PSLQ

Kevin Ly
AIG

Disclaimer

PSLQ was picked as a top 10 algorithm by two editors of *CiSE*. Their words: “We tried to assemble the 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century.”

Some others that made the cut: Monte Carlo methods (Neumann, Ulam, Metropolis), decompositions in matrix computations (Householder), Quicksort (Hoare), FFT (Cooley and Tukey)

- What is PSLQ?
- How does it work? (briefly)
- ~~Why does it work?~~
- Applications / demos

What is PSLQ?

PSLQ is an algorithm for finding integer relations. Given a vector $x = (x_1, \ldots, x_n)$, an integer relation for x (if it exists) is an *integer* vector $m = (m_1, \ldots, m_n)$ such that $\sum_i x_i m_i = 0$

Specifically, each iteration of PSLQ produces a lower bound on the norm $|m|$ for all possible integer relations

Wait…isn't that easy?

Just generate a vector orthogonal to x and rescale it until you have integers?

Suppose $x =$ (1, 3.65028154, 13.32455532, 48.63837831, 177.54377448).
One orthogonal vector is $x_o =$ (3.65028154, -1, 0, 0, 0). Rescale it to $m =$ (182513077, 50000000, 0, 0, 0), and there's your integer relation!

But…is there a smaller one?

PSLQ attempts to find the *smallest* integer relation, up to some precision

“PSLQ operates by constructing a sequence of integer-valued matrices B_n that reduces the vector $y = xB_n$, until either the relation is found (as one of the columns of B_n), or else precision is exhausted.”

the relation is real and not merely a numerical artifact. A drop of 20 or more orders of magnitude almost always indicates a real relation (see Figure 1).

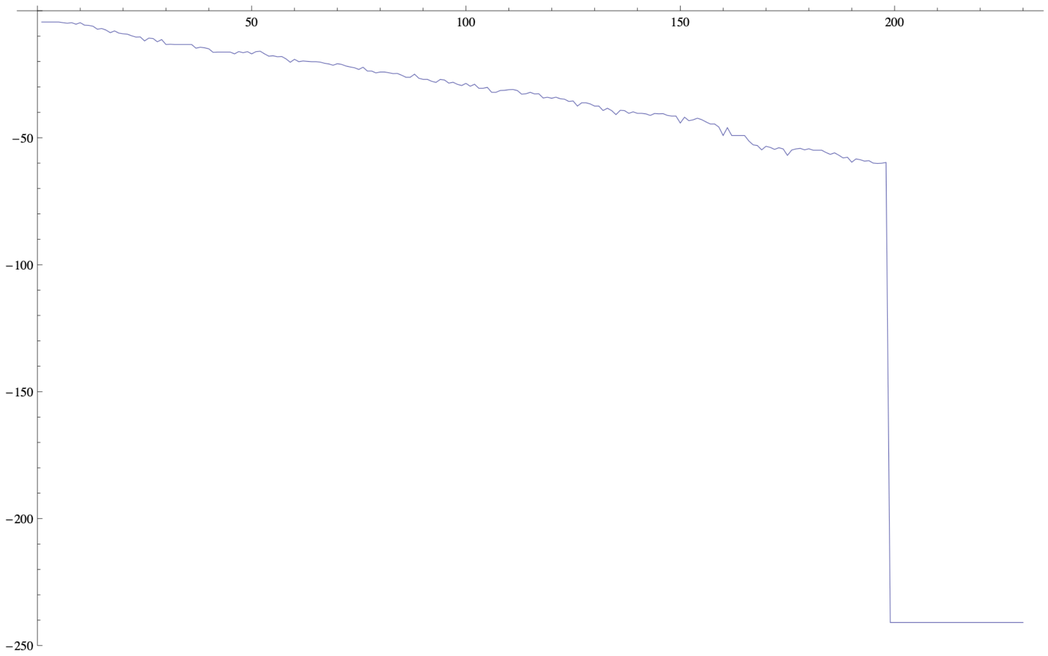


Figure 1: $\log_{10} \min_k |y_k|$ versus iteration number in a typical PSLQ run

excerpt and figure from Bailey & Borwein, *PSLQ: An Algorithm to Discover Integer Relations*

Pseudocode

Initialize:

1. Set the $n \times n$ matrices **A** and **B** to the identity.
2. For $k := 1$ to **n**: Set $s_k := \sqrt{\sum_{j=k}^{n} x_j^2}$; endfor. Set $t := s_1$. For $k := 1$ to **n**: $y_k := x_k / t$; $s_k := s_k / t$; endfor.
3. Compute the $n \times (n-1)$ matrix **H** as follows:

   For $i := 1$ to **n**: for $j := i+1$ to **n - 1**: set $H_{ij} := 0$; endfor; if $i \leq n-1$ then set $H_{ii} := s_{i+1}/s_i$; for $j := 1$ to **i - 1**: set $H_{ij} := -y_i y_j / (s_j s_{j+1})$; endfor; endfor.
4. Perform full reduction on **H**, simultaneously updating y, A and **B**:

   For $i := 2$ to **n**: for $j := i-1$ to **1** step **-1**: $t := \mathrm{nint}(H_{ij}/H_{jj})$; $y_j := y_j + t y_i$; for $k := 1$ to **j**: $H_{ik} := H_{ik} - t H_{jk}$; endfor; for $k := 1$ to **n**: $A_{ik} := A_{ik} - t A_{jk}$, $B_{kj} := B_{kj} + t B_{ki}$; endfor; endfor; endfor.

Repeat until precision is exhausted or a relation has been detected:

1. Select **m** such that $\gamma^i |H_{ii}|$ is maximal when **i = m**.
2. Exchange entries **m** and **m + 1** of **y**, corresponding rows of **A** and **H**, and corresponding columns of **B**.
3. If $m \leq n-2$ then update **H** as follows:

   Set $t_0 := \sqrt{H_{mm}^2 + H_{m,m+1}^2}$, $t_1 := H_{mm}/t_0$ and $t_2 := H_{m,m+1}/t_0$. Then for $i := m$ to **n**: $t_3 := H_{im}$; $t_4 := H_{i,m+1}$; $H_{im} := t_1 t_3 + t_2 t_4$; $H_{i,m+1} := -t_2 t_3 + t_1 t_4$; endfor.
4. Perform block reduction on **H**, simultaneously updating y, A and **B**:

   For $i := m+1$ to **n**: for $j := \min(i-1, m+1)$ to **1** step **-1**: $t := \mathrm{nint}(H_{ij}/H_{jj})$; $y_j := y_j + t y_i$; for $k := 1$ to **j**: $H_{ik} := H_{ik} - t H_{jk}$; endfor; for $k := 1$ to **n**:
   $A_{ik} := A_{ik} - t A_{jk}$, $B_{kj} := B_{kj} + t B_{ki}$; endfor; endfor; endfor.
5. Norm bound: Compute $M := 1/\max_j |H_j|$, where H_j denotes the **j**-th row of **H**. Then there can exist no relation vector whose Euclidean norm is less than **M**.
6. Termination test: If the largest entry of **A** exceeds the level of numeric precision used, then precision is exhausted. If the smallest entry of the **y** vector is less than the detection threshold, a relation has been detected and is given in the corresponding column of **B**.

mpmath.pslq()

The pseudocode is relatively straightforward, but note the indices start from 1 and not 0. There is also another catch:

“If one wishes to recover a relation of length n, with coefficients of maximum size d digits, then the input vector x must be specified to at least nd digits, and one must employ nd-digit floating-point arithmetic.”

To save myself some trouble: I’m just going to use the algorithm as implemented in the mpmath library

Application: identifying algebraic numbers

Given a number α, let $x = (1, \alpha, \ldots, \alpha^n)$. An integer relation m for x would mean that α is a root to the polynomial $m_0 + m_1\alpha + \ldots m_n\alpha^n = 0$ so for simple enough polynomials, an expression for α can be deduced

Demo: identify 3.65028154

Solution

$$0 = -9 + 14\alpha^2 - \alpha^4$$

$$\implies \alpha = \sqrt{2\sqrt{10} + 7}$$

Application: Euler-Mashceroni constant

This constant is the limiting difference between the harmonic series and natural log

$$\gamma \equiv \lim_{n\to\infty} \left(-\log n + \sum_{k=1}^{n} \frac{1}{k} \right)$$

It is not known whether or not this is an algebraic number, or even if it is rational. With careful attention to precision, it can be checked with PSLQ that it is not the root of any integer polynomial of degree n, where you can try as large n as your machine can handle

Application: a new formula for pi

The formula

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right)$$

was found with PSLQ and subsequently proved. It can be shown that this can be used to compute the nth digit of pi in hexadecimal without having to compute the previous $n-1$ digits

Application: a new formula for pi

Found by looking for an integer relation for $x = (X_1, \ldots, X_8, \pi)$ where

$$X_j \equiv \sum_{k=0}^{\infty} \frac{1}{16^k(8k+j)}$$

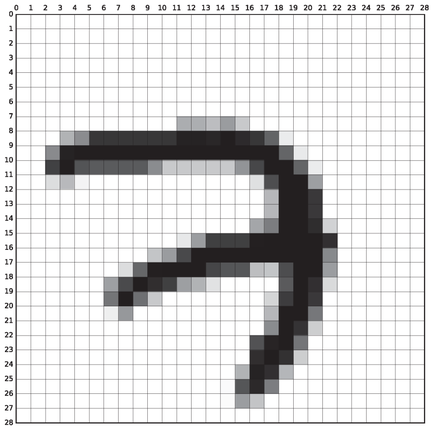
Summary

PSLQ:

- is an algorithm which either finds integer relations or bounds for possible relations
- is commonly used in number recognition routines
- has led numerous “discoveries,” notably a new formula for π, but also other more technical discoveries that were not covered here

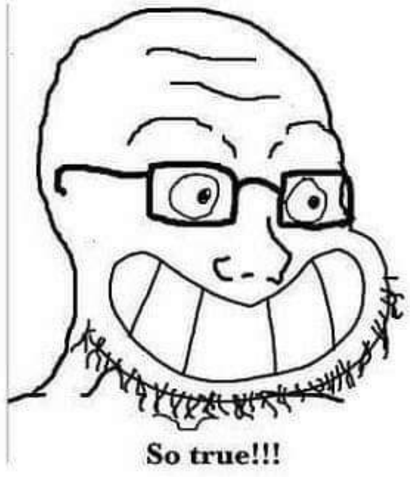
We also went over how to “use” PSLQ, paying attention to the behavior of the “error” over time, as well as the precision used

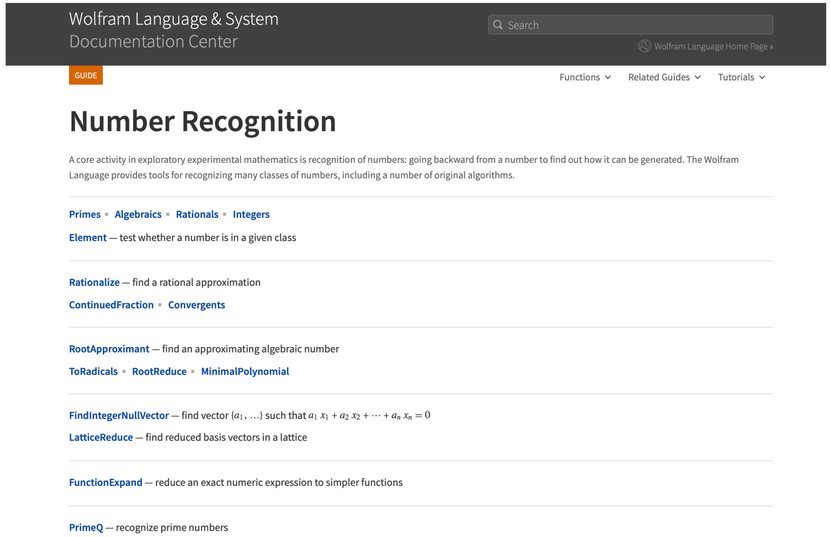
Summary



(**a**) MNIST sample belonging to the digit ‘7’.

(**b**) 100 samples from the MNIST training set.





References

- Bailey and Borwein, *PSLQ: An Algorithm to Discover Integer Relations*, from Bailey's personal website

- Ferguson, Bailey, and Arno, *Analysis of PSLQ, an integer relation finding algorithm* (1999)

- Pseudocode (courtesy of Bailey): http://www.cecm.sfu.ca/organics/papers/bailey/paper/html/node3.html

- mpmath: http://mpmath.org