GAMSPy - Where Convenience Meets Performance

Muhammet Abdullah Soyturk

2024-05-09

Table of contents

What is GAMSPy?

- **Fast**: Compute intensive operations are done by GAMS.
- **Convenient**: Switch solvers, backends with minimal change.
- **Readable**: Syntax is very close to what you would write on a paper.
- **Interoperable**: One can easily convert their model to other formats (LaTeX, GAMS).

- GAMSPy is a mathematical modelling language in Python.
- Fast: Offloads compute intensive work to GAMS execution engine. Just like other libraries like Gurobipy.
- Convenient: You can easily switch solvers and backends. You can also combine your favori Python library with GAMSPy.

Learn more about: GAMSPy

Philosophy

Mathematical Model

$$\sum_{i \in \mathcal{I}} \frac{p_{i,j} - q_i}{a_j} \cdot x_{i,j} \leq \sum_{k \in \mathcal{K}} d_{k,j} \qquad \forall\, j \in \mathcal{J}$$

Model Instance

$$5 \cdot x_{i1,j1} + 3 \cdot x_{i2,j1} + 2 \cdot x_{i3,j1} \leq 7$$
$$2 \cdot x_{i1,j2} + 6 \cdot x_{i2,j2} + 4 \cdot x_{i3,j2} \leq 10$$

Think of a mathematical model as a pure representation of mathematical symbols without any data. In contrast, a model instance is the unrolled representation of a model with its actual data.

Programming the mathematical model is more intuitive, hence faster.

Philosophy

Other Libraries

```
import other_library as ol

data = get_data()
m = ol.Model()
m.i = ol.Set(records=data.i)
m.j = ol.Set(records=data.j))
m.a = ol.Parameter(m.i, m.j, records=data.a)
m.b = ol.Parameter(m.i, records=data.b)
m.c = ol.Parameter(m.j, records=data.c)
m.x = ol.Variable(m.j, type="Positive")

def constraint(m, i):
  return sum(
      m.a[i, j] * m.x[j] for j in m.j
    ) >= m.b[i]
m.const = ol.Equation(m.i, definition=constraint)

def objective(m, j):
  return sum(m.c[j] * m.x[j] for j in m.j)
m.obj = ol.Equation(m.i, definition=objective)
m.solve()
```

GAMSPy

```
import gamspy as gp

m = gp.Container()
i = gp.Set(m)
j = gp.Set(m)
a = gp.Parameter(m, domain=[i, j])
b = gp.Parameter(m, domain=i)
c = gp.Parameter(m, domain=j)
x = gp.Variable(m, domain=j, type="Positive")
e = gp.Equation(m, domain=i)
e[i] = gp.Sum(j, a[i, j] * x[j]) >= b[i]
model = gp.Model(
```

```
  m, equations=[e], sense="min", problem="lp",
  objective=gp.Sum(j, c[j] * x[j])
)

m.loadRecordsFromGdx("input.gdx")
model.solve()
```

- Model Instance with data. No data required for mathematical model.
- Syntax is much simpler. No explicit for loops.

Convenience: Switching Solvers

```
import gamspy as gp
m = gp.Container()

x1 = gp.Variable(m)
x2 = gp.Variable(m)
x3 = gp.Variable(m)
eq1 = gp.Equation(m)
eq2 = gp.Equation(m)
eq3 = gp.Equation(m)
eq1[...] = x1 + 2 * x2 >= 3
eq2[...] = x3 + x2 >= 5
eq3[...] = x1 + x3 == 4
obj = x1 + 3 * x2 + 3 * x3

LP1 = gp.Model(
  m, equations=m.getEquations(),
  problem="lp", sense="min", objective=obj
)
LP1.solve(solver="CPLEX")
```

Convenience: Switching Solvers

```
import gamspy as gp
m = gp.Container()

x1 = gp.Variable(m)
x2 = gp.Variable(m)
```

```
x3 = gp.Variable(m)
eq1 = gp.Equation(m)
eq2 = gp.Equation(m)
eq3 = gp.Equation(m)
eq1[...] = x1 + 2 * x2 >= 3
eq2[...] = x3 + x2 >= 5
eq3[...] = x1 + x3 == 4
obj = x1 + 3 * x2 + 3 * x3

LP1 = gp.Model(
  m, equations=m.getEquations(),
  problem="lp", sense="min", objective=obj
)
LP1.solve(solver="IPOPTH")
```

Convenience: Switching Backends

```
import gamspy as gp
m = gp.Container()

x1 = gp.Variable(m)
x2 = gp.Variable(m)
x3 = gp.Variable(m)
eq1 = gp.Equation(m)
eq2 = gp.Equation(m)
eq3 = gp.Equation(m)
eq1[...] = x1 + 2 * x2 >= 3
eq2[...] = x3 + x2 >= 5
eq3[...] = x1 + x3 == 4
obj = x1 + 3 * x2 + 3 * x3

LP1 = gp.Model(
  m, equations=m.getEquations(),
  problem="lp", sense="min", objective=obj
)
client = gp.NeosClient(email="<your_email>")
LP1.solve(backend="neos", client=client)
```

Learn more about: NEOS Server

Convenience: Switching Backends

```python
import gamspy as gp
m = gp.Container()

x1 = gp.Variable(m)
x2 = gp.Variable(m)
x3 = gp.Variable(m)
eq1 = gp.Equation(m)
eq2 = gp.Equation(m)
eq3 = gp.Equation(m)
eq1[...] = x1 + 2 * x2 >= 3
eq2[...] = x3 + x2 >= 5
eq3[...] = x1 + x3 == 4
obj = x1 + 3 * x2 + 3 * x3

LP1 = gp.Model(
  m, equations=m.getEquations(),
  problem="lp", sense="min", objective=obj
)
client = gp.EngineClient(
  username="username", password="password"
)
LP1.solve(backend="engine", client=client)
```

- Up to 400 parallel machines.

Learn more about: GAMS Engine

Performance: Model Generation

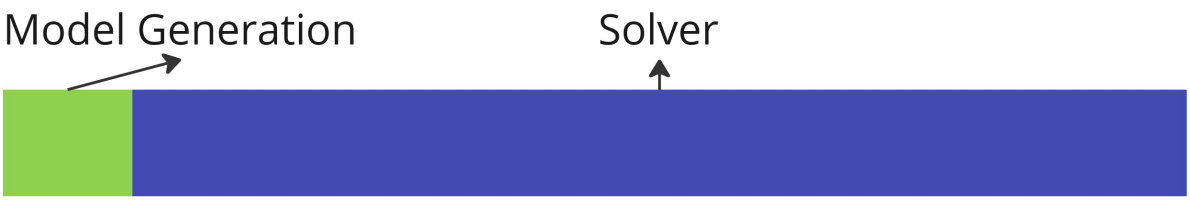

Figure 1: Common Case

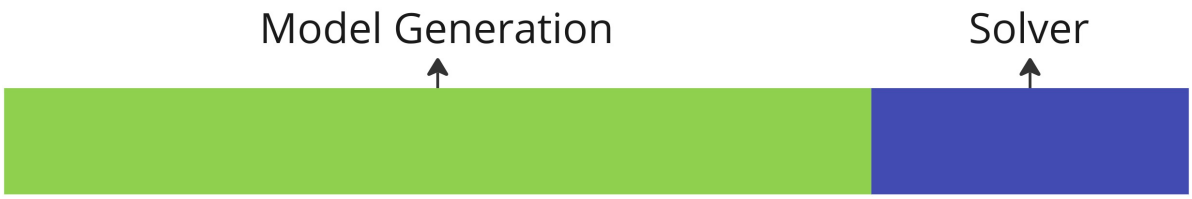

Figure 2: Rare Case

Performance: Model Generation

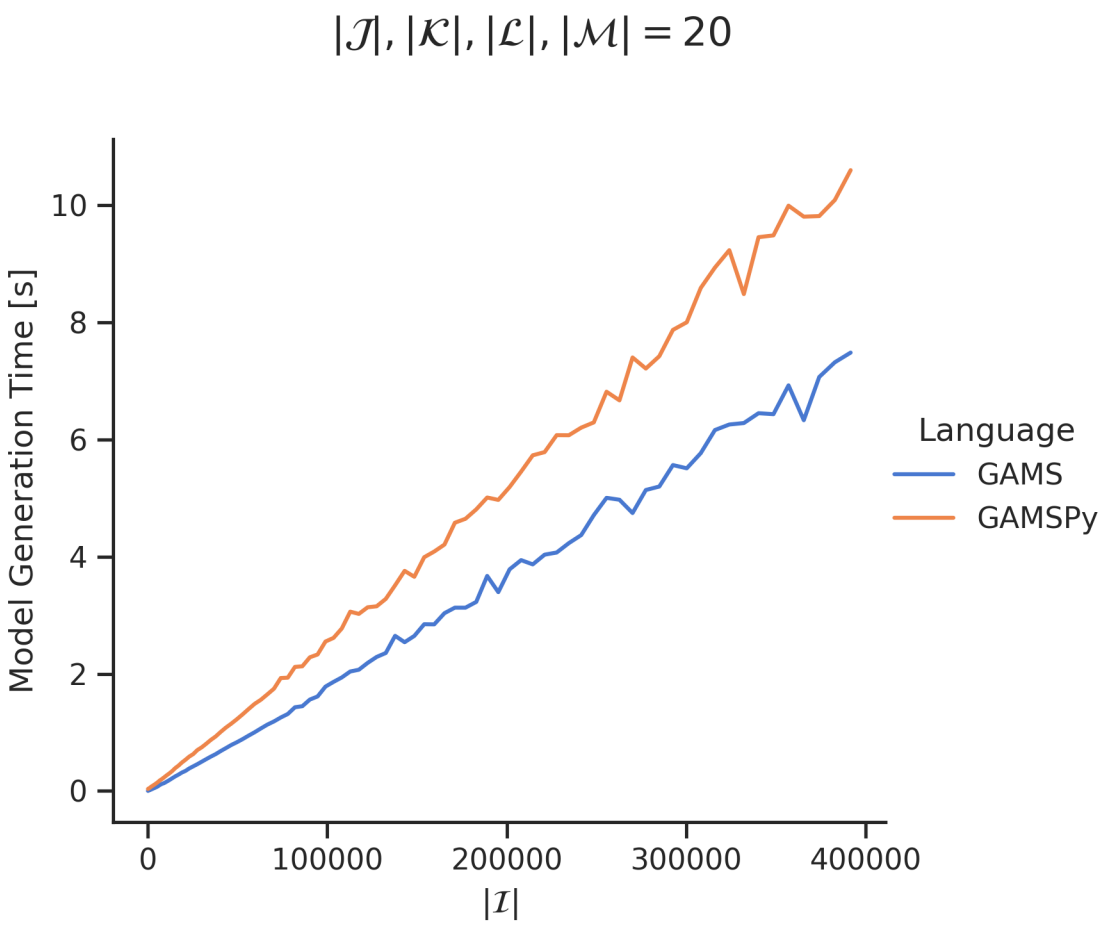

Figure 3: Model Generation Time

Check our blog post to see comparison with other libraries.

Performance: Frozen Solve

```
import gamspy as gp
m = gp.Container()

i, j = gp.Set(m), gp.Set(m)
a = gp.Parameter(m, domain=i)
b = gp.Parameter(m, domain=j)
d = gp.Parameter(m, domain=[i, j])
c = gp.Parameter(m, domain=[i, j])
m.loadRecordsFromGdx("input.gdx")
bmult = Parameter(m, records=1)
x = gp.Variable(m, "x", domain=[i, j], type="Positive")
supply = gp.Equation(m, domain=i)
demand = gp.Equation(m, domain=j)
supply[i] = gp.Sum(j, x[i, j]) <= a[i]
demand[j] = gp.Sum(i, x[i, j]) >= bmult * b[j]

transport = Model(m, equations=m.getEquations(),
                  problem="LP", sense=Sense.MIN,
                  objective=gp.Sum((i, j), c[i, j] * x[i, j]))
transport.freeze(modifiables=[bmult])
bmult.setRecords(2)
transport.solve()
```

Performance: Frozen Solve

```
import gamspy as gp
m = gp.Container()

i, j = gp.Set(m), gp.Set(m)
a = gp.Parameter(m, domain=i)
b = gp.Parameter(m, domain=j)
d = gp.Parameter(m, domain=[i, j])
c = gp.Parameter(m, domain=[i, j])
m.loadRecordsFromGdx("input.gdx")
bmult = Parameter(m, records=1)
x = gp.Variable(m, "x", domain=[i, j], type="Positive")
supply = gp.Equation(m, domain=i)
demand = gp.Equation(m, domain=j)
```

```
supply[i] = gp.Sum(j, x[i, j]) <= a[i]
demand[j] = gp.Sum(i, x[i, j]) >= bmult * b[j]

transport = Model(m, equations=m.getEquations(),
                  problem="LP", sense=Sense.MIN,
                  objective=gp.Sum((i, j), c[i, j] * x[i, j]))
transport.freeze(modifiables=[bmult])
bmult.setRecords(2)
transport.solve()
```

Performance: Frozen Solve

```
import gamspy as gp
m = gp.Container()

i, j = gp.Set(m), gp.Set(m)
a = gp.Parameter(m, domain=i)
b = gp.Parameter(m, domain=j)
d = gp.Parameter(m, domain=[i, j])
c = gp.Parameter(m, domain=[i, j])
m.loadRecordsFromGdx("input.gdx")
bmult = Parameter(m, records=1)
x = gp.Variable(m, "x", domain=[i, j], type="Positive")
supply = gp.Equation(m, domain=i)
demand = gp.Equation(m, domain=j)
supply[i] = gp.Sum(j, x[i, j]) <= a[i]
demand[j] = gp.Sum(i, x[i, j]) >= bmult * b[j]

transport = Model(m, equations=m.getEquations(),
                  problem="LP", sense=Sense.MIN,
                  objective=gp.Sum((i, j), c[i, j] * x[i, j]))
transport.freeze(modifiables=[bmult])
bmult.setRecords(3)
transport.solve()
```

Performance: Frozen Solve

```python
import gamspy as gp
m = gp.Container()

i, j = gp.Set(m), gp.Set(m)
a = gp.Parameter(m, domain=i)
b = gp.Parameter(m, domain=j)
d = gp.Parameter(m, domain=[i, j])
c = gp.Parameter(m, domain=[i, j])
m.loadRecordsFromGdx("input.gdx")
bmult = Parameter(m, records=1)
x = gp.Variable(m, "x", domain=[i, j], type="Positive")
supply = gp.Equation(m, domain=i)
demand = gp.Equation(m, domain=j)
supply[i] = gp.Sum(j, x[i, j]) <= a[i]
demand[j] = gp.Sum(i, x[i, j]) >= bmult * b[j]

transport = Model(m, equations=m.getEquations(),
                  problem="LP", sense=Sense.MIN,
                  objective=gp.Sum((i, j), c[i, j] * x[i, j]))
transport.freeze(modifiables=[bmult])
bmult.setRecords(3)
transport.solve()
transport.unfreeze()
```

Interoperability: LaTeX

```python
import gamspy as gp
m = gp.Container()

i = gp.Set(m, "i", description="canning plants")
j = gp.Set(m, "j", description="markets")
a = gp.Parameter(m, "a", domain=i, description="capacity of plant i in cases")
b = gp.Parameter(m, "b", domain=j, description="demand at market j in cases")
d = gp.Parameter(m, "d", domain=[i, j], description="distance in thousands of miles")
c = gp.Parameter(m, "c", domain=[i, j], description="transport cost in thousands of dollars p
x = gp.Variable(m, "x", domain=[i, j], type="Positive", description="shipment quantities in c
supply = gp.Equation(m, "supply", domain=i, description="observe supply limit at plant i")
demand = gp.Equation(m, "demand", domain=j, description="satisfy demand at market j")
supply[i] = gp.Sum(j, x[i, j]) <= a[i]
print(supply.latexRepr())
```

$$\sum_j x_{i,j} \leq a_i \qquad \forall i$$

Interoperability: LaTeX

```
import gamspy as gp
m = gp.Container()

i = gp.Set(m, "i", description="canning plants")
j = gp.Set(m, "j", description="markets")
a = gp.Parameter(m, "a", domain=i, description="capacity of plant i in cases")
b = gp.Parameter(m, "b", domain=j, description="demand at market j in cases")
d = gp.Parameter(m, "d", domain=[i, j], description="distance in thousands of miles")
c = gp.Parameter(m, "c", domain=[i, j], description="transport cost in thousands of dollars p
x = gp.Variable(m, "x", domain=[i, j], type="Positive", description="shipment quantities in c
supply = gp.Equation(m, "supply", domain=i, description="observe supply limit at plant i")
demand = gp.Equation(m, "demand", domain=j, description="satisfy demand at market j")
supply[i] = gp.Sum(j, x[i, j]) <= a[i]
demand[j] = gp.Sum(i, x[i, j]) >= b[j]

transport = Model(m, "transport", equations=m.getEquations(),
                  problem="LP", sense=Sense.MIN,
                  objective=gp.Sum((i, j), c[i, j] * x[i, j]))
transport.toLatex("latex_out_path")
```

Interoperability: LaTeX

Interoperability: GAMS

```
import gamspy as gp
m = Container()

i = gp.Set(m, "i", description="canning plants")
j = gp.Set(m, "j", description="markets")
a = gp.Parameter(m, "a", domain=i, description="capacity of plant i in cases")
b = gp.Parameter(m, "b", domain=j, description="demand at market j in cases")
d = gp.Parameter(m, "d", domain=[i, j], description="distance in thousands of miles")
```

```
c = gp.Parameter(m, "c", domain=[i, j], description="transport cost in thousands of dollars p
x = gp.Variable(m, "x", domain=[i, j], type="Positive", description="shipment quantities in c
supply = gp.Equation(m, "supply", domain=i, description="observe supply limit at plant i")
demand = gp.Equation(m, "demand", domain=j, description="satisfy demand at market j")
supply[i] = gp.Sum(j, x[i, j]) <= a[i]
demand[j] = gp.Sum(i, x[i, j]) >= b[j]

transport = Model(m, "transport", equations=m.getEquations(),
                  problem="LP", sense=Sense.MIN,
                  objective=gp.Sum((i, j), c[i, j] * x[i, j]))
transport.toGams("gams_out_path")
```

Interactivity: GAMS MIRO

Learn more about: GAMS MIRO

Interactivity: GAMS MIRO

Learn more about: GAMS MIRO

Academic Program

Learn more about: Academic Program

- Commercial integration license CPLEX, COPT, and MOSEK.
- Link-licenses are available for GUROBI and XPRESS.

Academic Program

Academic GAMSPy Licenses

- No size limitations.
- Valid for 12 months and easily renewable through our academic user portal.
- Full access to all available **open-source** solvers.
- Integrated licenses for selected **commercial solvers** (**CPLEX**, **COPT**, and **MOSEK**). Link-licenses are also available for **GUROBI** and **XPRESS**.

Node Licenses

- Activate on up to 2 fixed nodes.
- Unlimited concurrent usage.
- No internet connection required to use.

Network Licenses

- Activate on up to 2 concurrent nodes.
- Unlimited concurrent usage.
- Requires internet access during use.
- Ideal for virtual environments.

We are in negotiation with FICO to add integrated FICO license to the academic program as well.

Learn more about: Academic Program

Thank you!

- Documentation: Readthedocs
- GAMSPy Examples Repository: Examples
- GAMSPy 1.0 stable release is out on PyPI: PyPI
- Academic Program: Academic
- GAMS Forum: Forum

Visit us at our booth!

Muhammet Soyturk - GAMS Development