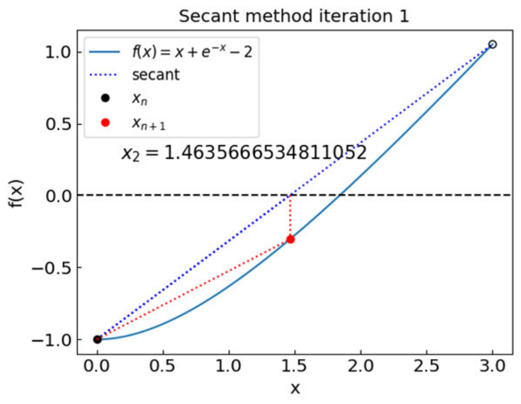
Lecture 07: Root Finding and Optimization

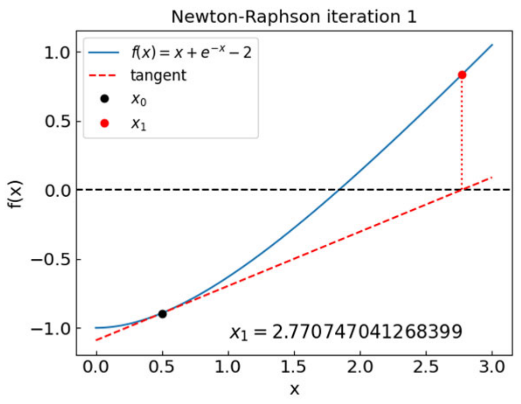
Sergei V. Kalinin

Summary


Iteration 1
$f(x) = x + e^{-x} - 2$
midpoint
$x_1 = 1.5$
f(x)
x


False position method iteration 1
$f(x) = x + e^{-x} - 2$
secant
new point
$x_1 = 1.4635666534811052$
f(x)
x


Secant method iteration 1
$f(x) = x + e^{-x} - 2$
secant
x_n
x_{n+1}
$x_2 = 1.4635666534811052$
f(x)
x


Newton-Raphson iteration 1
$f(x) = x + e^{-x} - 2$
tangent
x_0
x_1
$x_1 = 2.770747041268399$
f(x)
x

Summary

Bisection method:

- Guaranteed to converge with a fixed rate
- Need to bracket the root

False position method:

- Guaranteed to converge
- Can be faster than bisection but not always
- Need to bracket the root

Secant method:

- Typically faster than bisection and false position
- May not always converge

Newton-Raphson method:

- Very fast when converges
- Can be sensitive to initial guess
- May not converge if f'(x)=0
- Requires evaluation of the derivative at each step

Relaxation method:

- Simple to implement
- Does not require derivative
- Often does not converge

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Systems of non-linear equations

$$\begin{gathered}
f_1(x_1, \dots, x_N) = 0, \\
f_2(x_1, \dots, x_N) = 0, \\
\dots \\
f_N(x_1, \dots, x_N) = 0
\end{gathered}$$

References: Chapter 9.6 of *Numerical Recipes Third Edition* by W.H. Press et al.

Systems of non-linear equations

Sometimes we need to solve a system of non-linear equations, e.g.

$$\begin{gathered} f_1(x_1,\dots,x_N) = 0, \\ f_2(x_1,\dots,x_N) = 0, \\ \dots \\ f_N(x_1,\dots,x_N) = 0 \end{gathered}$$

Denoting $\mathbf{f} = (f_1,\dots,f_N)$ and $\mathbf{x} = (x_1,\dots,x_N)$ this can be written in compact form as

$$\mathbf{f}(\mathbf{x}) = 0\,.$$

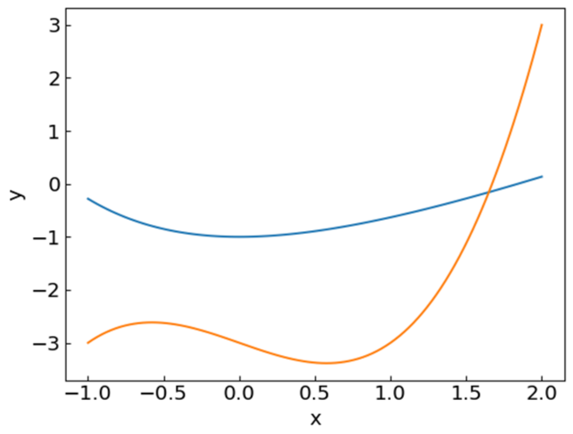
For example:

$$\begin{gathered} x + \exp(-x) - 2 - y = 0, \\ x^3 - x - 3 - y = 0. \end{gathered}$$

i.e.

$$f_1(x_1,x_2) = x_1 + \exp(-x_1) - 2 - x_2$$

$$f_2(x_1,x_2) = x_1^3 - x_1 - 3 - x_2$$



From the course by Volodymyr Vovchenko, https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Newton-Raphson method in multiple dimensions

Recall the Taylor expansion of function f around the root x^* in one-dimensional case:

$$f(x^*) \approx f(x) + f'(x)(x^* - x)$$

The multi-dimensional version of this expansion reads

$$\mathbf{f}(\mathbf{x}^*) \approx \mathbf{f}(\mathbf{x}) + \boldsymbol{J}(\mathbf{x})(\mathbf{x}^* - \mathbf{x})$$

Here $\boldsymbol{J}(\mathbf{x})$ is the Jacobian, i.e. a $N \times N$ matrix of derivatives evaluated at $\mathbf{x}$

$$\boldsymbol{J}_{ij}(\mathbf{x}) = \frac{\partial f_i}{\partial x_j} .$$

Given that $\mathbf{f}(\mathbf{x}^*) = 0$, we have

$$\boldsymbol{J}(\mathbf{x})(\mathbf{x}^* - \mathbf{x}) \approx -\mathbf{f}(\mathbf{x}),$$

which is a system of linear equations for $\mathbf{x}^* - \mathbf{x}$. Solving this system yields

$$\mathbf{x}^* \approx \mathbf{x} - \boldsymbol{J}^{-1}(\mathbf{x})\,\mathbf{f}(\mathbf{x}) .$$

Here $\boldsymbol{J}^{-1}(\mathbf{x})$ is the inverse Jacobian matrix.

The multi-dimensional Newton's method is an iterative procedure

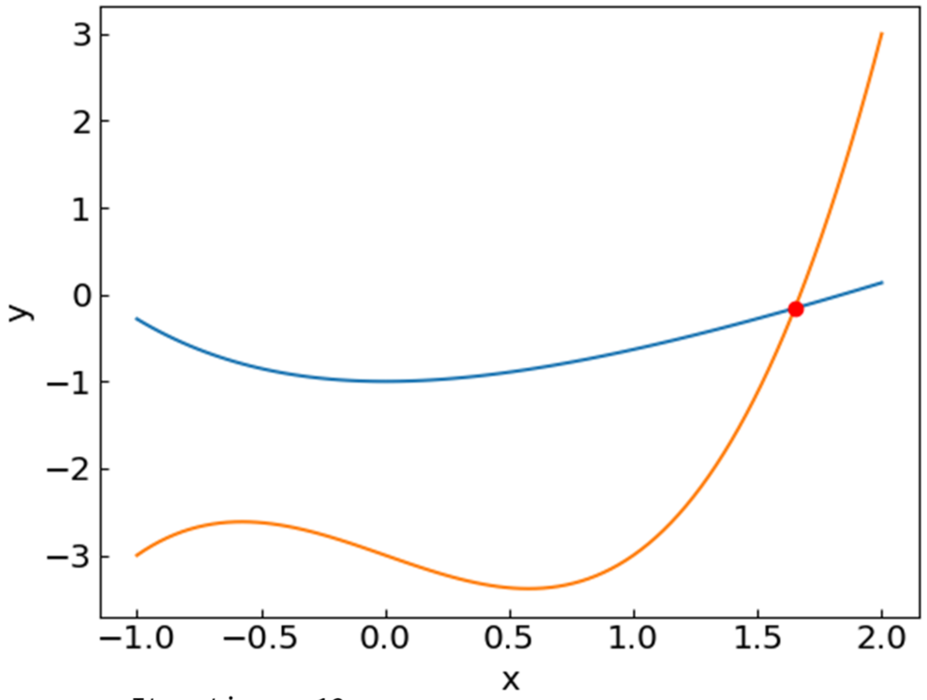
$$\mathbf{x_{n+1}} = \mathbf{x_n} - \boldsymbol{J}^{-1}(\mathbf{x_n})\,\mathbf{f}(\mathbf{x_n}) .$$

Reduces to Newton's method in 1D $\quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

From the course by Volodymyr Vovchenko, https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Newton-Raphson method in multiple dimensions

```python
def newton_method_multi(
    f,
    jacobian,
    x0,
    accuracy=1e-8,
    max_iterations=100):
    x = x0
    global last_newton_iterations
    last_newton_iterations = 0

    if newton_verbose:
        print("Iteration: ", last_newton_iterations)
        print("x = ", x0)
        print("f = ", f(x0))
        print("|f| = ", ftil(f(x0)))

    for i in range(max_iterations):
        last_newton_iterations += 1
        f_val = f(x)
        jac = jacobian(x)
        jinv = np.linalg.inv(jac)
        delta = np.dot(jinv, -f_val)
        x = x + delta

        if np.linalg.norm(delta, ord=2) < accuracy:
            return x
    return x
```



```
Iteration:  12
x =  [ 1.64998819 -0.15795963]
f =  [ 0.00000000e+00 -6.66133815e-16]
|f| =  2.2186712959340957e-31
```

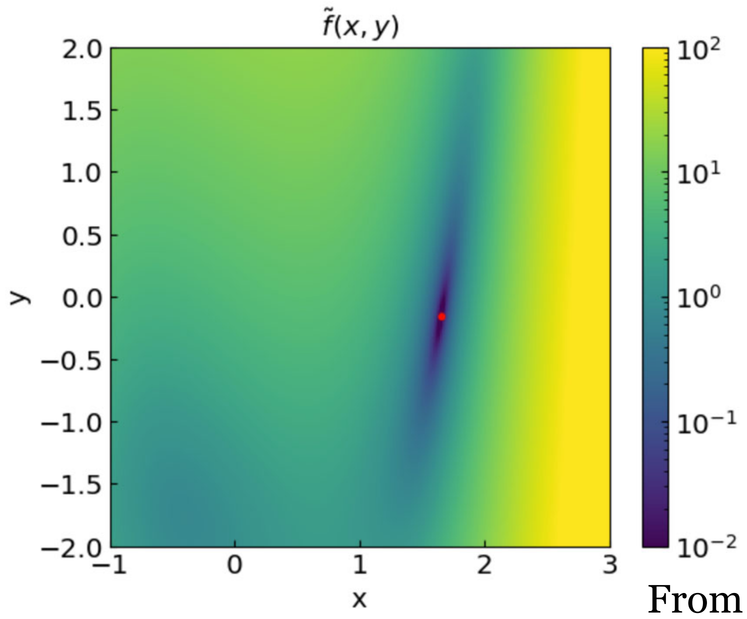
From the course by Volodymyr Vovchenko, https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Newton-Raphson method in multiple dimensions

Introduce an objective function

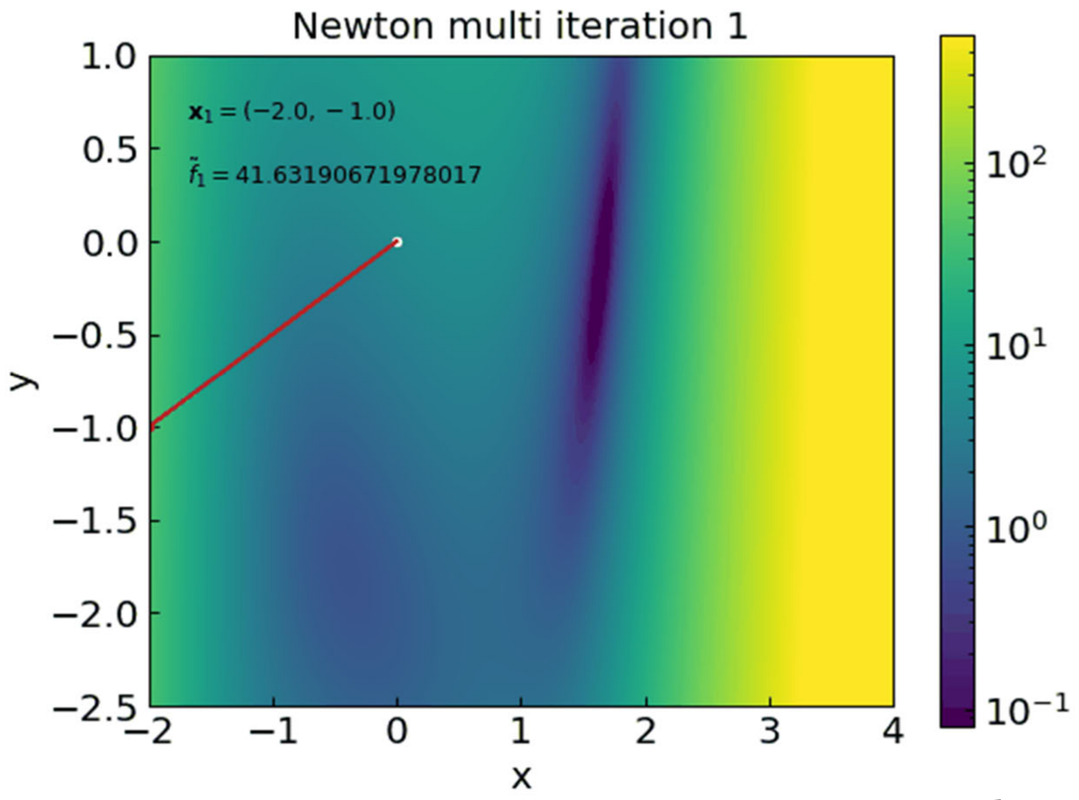
$$\tilde{f}(\mathbf{x}) = \frac{\mathbf{f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x})}{2}$$

Its value is equal to zero (minimized) at the root



From the course by Volodymyr Vovchenko, https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Newton-Raphson method in multiple dimensions



From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Broyden method

Broyden method is a multi-dimensional generalization of the secant method

Secant method: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ with $f'(x_n) \simeq \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$

Broyden method: $\mathbf{x_{n+1}} = \mathbf{x_n} - J^{-1}(\mathbf{x_n})\,\mathbf{f}(\mathbf{x_n})$ with

The solution to $J(\mathbf{x_n})\,(\mathbf{x_n} - \mathbf{x_{n-1}}) \simeq \mathbf{f}(\mathbf{x_n}) - \mathbf{f}(\mathbf{x_{n-1}})$ is not unique

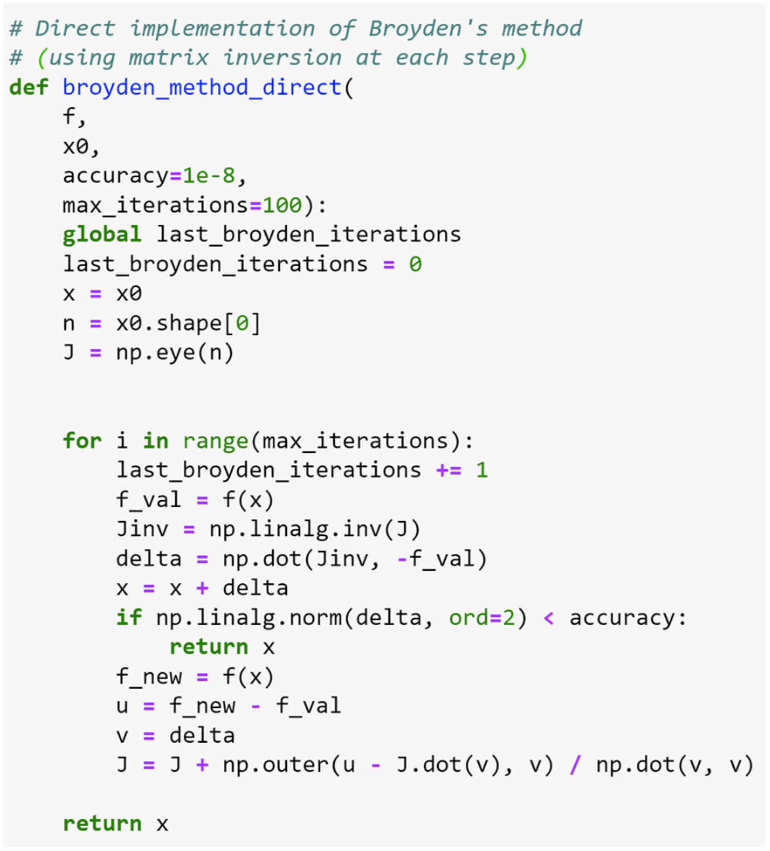
Broyden: $\mathbf{J}_n = \mathbf{J}_{n-1} + \frac{\Delta\mathbf{f}_n - \mathbf{J}_{n-1}\Delta\mathbf{x}_n}{\|\Delta\mathbf{x}_n\|^2}\Delta\mathbf{x}_n^{\mathrm{T}}$ with

$$\mathbf{f}_n = \mathbf{f}(\mathbf{x}_n),$$
$$\Delta\mathbf{x}_n = \mathbf{x}_n - \mathbf{x}_{n-1},$$
$$\Delta\mathbf{f}_n = \mathbf{f}_n - \mathbf{f}_{n-1},$$

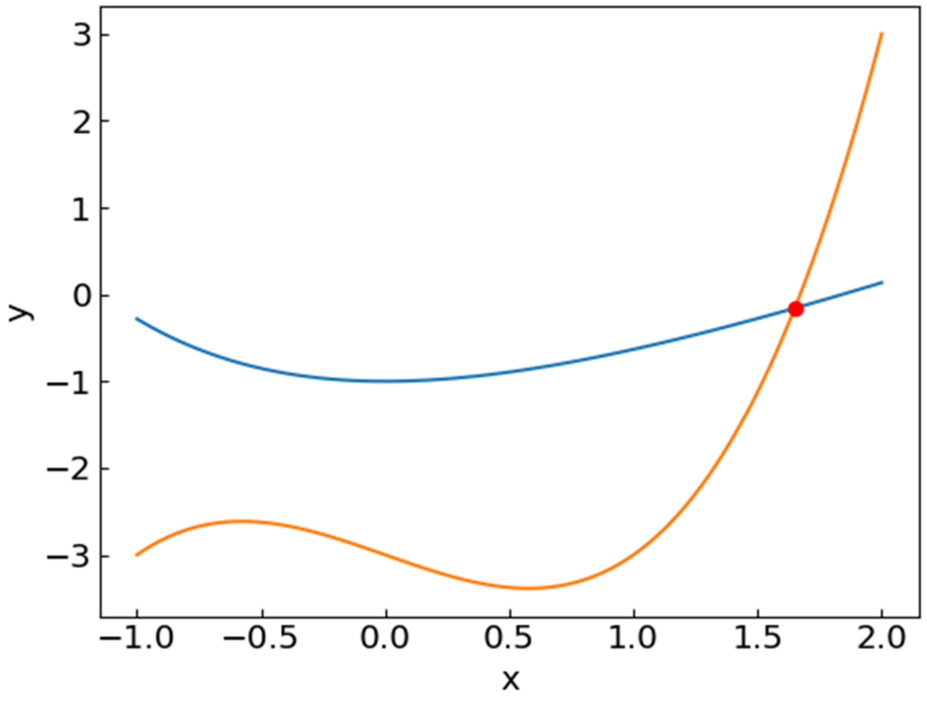
For J_0 one can take the identity matrix or full Jacobian calculated once

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Broyden method (direct)



```python
# Direct implementation of Broyden's method
# (using matrix inversion at each step)
def broyden_method_direct(
    f,
    x0,
    accuracy=1e-8,
    max_iterations=100):
    global last_broyden_iterations
    last_broyden_iterations = 0
    x = x0
    n = x0.shape[0]
    J = np.eye(n)


    for i in range(max_iterations):
        last_broyden_iterations += 1
        f_val = f(x)
        Jinv = np.linalg.inv(J)
        delta = np.dot(Jinv, -f_val)
        x = x + delta
        if np.linalg.norm(delta, ord=2) < accuracy:
            return x
        f_new = f(x)
        u = f_new - f_val
        v = delta
        J = J + np.outer(u - J.dot(v), v) / np.dot(v, v)

    return x
```



```
Iteration:  54
x =  [ 1.64998819 -0.15795963]
f =  [ 2.97817326e-14 -4.50097265e-10]
|f| =  1.0129377443026415e-19
```

From the course by Volodymyr Vovchenko, https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Broyden method: avoid matrix inversion

$$\mathbf{J}_n = \mathbf{J}_{n-1} + \frac{\Delta \mathbf{f}_n - \mathbf{J}_{n-1}\Delta \mathbf{x}_n}{\|\Delta \mathbf{x}_n\|^2}\Delta \mathbf{x}_n^{\mathrm{T}}$$

Sherman-Morrison formula:

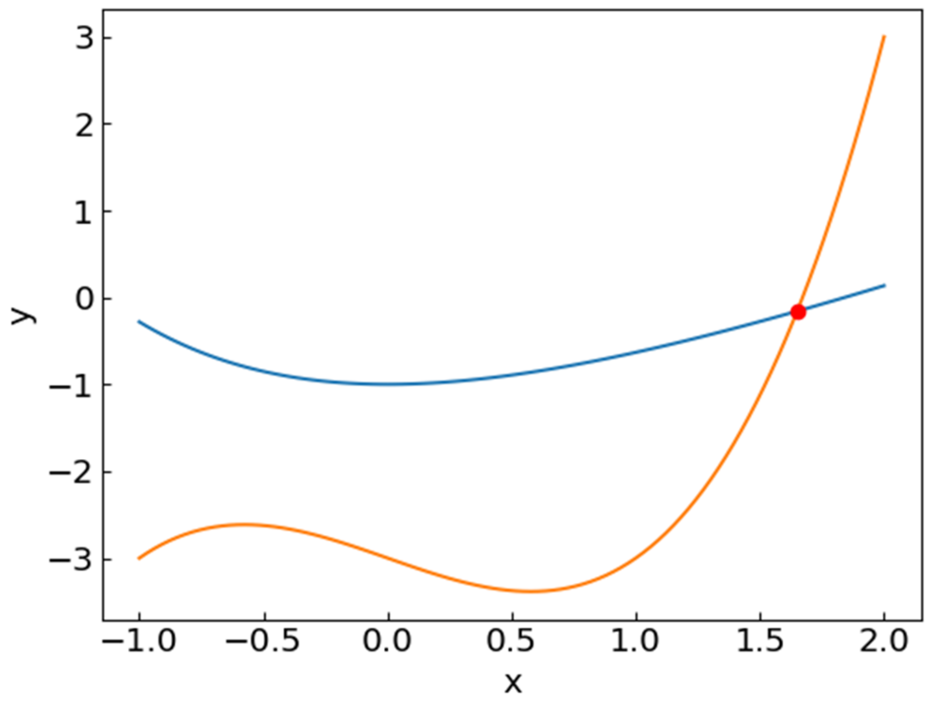
$$\mathbf{J}_n^{-1} = \mathbf{J}_{n-1}^{-1} + \frac{\Delta \mathbf{x}_n - \mathbf{J}_{n-1}^{-1}\Delta \mathbf{f}_n}{\Delta \mathbf{x}_n^{\mathrm{T}}\mathbf{J}_{n-1}^{-1}\Delta \mathbf{f}_n}\Delta \mathbf{x}_n^{\mathrm{T}}\mathbf{J}_{n-1}^{-1}$$

Update the inverse Jacobian directly!

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Broyden method (Sherman-Morrison)

```python
def broyden_method(
    f,
    x0,
    accuracy=1e-8,
    max_iterations=100):
    global last_broyden_iterations
    last_broyden_iterations = 0
    x = x0
    n = x0.shape[0]
    Jinv = np.eye(n)

    for i in range(max_iterations):
        last_broyden_iterations += 1
        f_val = f(x)
        delta = -Jinv.dot(f_val)
        x = x + delta
        if np.linalg.norm(delta, ord=2) < accuracy:
            return x
        f_new = f(x)
        df = f_new - f_val
        dx = delta
        Jinv = Jinv + np.outer(dx - Jinv.dot(df), dx.T.dot(Jinv))
        / np.dot(dx.T, Jinv.dot(df))

    return x
```



```
Iteration:  54
x =  [ 1.64998819 -0.15795963]
f =  [ 2.8255176e-14 -3.8877096e-10]
|f| =  7.557143001803891e-20
```

From the course by Volodymyr Vovchenko, https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Broyden method vs Newton-Raphson method

Broyden method converges somewhat slower (e.g. 54 vs 12 iterations in our example) but:

- Does not involve the calculation of Jacobian
- Does not involve matrix inversion

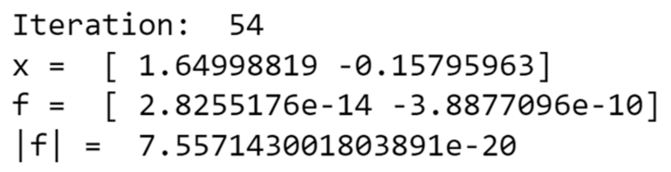
From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Broyden method vs Newton-Raphson method

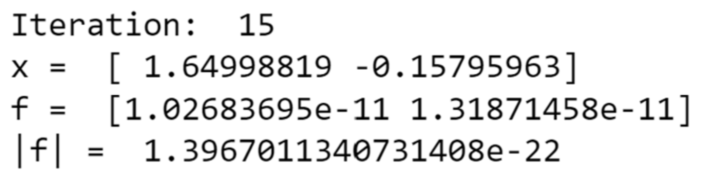
Broyden method converges somewhat slower (e.g. 54 vs 12 iterations in our example) but:

- Does not involve the calculation of Jacobian
- Does not involve matrix inversion

Possible refinement: improve the initial estimate for the Jacobian



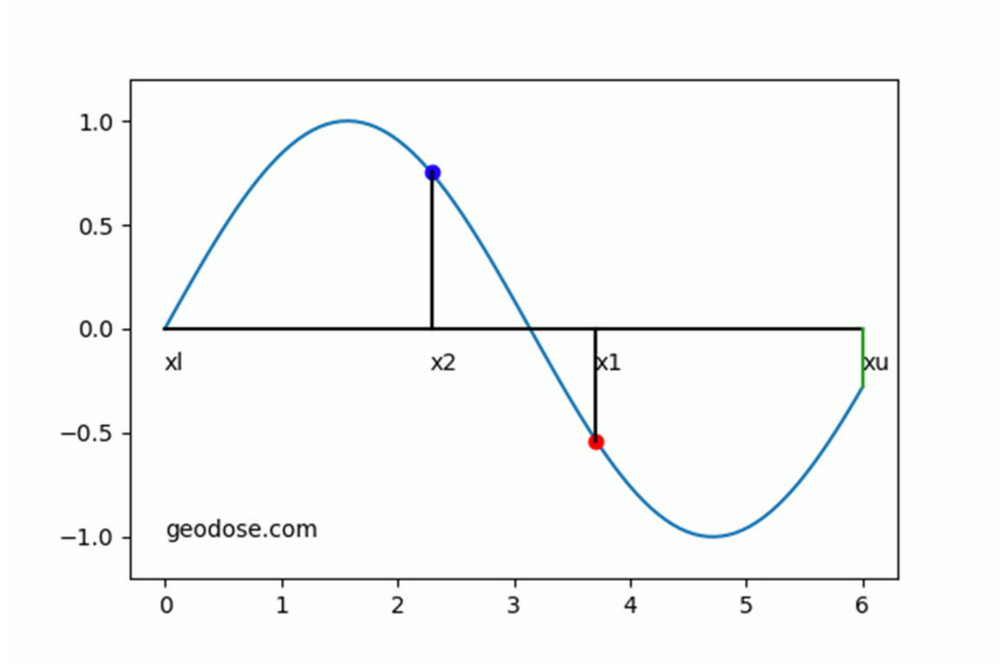
```
Iteration:  54
x =  [ 1.64998819 -0.15795963]
f =  [ 2.8255176e-14 -3.8877096e-10]
|f| =  7.557143001803891e-20
```



```
Iteration:  15
x =  [ 1.64998819 -0.15795963]
f =  [1.02683695e-11 1.31871458e-11]
|f| =  1.3967011340731408e-22
```

From the course by Volodymyr Vovchenko, https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Function minimization/maximization



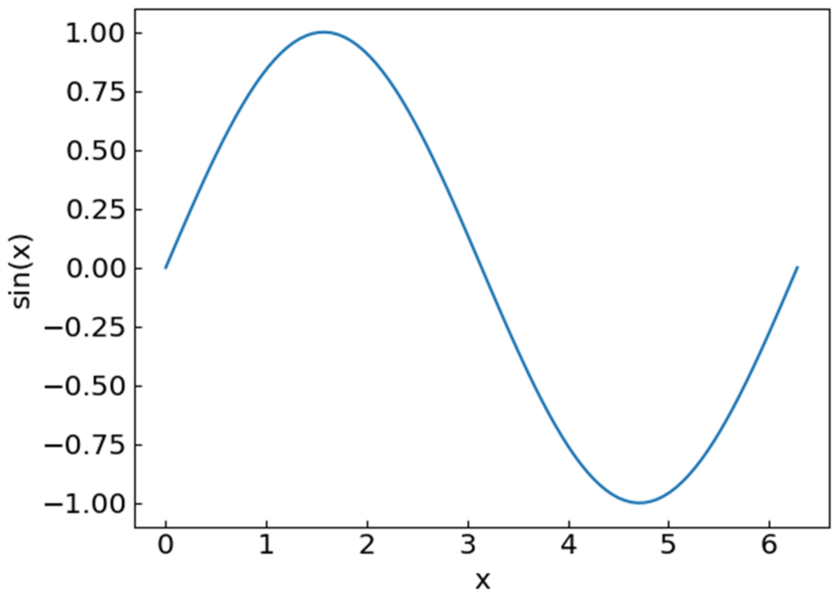
References: Chapter 6.4 of *Computational Physics* by Mark Newman
Chapter 10 of *Numerical Recipes Third Edition* by W.H. Press et al.

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Function extrema

Often we are interested to find the minimum of a function (e.g. energy minimization)

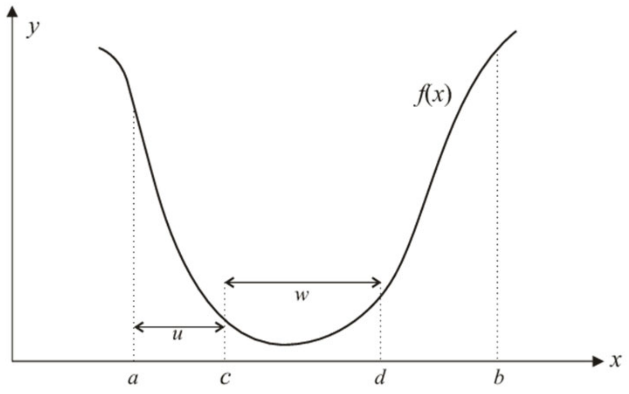
Consider the minimum of f(x)=sin(x) on interval 0..2π



From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Golden section search

- Bracket the minimum x_{min} in (a,b)
- Take c = b – (b-a)/φ and d = a + (b-a)/φ
- if f(c)<f(d), take b = d as new right endpoint
- Otherwise, take a = c as new left endpoint
- Repeat over the new interval (a,b) until the desired accuracy is reached



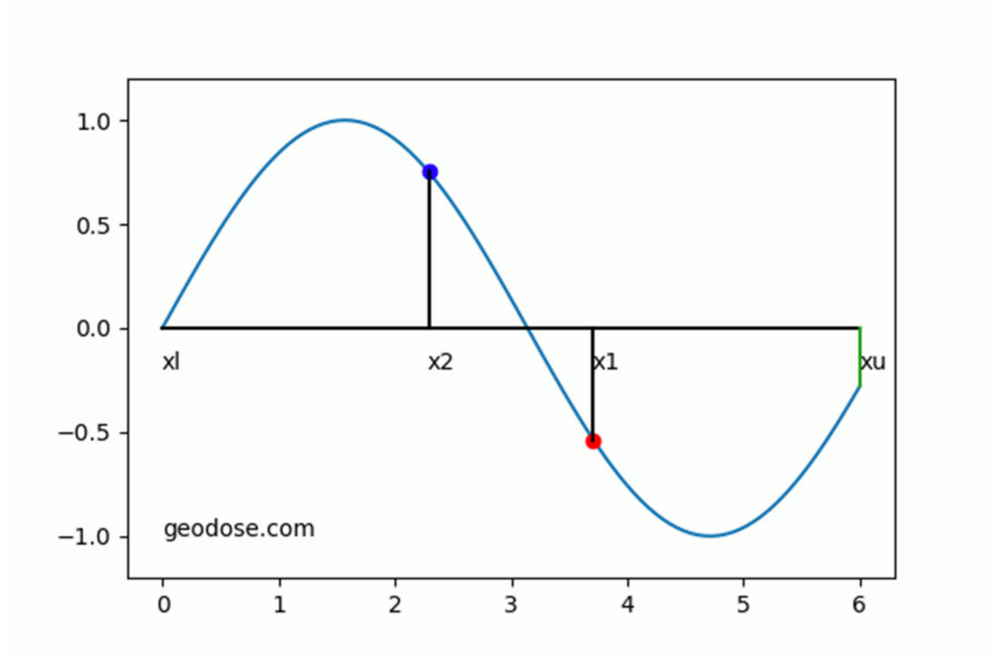
$$\varphi = \frac{1+\sqrt{5}}{2} = 1.618\ldots$$ is the **golden ratio**

This value ensures that the interval decreases by factor φ no matter what

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Golden section search

```python
def gss(f, a, b, accuracy=1e-7):
    c = b - (b - a) / phi
    d = a + (b - a) / phi
    while abs(b - a) > accuracy:
        if f(c) < f(d):
            b = d
        else:
            a = c

        c = b - (b - a) / phi
        d = a + (b - a) / phi

    return (b + a) / 2
```



```
The minimum of sin(x) over the interval ( 0.0 , 6.283185307179586 ) is 4.712388990891052
```

To search for maximum of f(x) look for minimum of –f(x)

From the course by Volodymyr Vovchenko,
https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Newton method

The extremum of f(x) is the root of the derivative, $f'(x) = 0$

Simply apply Newton-Raphson method for finding the root of $f'(x)$

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

$f''(x) > 0, \quad \rightarrow \quad$ minimum

$f''(x) < 0, \quad \rightarrow \quad$ maximum

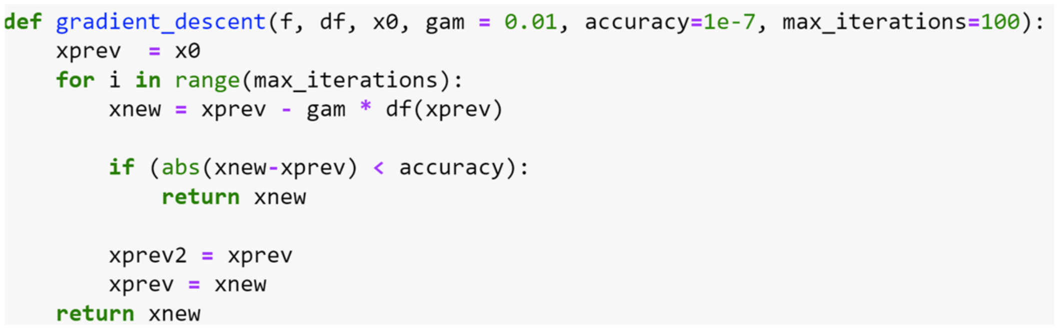
```python
def newton_extremum(f, df, d2f, x0, accuracy=1e-7, max_iterations=100):
    xprev = xnew = x0
    for i in range(max_iterations):
        xnew = xprev - df(xprev) / d2f(xprev)

        if (abs(xnew-xprev) < accuracy):
            return xnew

        xprev = xnew
    return xnew
```

```
An extremum of sin(x) using Newton's method starting from x0 =  5.0  is ( 0.0 , 6.283185307179586 ) is 4.71238898038469
```

Gradient descent method

Replace, $f''(x)$ by a descent factor $1/\gamma_n$

$$x_{n+1} = x_n - \gamma_n f'(x_n)$$



```python
def gradient_descent(f, df, x0, gam = 0.01, accuracy=1e-7, max_iterations=100):
    xprev  = x0
    for i in range(max_iterations):
        xnew = xprev - gam * df(xprev)

        if (abs(xnew-xprev) < accuracy):
            return xnew

        xprev2 = xprev
        xprev = xnew
    return xnew
```

From the course by Volodymyr Vovchenko, https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Freedom in choosing γ_n

Can be generalized to multi-variable function F(x_1,x_2,…)

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n)$$

What (and why) are Hackathons?

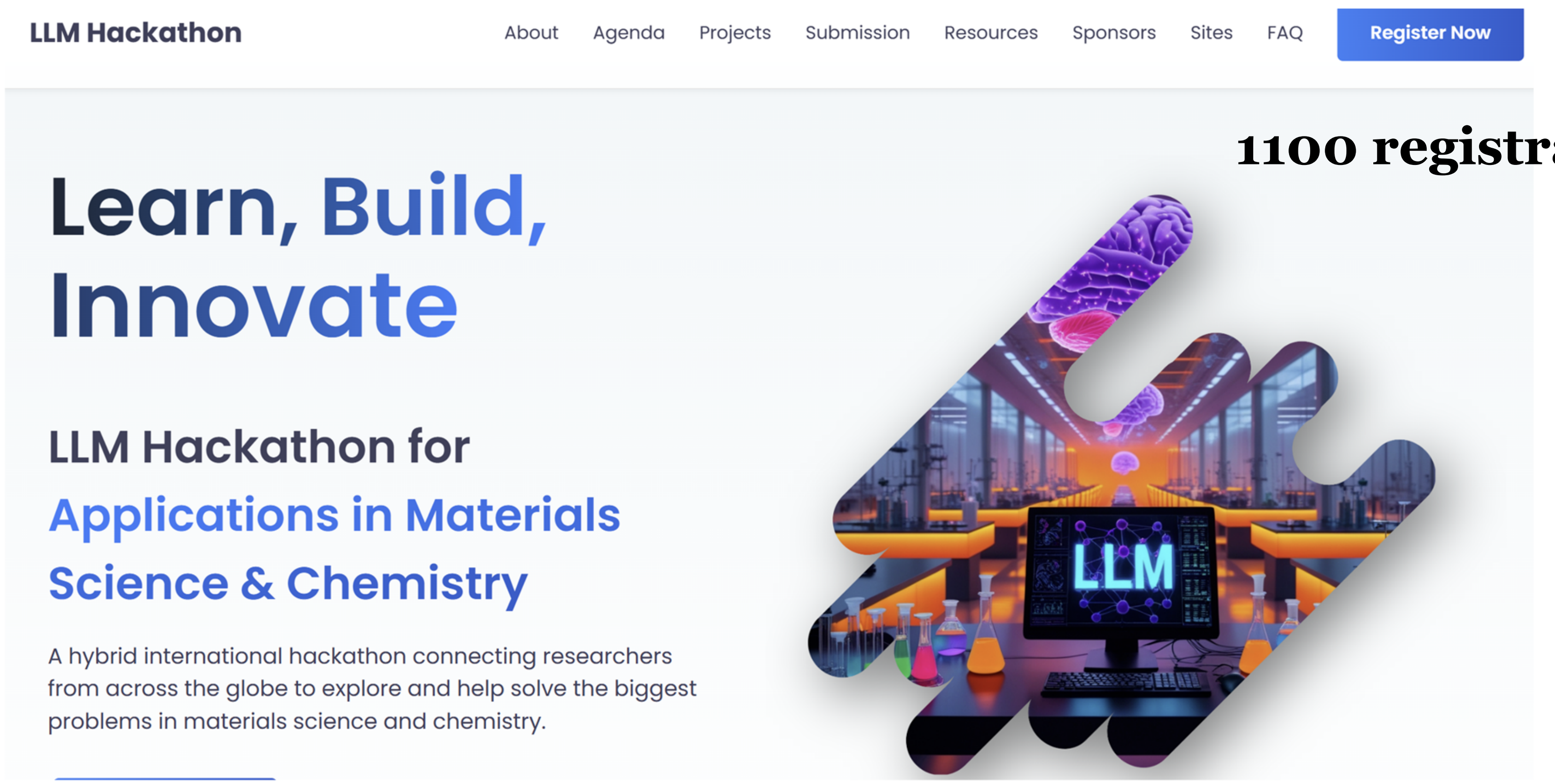
1. Competition

2. Maintaining software ecosystem (e.g. weekly PyCroscopy hackathon run by Rama Vasudeva and Gerd Duscher)

3. Focused problem solving

MARATHON
HACKATHON

LLM for Materials Hackathon



Learn, Build, Innovate

LLM Hackathon for Applications in Materials Science & Chemistry

A hybrid international hackathon connecting researchers from across the globe to explore and help solve the biggest problems in materials science and chemistry.

1100 registrations

UTK PoC: Sheryl Sanchez, ssanch18@vols.utk.edu

Require registration!

September 11-12, IAMM. Pizza will be provided!

ML for Microscopy Hackathon

Only UTK and remote
250 registrations,
80 participants

December 2024 Hackathon:

https://kaliningroup.github.io/mic-hackathon/projects/

~10 satellite sites

December 2025 Hackathon:

https://kaliningroup.github.io/mic_hackathon_2/

Focussed Problem Solving

Heat conductivity vs. concentration

1. **Preparation:**
   1. Create conductivity_dataset_colab.ipynb with:
   2. Upload/Load CSV (sample_id, T, data from combi).
   3. Alignment (if measurements were done on different grids)
   4. One cell for **train/val/test split** (+ saved splits to Drive).
   5. One **baseline** (linear reg) and a simple plot (σ vs T).
   6. Drop the CSV + notebook in a shared Drive folder.

2. **Hack-day (Colab)**
   1. Extend the notebook:
   2. Train 2–3 quick models (RandomForest/KRR/XGB).
   3. Add a **predict()** cell + tiny widget form for inputs.
   4. Save models
   5. **Success metrics and explainability**

Digital twin for film roughness

1. **Prep (Colab)**
    1. Create film_roughness_twin_colab.ipynb with:
    2. Load a roughness map from experiment.
2. **Hack-day (Colab)**
    1. Build the simulator (with ChatGPT).
    2. Build predictive map
    3. Calibrate re data and get model parameters
    4. Potentially build the exploration or co-navigation.

X-ray fitting with LMFit (XRD/XRR)

1. **Prep (Colab)**
    1. Create xray_fit_lmfit_colab.ipynb with:
    2. Mount Drive; read 3 example scans (CSV: angle/Q, intensity).
    3. Background subtract + **LMFit** demo (Voigt peaks for XRD or Parratt for XRR).
    4. Save a **template function** fit_scan(filepath)
2. **Hack-day (Colab)**
    1. Batch loop over a Drive folder; write results.csv (fit params, reduced χ^2, flags).
    2. Add a simple plot for each fit (saved PNGs).
    3. Analyze over combi library and plot results. Offer interpretations

Value Proposition

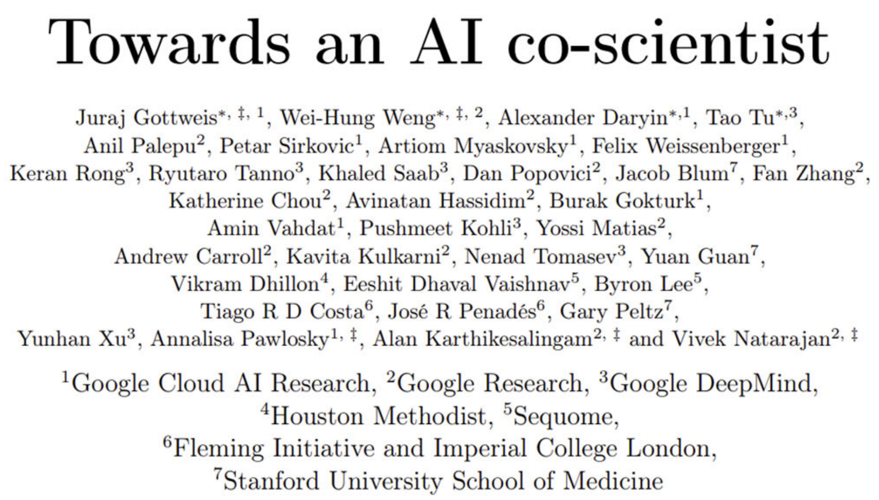
1. For trainees:
   - Common programming language
   - Solve actual problems
   - Upskilling and learning what to learn
   - Prepare for competition hackathons
   - Build up GitHub repo

2. For PIs:
   - Connecting domain science to ML and back
   - Learning what is out there
   - Learning to prototype

3. For the center:
   - Education model/outreach: involve more people
   - Getting stuff done
   - Visibility in the ML world

https://github.com/KalininGroup/camm_hackathon/blob/k4my4r/docs/day_1_09262025/CAMM_Hackathon_1.ipynb

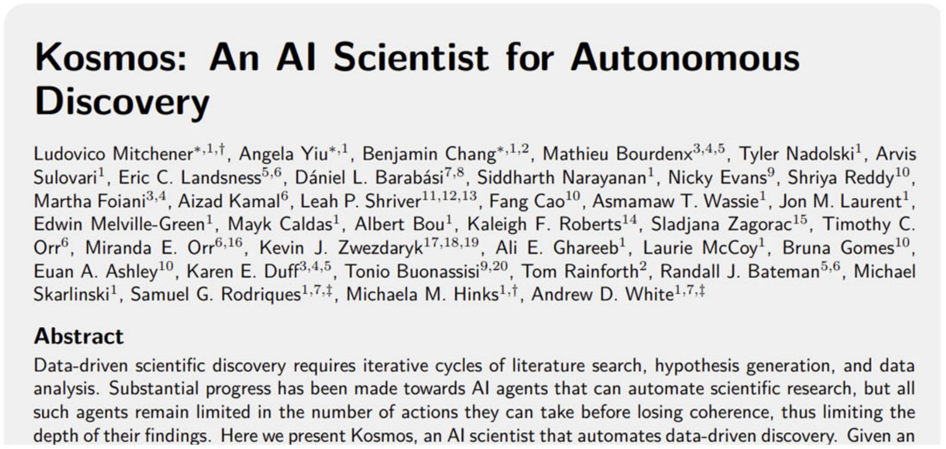
AI Scientists



Towards an AI co-scientist

Juraj Gottweis[*,‡,1], Wei-Hung Weng[*,‡,2], Alexander Daryin[*,1], Tao Tu[*,3], Anil Palepu[2], Petar Sirkovic[1], Artiom Myaskovsky[1], Felix Weissenberger[1], Keran Rong[3], Ryutaro Tanno[3], Khaled Saab[3], Dan Popovici[2], Jacob Blum[7], Fan Zhang[2], Katherine Chou[2], Avinatan Hassidim[2], Burak Gokturk[1], Amin Vahdat[1], Pushmeet Kohli[3], Yossi Matias[2], Andrew Carroll[2], Kavita Kulkarni[2], Nenad Tomasev[3], Yuan Guan[7], Vikram Dhillon[4], Eeshit Dhaval Vaishnav[5], Byron Lee[5], Tiago R D Costa[6], José R Penadés[6], Gary Peltz[7], Yunhan Xu[3], Annalisa Pawlosky[1,‡], Alan Karthikesalingam[2,‡] and Vivek Natarajan[2,‡]

[1]Google Cloud AI Research, [2]Google Research, [3]Google DeepMind, [4]Houston Methodist, [5]Sequome, [6]Fleming Initiative and Imperial College London, [7]Stanford University School of Medicine



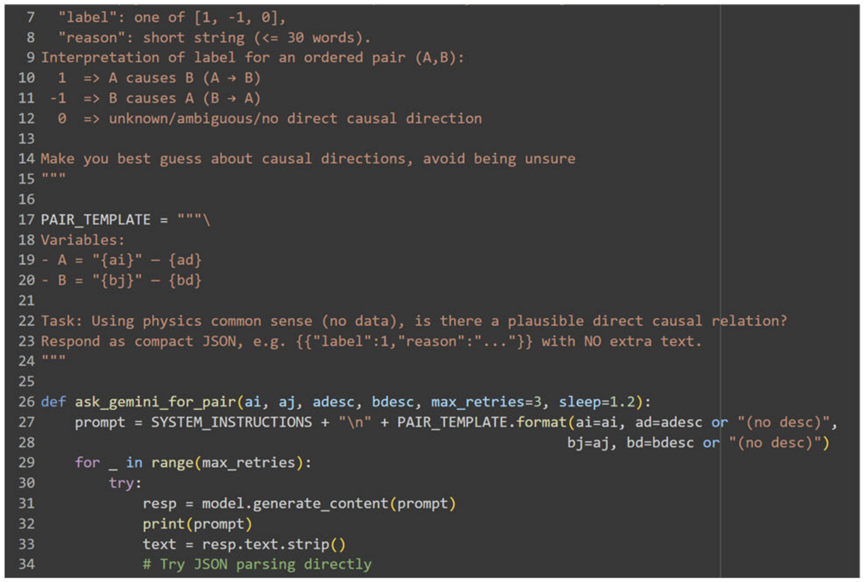
Kosmos: An AI Scientist for Autonomous Discovery

Ludovico Mitchener[*,1,†], Angela Yiu[*,1], Benjamin Chang[*,1,2], Mathieu Bourdenx[3,4,5], Tyler Nadolski[1], Arvis Sulovari[1], Eric C. Landsness[5,6], Dániel L. Barabási[7,8], Siddharth Narayanan[1], Nicky Evans[9], Shriya Reddy[10], Martha Foiani[3,4], Aizad Kamal[6], Leah P. Shriver[11,12,13], Fang Cao[10], Asmamaw T. Wassie[1], Jon M. Laurent[1], Edwin Melville-Green[1], Mayk Caldas[1], Albert Bou[1], Kaleigh F. Roberts[14], Sladjana Zagorac[15], Timothy C. Orr[6], Miranda E. Orr[6,16], Kevin J. Zwezdaryk[17,18,19], Ali E. Ghareeb[1], Laurie McCoy[1], Bruna Gomes[10], Euan A. Ashley[10], Karen E. Duff[3,4,5], Tonio Buonassisi[9,20], Tom Rainforth[2], Randall J. Bateman[5,6], Michael Skarlinski[1], Samuel G. Rodriques[1,7,‡], Michaela M. Hinks[1,†], Andrew D. White[1,7,‡]

Abstract

Data-driven scientific discovery requires iterative cycles of literature search, hypothesis generation, and data analysis. Substantial progress has been made towards AI agents that can automate scientific research, but all such agents remain limited in the number of actions they can take before losing coherence, thus limiting the depth of their findings. Here we present Kosmos, an AI scientist that automates data-driven discovery. Given an

There is no moat: build one yourself!



```
  "label": one of [1, -1, 0],
  "reason": short string (<= 30 words).
Interpretation of label for an ordered pair (A,B):
  1  => A causes B (A → B)
 -1  => B causes A (B → A)
  0  => unknown/ambiguous/no direct causal direction

Make you best guess about causal directions, avoid being unsure
"""


PAIR_TEMPLATE = """\
Variables:
- A = "{ai}" — {ad}
- B = "{bj}" — {bd}

Task: Using physics common sense (no data), is there a plausible direct causal relation?
Respond as compact JSON, e.g. {{"label":1,"reason":"..."}} with NO extra text.
"""

def ask_gemini_for_pair(ai, aj, adesc, bdesc, max_retries=3, sleep=1.2):
    prompt = SYSTEM_INSTRUCTIONS + "\n" + PAIR_TEMPLATE.format(ai=ai, ad=adesc or "(no desc)",
                                                              bj=aj, bd=bdesc or "(no desc)")
    for _ in range(max_retries):
        try:
            resp = model.generate_content(prompt)
            print(prompt)
            text = resp.text.strip()
            # Try JSON parsing directly
```

Competition

- Hackathon goals
- Value proposition
- Hackathon team and sponsors
- Logistics: Slack and Miro
- Data and problems

Value Proposition

1. **For participants:**
   - Solve actual problems
   - Upskilling and learning what to learn
   - Visibility/reputation
   - Build up GitHub repo
   - Job search

2. **For organizers:**
   - Visibility on organizational and personal level
   - Learning what is out there
   - Hiring strong teams

3. **For the community:**
   - Downstream integration: code and data ecosystem
   - Connection to the mainstream ML community
   - Steering commercial vendors

While a list of topics is provided, we are open to new ideas. If you have a project in mind that doesn’t fit into one of the topics below, you are welcome and encouraged to submit a project proposal for it.

Topic 1: Image and Hyperspectral Image Analysis

Topic 2: New Problems for Real-world Materials Imaging and Spectroscopy

Topic 3: Creating Instructional Tutorials

Topic 4: Active Learning and Real-time Analysis

Topic 5: Simulators for Active Learning

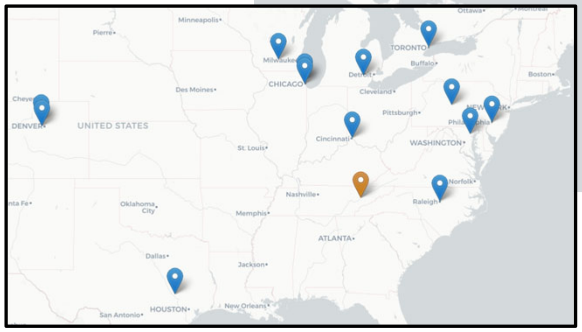
Topic 6: From Microscopy to ML and Back

Topic 7: Grand Challenge Problems

Topic 8: General

Participating Sites

NORTH AMERICA
EUROPE
ASIA
AFRICA
AUSTRALIA



US & Canada

1. University of Tennessee, Knoxville
2. North Carolina State University
3. Northwestern University
4. University of Illinois at Chicago
5. University of Wisconsin–Madison
6. University of Colorado Boulder
7. Colorado School of Mines
8. Johns Hopkins University
9. Pennsylvania State University
10. University of Pennsylvania
11. University of Michigan, Ann Arbor
12. Texas A&M University
13. University of Cincinnati
14. University of Toronto

International

15. ICN2 – Institut Català de Nanociència i Nanotecnologia (Spain)
16. University of Cambridge
17. Indian Institute of Technology Delhi (India)
18. Sungkyunkwan University (Republic of Korea)
19. Nanyang Technological University (NTU), Singapore
20. AISCIA Informatics – Doha, Qatar
21. Thermo Fisher Scientific – Eindhoven (Netherlands)

And technology

"New directions in science are launched by new tools much more often than by new concepts. The effect of a concept-driven revolution is to explain old things in new ways. The effect of a tool-driven revolution is to discover new things that have to be explained."

Freeman Dyson