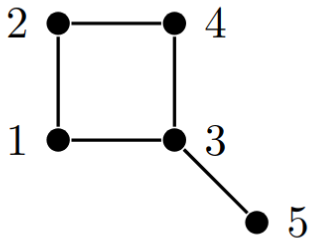
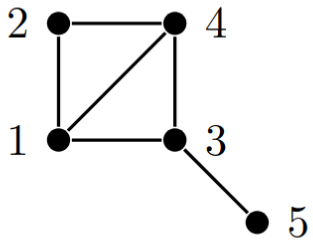
Problem 1: Appreciate Cholesky Graphically

Problem Statement



(a) Graph G_1



(b) Graph G_2

a

Find a matrix $\mathbf{A} \in \mathbb{R}^{5\times5}$ s.t $\mathcal{G}(\mathbf{A}) = G_1$ in the figure above, and s.t. $\mathbf{A} \succ 0$. *Hint: Associate positive numbers or weights to each node and each edge of* G_1. *Defin* $A_{ij} = A_{ji}$ *as the negative of the weight on the edge* (i, j) *of* G_1. *Defin* A_{ii} *to be the sum of the weight of node* i *and the weights of all edges connected to node* i. *Such a matrix is called a Laplacian matrix for* G_1. *Compute the eigenvalues to verify it is indeed PD.*

b

Compute the lower triangular Cholesky factor $\mathbf{L} \in \mathbb{R}^{5\times5}$ with positive diagonals such that $\mathbf{LL}^T$ equals $\mathbf{A}$ in part (a). Draw a graph on 5 nodes that describes the sparsity pattern of $\mathbf{L}$; i.e. draw $G(\mathbf{L})$. Verify that $G(\mathbf{L})$ is a chordal graph.

c

Notice that G_2 in figure 1b is a chordal graph. Find a permutation matrix $\mathbf{P}$ s.t. that when you compute the Cholesky factor $\mathbf{L}'$ of $\mathbf{PAP}^T$ with your $\mathbf{A}$ from part (a), then $G(\mathbf{L}') = G_2$. *Hint: Eliminate the nodes of* G_1 *in the sequence* $(5, 2, 3, 1, 4)$. *Encode that elimination order in a permutation matrix.*

d(**)

Let $\mathbf{X} \in \mathbb{R}^{n\times n}$ be a PD matrix and $\mathbf{\Gamma} \in \mathbb{R}^{n\times n}$ be a lower triangular matrix with positive diagonals, s.t. $\mathbf{X} = \mathbf{\Gamma\Gamma}^T$. Prove that $G(\mathbf{X})$ is a subgraph of $G(\mathbf{\Gamma})$ and $G(\mathbf{\Gamma})$ is a chordal graph. *Hint: Use an induction argument on the size of the matrix. Characterize how the steps in Cholesky factorization shapes the sparsity pattern of* $\mathbf{\Gamma}$. *Finally, utilize the fact that if a node and a set of edges from that node is added to a chordal graph* G, *then the obtained graph is chordal IFF every two neighbors of the new node in* G *already shared an edge in* G.

Solution

a

b

c

d

Problem 2: Split 'em matrices for Linear Systems

Problem Statement

Consider a linear system of equations $Ax = b$ where $A \in \mathbb{R}^{n\times n}$ and $\exists A^{-1}$. The matrix splitting method splits A as $A = M - n$ and successively solves the equation:

$$x^{k+1} = M^{-1}\left(Nx^k + b\right) \tag{1}$$

for $k \geq 0$ to perform a fixed-point iteration in order to solve the linear system $Ax = b$. Identify L, D and U as the strictly lower triangular, diagonal, and strictly upper triangular part of A. Then, $A = L + D + U$. Whem $M = L + D$, then the fixed point iteration becomes:

$$x_i^{k+1} \coloneqq \frac{1}{A_{ii}}\left[b_i - \sum_{j<i} A_{ij}x_j^{k+1} - \sum_{j>i} A_{ij}x_j^k\right] \tag{2}$$

for $i = 1, \ldots, n$ and $k \geq 0$.

a

Write the update equation for each element of x as in equation 2 when $M = D+U$ using backward induction to compute the iterates in equation 1 successively.

b

Call the method you devised in part (a) as the *backward* Gauss-Seidel method. Which among this method and Jacobi methods would you expect to converge faster to the solution of $Ax = b$? Which one will you choose if distributed computation is involved, and why?

c

Implement both backward Gauss-Seidel and Jacobi methods to solve the linear system of equations $Ax = b$ with

$$A \coloneqq \begin{pmatrix} 10 & 5 & 3 & 4 \\ 4 & 10 & 2 & 1 \\ 1 & 3 & 8 & 2 \\ 1 & 6 & 3 & 9 \end{pmatrix}, \; b \coloneqq \begin{pmatrix} 4 \\ -5 \\ 4 \\ -11 \end{pmatrix}$$

starting with $x^0 \coloneqq (1, 1, 1, 1)^T$. Plot the residues and errors in the succesive iterates on a semilog plot. Iterate until the residue falls below 10^-5. The error and residue, respectively, at iteration k are given by:

$$\epsilon^k \coloneqq \|x - x^*\|, \; r^k \coloneqq \|b - Ax^k\|$$

where x^* is the solution to the linear system.

d

Recall that the method of succesive over-relaxation (SOR) uses $M = \frac{1}{\omega}D + L$. Any matrix splitting method converges to the solution of a linear system *IF* the spectral radius of the matrix $I - M^{-1}A$ is less than 1. Plot the spectral radius of $I - M^{-1}A$ for SOR as a function of ω in the range $[0.01, 2.20]$ for 20 randomly generated PD A matrices on the same graph. Based on your plot, can you guess for what values of ω SOR converges for PD matrices?

Solution

a

$$x_i^{k+1} := \frac{1}{A_{ii}} \left[b_i - \sum_{j>i} A_{ij} x_j^{k+1} - \sum_{j<i} A_{ij} x_j^k \right]$$

b

It will depend. Gauss-Seidel methods use more "up to date" information, and so intuitively we would expect faster convergence. However, this is not a hard and fast rule, and there are counterexamples wherein one method converges and the other doesn't, OR that one will converge faster in certain cases, or slower in others. See [1].

I think that using a typical scatter-gather scheme for distributed computation, Jacobi iteration would be preferred for distributed computing. Each element of x^{k+1} can be calculated in an embarrasingly parallel fashion (i.e. each element can be calculated in a fully independent fashion), whereas Gauss-Seidel will require iterations and sharing current estimates of x^{k+1} around, which will limit the ability to parallelize.

c

d

Code

```python
#!/usr/bin/env python3
import time
from typing import Callable, Iterable

import numpy as np
import torch

N_FACT = np.sqrt(2 * np.pi * 100) * (100 / np.e) ** 100


def naive_loop(x: Iterable) -> None:
    sum_ = 0.0
    for i in x:
        sum_ += i


def numpy_vector_op(x: np.array) -> None:
    sum_ = np.sum(x)


def torch_cpu_or_gpu(x: torch.Tensor) -> None:
    sum_ = torch.sum(x)


def estimate_flops(
    f: Callable[[int], float], n: int, lib: str = "numpy", iterations: int = 10
) -> float:
    times_ns = []
    if lib == "native":
        x = [1.0] * n
    elif lib == "numpy":
        x = np.ones(n, dtype=np.float64)
    elif lib == "torch_cpu":
        x = torch.ones(n, dtype=torch.float64)
    elif lib == "torch_gpu":
        x = torch.randn(n, dtype=torch.float64).to(torch.device("cuda:0"))
    else:
        raise RuntimeError(f"Invalid library {lib} requested!")

    for _ in range(iterations):
        torch.cuda.synchronize()
        t0 = time.perf_counter_ns()
        f(x)
        t1 = time.perf_counter_ns()
        times_ns.append(t1 - t0)

    avg_time_ns = np.average(np.array(times_ns))

    return n / (avg_time_ns / int(1e9))


def print_results(desc: str, flops: float) -> None:
    naivest_loop_flops = estimate_flops(naive_loop, int(n), "native")
    flop_time = 1.0 / flops
    print(f"{desc}: {flops:.2E} FLOPS, or {flop_time:.2E} seconds for 1 FLOP")
    print(f"Time for 100x100 Matrix: {(N_FACT/3.15576e+7)*flop_time} years")


if __name__ == "__main__":
    n = 1e6
    naivest_loop_flops = estimate_flops(naive_loop, int(n), "native")
    print_results("Python list, naive for loop", naivest_loop_flops)

    naive_loop_flops = estimate_flops(naive_loop, int(n), "native")
    print_results("NumPy array, naive for loop", naive_loop_flops)

    n = 1e8
    numpy_flops = estimate_flops(numpy_vector_op, int(n), "numpy")
    print_results("NumPy array, vectorized sum", numpy_flops)

    torch_cpu_flops = estimate_flops(torch_cpu_or_gpu, int(n), "torch_cpu")
    print_results("Torch Tensor (CPU), vectorized sum", torch_cpu_flops)

    torch_gpu_flops = estimate_flops(torch_cpu_or_gpu, int(n), "torch_gpu")
    print_results("Torch Tensor (GPU), vectorized sum", torch_gpu_flops)
```

Bibliography

[1] Stewart Venit. "The Convergence of Jacobi and Gauss-Seidel Iteration". In: *Mathematics Magazine* 48.3 (May 1975), pp. 163–167.