

## AdvanceSyn Toolkit Supplementary Materials

### Table of Contents

<b>S1. Writing an AdvanceSyn Model Specification (Type 1)</b> .....	2
<b>S2. Operation: genMO</b> .....	4
<b>S3. Operation: genNetwork</b> .....	4
<b>S4. Operation: genODE</b> .....	5
<b>S5. Operation: installdep</b> .....	6
<b>S6. Operation: LSA</b> .....	6
<b>S7. Operation: mergeASM</b> .....	7
<b>S8. Operation: printASM</b> .....	7
<b>S9. Operation: readmodel</b> .....	8
<b>S10. Operation: readflux</b> .....	8
<b>S11. Operation: runODE</b> .....	8
<b>S12. Operation: senGen</b> .....	9
<b>S13. Operation: cameo-fba</b> .....	9
<b>S14. Operation: cameo-medium-cpds</b> .....	10
<b>S15. Operation: cameo-medium-fba</b> .....	10
<b>S16. Operation: cameo-medium-pfba</b> .....	10
<b>S17. Operation: cameo-mutant-fba</b> .....	11
<b>S18. Operation: cameo-mutant-pfba</b> .....	11
<b>S19. Operation: cameo-pfba</b> .....	12
<b>S20. Operation: cameo-rxn-cpds</b> .....	12
<b>S21. Operation: cameo-rxn-names</b> .....	13
<b>S22. Example: Glycolysis Model</b> .....	13
<b>S23. Example: Pentose Phosphate Pathway Model</b> .....	14
<b>S24. Example: Merged Glycolysis / Pentose Phosphate Pathway</b> .....	15
<b>S25. Example: Glycolysis / Pentose Phosphate Pathway Simulation Script</b> .....	17
<b>S26. Example: Execution of Simulation Script</b> .....	24
<b>S27. Example: Running Sensitivity Analysis</b> .....	25
<b>S28. Example: Simplified Full Model from Yeoh et al. (2020)</b> .....	25
<b>S29. Example: Simplified Cell Growth Model from Yeoh et al. (2020)</b> .....	26
<b>S30. Example: Simplified Vanillin Production Model from Yeoh et al. (2020)</b> .....	26
<b>S31. Example: Simplified Merged Model from Yeoh et al. (2020)</b> .....	27

## S1. Writing an AdvanceSyn Model Specification (Type 1)

AdvanceSyn Model (ASM) Specification is based on INI file, which is commonly used for software configuration.

In the simplest sense, a ASM file is a set of key-value pairs in the format of "key : value". For example, ip\_address : 100.0.0.1 means that the IP address (ip\_address) is set to 100.0.0.1. The main thing to note is that key must be a single string (no spaces in between). For organization purposes, key-value pairs can be grouped into sections. In this case, each key within a section must be unique.

Six sections are defined in ASM (Type 1):

1. Specification: Defining the ASM version.
2. Identifiers: Providing metadata about the model.
3. Objects: Defining the objects / entities in the model.
4. Initials: Defining the initial values of each object / entity.
5. Variables: Defining changeable variables used in the model.
6. Reactions: Defining the reactions / flows between each object / entity in the model.

### Example Reaction

As an example, we will define a chemical reaction  $A + B = P + Q$  where the rate term can be defined as  $bAB$ . This means that the rate where substrates A and B forms products P and Q is  $bAB/\text{second}$  -  $b(\text{concentration of A})(\text{concentration of B})$ .

Written in ordinary differential equations (ODEs),

$$\begin{aligned}dA/dt &= -bAB \\dB/dt &= -bAB \\dP/dt &= bAB \\dQ/dt &= bAB\end{aligned}$$

In another words, coefficient b is a rate variable.

### Specification Section

Specification section is used to tell AdvanceSyn Toolkit what specification type or version this ASM file is written under. Yes, it is very likely that we will have different specification versions in future. For example,

```
[Specification]
type: 1
```

defines this file as ASM specification type 1.

### Identifiers Section

Identifiers section is for the author(s) to put in descriptions and other details to describe this model. This section is mainly for description and will not be used by AdvanceSyn Toolkit for processing; thus, giving the author(s) rather free rein to describe the model. For example,

```
[Identifiers]
name: 2 substrates to 2 products
author: Maurice Ling
```

## Objects Section

Objects section defines all the objects (in this case, molecules) used in this model. The key is the molecule name while the value is the corresponding molecule descriptor. For example,

```
[Objects]
A: molecule A
B: molecule B
P: molecule P
Q: molecule Q
```

## Initials Section

Initials section defines the initial conditions (in this case, initial concentrations of the molecules). For example,

```
[Initials]
A: 1e-3
B: 1e-3
P: 0.0
Q: 0.0
```

defines that molecules A and B are 1 mM each while there is no molecules P and Q at the start of the reaction.

## Variables Section

Variables section defines the changeable variables used in the model. These variables will be used in define the reactions. In this case, there is only 1 variable, b, as

```
[Variables]
b: 100
```

## Reactions Section

Reactions section defines the flow or movement within the model. The keys are just unique identifiers in this model. It is the values that are important. For reactions, the values have 2 parts, flow and rate term (or rate law), separated by |. For example,

```
[Reactions]
r1: A + B -> P + Q | ${Variables:b} * A * B
```

defines reaction r1 as  $A + B \rightarrow P + Q$  (A and B to produce P and Q). The rate term is  $\text{\${Variables:b}} * A * B$  where  $\text{\${Variables:b}}$  takes the value of b in Variable section, resulting in the final rate term as  $100 * A * B$  in this case.

## Full Resulting Model

```
[Specification]
type: 1

[Identifiers]
name: 2 substrates to 2 products
```

author: Maurice Ling

[Objects]

A: molecule A

B: molecule B

P: molecule P

Q: molecule Q

[Initials]

A: 1e-3

B: 1e-3

P: 0.0

Q: 0.0

[Variables]

b: 100

[Reactions]

r1: A + B -> P + Q | \${Variables:b} \* A \* B

## S2. Operation: genMO

**Synopsis:** Read the AdvanceSyn model specification file(s) and generate a file consisting of the internal model objects.

**Usage:** python astools.py genMO [option]

where [option] can be

- modelfile: Relative path to the model specification file.
- outputfile: Relative path to the output model objects file.
- prefix: Prefix for new reaction IDs. This prefix cannot be any existing prefixes in any of the model specifications to be merged. Default = 'exp'.

**For example:**

```
python astools.py readflux \  
  --mtype=ASM \  
  --modelfile=models/asm/glycolysis.modelspec
```

## S3. Operation: genNetwork

**Synopsis:** Read the AdvanceSyn model specification file(s) and generate a network / reaction visualization file.

**Usage:** python astools.py genNetwork [option]

where [option] can be

- modelfile: Relative path(s) to the model specification file(s), separated by semi-colon.
- outputfile: Relative path to the output file.

- outfmt: Type of network visualization format to generate. Allowable options are 'SIF' (Simple Interaction Format). Default = 'SIF' (Simple Interaction Format).

**For example:**

```
python astools.py genNetwork \
    --outfmt=SIF \
    --
modelfile=models/asm/glycolysis.modelspec;models/asm/ppp.model
spec \
    --outputfile=glycolysis_ppp.sif
```

## S4. Operation: genODE

**Synopsis:** Generate Python ODE script from a given model specification file.

**Usage:** python astools.py genODE [option]

where [option] can be

- modelfile: Name of model specification file in models folder. This assumes that the model file is not in models folder.
- type: Type of model specification file. Allowable types are 'ASM' (AdvanceSyn Model Specification). Default = 'ASM'.
- solver: Type of solver to use. Allowable types are 'Euler', 'Heun' (Runge-Kutta 2nd method or Trapezoidal), 'RK3' (third order Runge-Kutta), 'RK4' (fourth order Runge-Kutta), 'RK38' (fourth order Runge-Kutta method, 3/8 rule), 'CK4' (fourth order Cash-Karp), 'CK5' (fifth order Cash-Karp), 'RK4F' (fourth order Runge-Kutta-Fehlberg), 'RK5F' (fifth order Runge-Kutta-Fehlberg), 'DP4' (fourth order Dormand-Prince), and 'DP5' (fifth order Dormand-Prince). Default = 'RK4'.
- timestep: Time step interval for simulation. Default = 1.0.
- endtime: Time to end simulation - the simulation will run from 0 to end time. Default = 21600.
- lowerbound: Define lower boundary of objects. For example, "1;2" means that when the value of the object hits 1, it will be bounced back to 2. Default = 0;0; that is, when the value of the object goes to negative, it will be bounced back to zero.
- upperbound: Define upper boundary of objects. For example, "10;9" means that when the value of the object hits 1, it will be pushed down to 9. Default = 1e-3;1e-3; that is, when the value of the object above 1e-3, it will be pushed back to 1e-3.
- odefile: Python ODE script file to write out. This file will be written into odescript folder. Default = odescript.py.

**For example:**

```
python genODE \
    --modelfile=models/asm/glycolysis.modelspec \
    --mtype=ASM \
    --solver=RK4 \
    --timestep=1 \
    --endtime=21600 \
    --lowerbound=0;0 \
    --upperbound=1e-3;1e-3 \
```

```
--odefile=glycolysis.py
```

## S5. Operation: installdep

**Synopsis:** Install external tools and dependencies. List of external tools and dependencies that will be installed:

- bokeh (<https://bokeh.pydata.org>), BSD 3-Clause "New" or "Revised" License
- cameo (<https://github.com/biosustain/cameo>), Apache Licence 2.0

**Usage:** `python astools.py installdep`

## S6. Operation: LSA

**Synopsis:** Perform local sensitivity analysis using OFAT/OAT (one factor at a time) method where the last data time (end time) simulation results are recorded into results file.

**Usage:** `python astools.py LSA [option]`

where [option] can be

- **modelfile:** Name of model specification file in models folder. This assumes that the model file is not in models folder.
- **multiple:** Multiplier to change each variable value. Default = 100 (which will multiply the original parameter value by 100).
- **prefix:** A prefixing string for the set of new model specification for identification purposes. Default = "".
- **type:** Type of model specification file. Allowable types are 'ASM' (AdvanceSyn Model Specification). Default = 'ASM'.
- **solver:** Type of solver to use. Allowable types are 'Euler', 'Heun' (Runge-Kutta 2nd method or Trapezoidal), 'RK3' (third order Runge-Kutta), 'RK4' (fourth order Runge-Kutta), 'RK38' (fourth order Runge-Kutta method, 3/8 rule), 'CK4' (fourth order Cash-Karp), 'CK5' (fifth order Cash-Karp), 'RK4F' (fourth order Runge-Kutta-Fehlberg), 'RK5F' (fifth order Runge-Kutta-Fehlberg), 'DP4' (fourth order Dormand-Prince), and 'DP5' (fifth order Dormand-Prince). Default = 'RK4'.
- **timestep:** Time step interval for simulation. Default = 1.0.
- **endtime:** Time to end simulation - the simulation will run from 0 to end time. Default = 21600.
- **lowerbound:** Define lower boundary of objects. For example, "1;2" means that when the value of the object hits 1, it will be bounced back to 2. Default = 0;0; that is, when the value of the object goes to negative, it will be bounced back to zero.
- **upperbound:** Define upper boundary of objects. For example, "10;9" means that when the value of the object hits 1, it will be pushed down to 9. Default = 1e-3;1e-3; that is, when the value of the object above 1e-3, it will be pushed back to 1e-3.
- **cleanup:** Flag to determine whether to remove all generated temporary models and ODE code files. Default = True.
- **outfmt:** Output format. Allowable types are 'reduced' (only the final result will be saved into resultfile) and 'full' (all data, depending on sampling, will be saved into resultfile).

- **sampling:** Sampling frequency. If 100, means only every 100th simulation result will be written out. The first (start) and last (end) result will always be written out. Default = 100.
- **resultfile:** Relative or absolute file path to write out sensitivity results. Default = 'sensitivity\_analysis.csv'

**For example:**

```
python genODE \
  --modelfile=models/asm/glycolysis.modelspec \
  --prefix=sen01 \
  --mtype=ASM \
  --multiple=100 \
  --solver=RK4 \
  --timestep=1 \
  --endtime=21600 \
  --cleanup=True \
  --outfmt=reduced \
  --resultfile=sensitivity_analysis.csv
```

## S7. Operation: mergeASM

**Synopsis:** Read the AdvanceSyn model specification file(s) and merge them into a single AdvanceSyn model specification file.

**Usage:** `python astools.py mergeASM [option]`

where [option] can be

- **modelfile:** Relative path(s) to the model specification file(s), separated by semi-colon.
- **outputfile:** Relative path to the output ASM model file.
- **prefix:** Prefix for new reaction IDs. This prefix cannot be any existing prefixes in any of the model specifications to be merged. Default = 'exp'.

**For example:**

```
python astools.py mergeASM \
  --prefix=exp \
  --
modelfile=models/asm/glycolysis.modelspec;models/asm/ppp.model
spec \
  --outputfile=models/asm/glycolysis_ppp.modelspec
```

## S8. Operation: printASM

**Synopsis:** Read the AdvanceSyn model specification file and print out its details before processing into model objects.

**Usage:** `python astools.py printASM [option]`

where [option] can be

- **modelfile:** Relative path to the model specification file.

- readertype: Reader type for AdvanceSyn Model specification. Allowable types are 'basic' and 'extended'.

**For example:**

```
python astools.py printASM \
    --modelfile=models/asm/glycolysis.modelspec \
    --readertype=extended
```

## S9. Operation: readmodel

**Synopsis:** Read a model file and print out its details after processing into model objects.

**Usage:** `python astools.py readmodel [option]`

where [option] can be

- modelfile: Relative path to the model specification file.
- mtype: Type of model specification file. Allowable types are 'ASM' (AdvanceSyn Model Specification), 'MO' (AdvanceSyn Model Objects). Default = 'ASM'.

**For example:**

```
python astools.py readmodel \
    --mtype=ASM \
    --modelfile=models/asm/glycolysis.modelspec
```

## S10. Operation: readflux

**Synopsis:** Read the AdvanceSyn model objects file and print out fluxes (productions and usages) of model objects.

**Usage:** `python astools.py readflux [option]`

where [option] can be

- modelfile: Relative path to the model specification file.
- mtype: Type of model specification file. Allowable types are 'ASM' (AdvanceSyn Model Specification), 'MO' (AdvanceSyn Model Objects). Default = 'ASM'.

**For example:**

```
python astools.py readflux \
    --mtype=ASM \
    --modelfile=models/asm/glycolysis.modelspec
```

## S11. Operation: runODE

**Synopsis:** Run / execute the ODE model and write out the simulation results.

**Usage:** `python astools.py runODE [option]`

where [option] can be



- `odefile`: Python ODE script file (in `odescript` folder) to run / execute.
- `sampling`: Sampling frequency. If 100, means only every 100th simulation result will be written out. The first (start) and last (end) result will always be written out. Default = 100.
- `resultfile`: Relative or absolute file path to write out simulation results. Default = `'oderesult.csv'`

**For example:**

```
python astools.py runODE \
    --odefile=glycolysis.py \
    --sampling=500 \
    --resultfile=oderesult.csv
```

## S12. Operation: `senGen`

**Synopsis:** Generate a series of AdvanceSyn Model Specifications from an existing model by multiplying a multiple to the variable in preparation for sensitivity analyses.

**Usage:** `python astools.py senGen [option]`

where `[option]` can be

- `modelfile`: Name of model specification file in `models` folder. This assumes that the model file is not in `models` folder.
- `multiple`: Multiples to change each variable value. Default = 100 (which will multiple the original parameter value by 100).
- `prefix`: A prefixing string for the set of new model specification for identification purposes. Default = "".
- `mtype`: Type of model specification file. Allowable types are 'ASM' (AdvanceSyn Model Specification). Default = 'ASM'.

**For example:**

```
python astools.py senGen \
    --modelfile=models/asm/glycolysis.modelspec \
    --prefix=sen01 \
    --mtype=ASM \
    --multiple=100
```

## S13. Operation: `cameo-fba`

**Synopsis:** Simulate a model using Flux Balance Analysis (FBA), with Cameo.

**Usage:** `python astools.py cameo-fba [option]`

where `[option]` can be

- `model`: Model acceptable by Cameo (see <http://cameo.bio/02-import-models.html>).
- `result_type`: Type of result to give. Allowable types are `objective` (objective value from FBA) or `flux` (table of fluxes). Default value = `objective`.

**For example:**

```
python astools.py cameo-fba \
  --model=iJO1366 \
  --result_type=objective
```

## S14. Operation: cameo-medium-cpds

**Synopsis:** List the medium in a model, with Cameo.

**Usage:** `python astools.py cameo-medium-cpds [option]`

where [option] can be

- `model`: Model acceptable by Cameo (see <http://cameo.bio/02-import-models.html>).

**For example:**

```
python astools.py cameo-medium-cpds --model=iAF1260
```

## S15. Operation: cameo-medium-fba

**Synopsis:** Simulate a model after adding media change(s) using Flux Balance Analysis (pFBA), with Cameo.

**Usage:** `python astools.py cameo-medium-fba [option]`

where [option] can be

- `model`: Model acceptable by Cameo (see <http://cameo.bio/02-import-models.html>).
- `change`: String to define medium change(s). Each change is defined as <compound ID>:<new value>. For example, EX\_o2\_e,0 will represent anaerobic condition. Multiple changes are delimited using semicolon.
- `result_type`: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

**For example:**

```
python astools.py cameo-medium-fba \
  --model=iAF1260 \
  --change=EX_o2_e,0;EX_glc__D_e,5.0 \
  --result_type=objective
```

## S16. Operation: cameo-medium-pfba

**Synopsis:** Simulate a model after adding media change(s) using Parsimonious Flux Balance Analysis (pFBA), with Cameo.

pFBA reference: Lewis, N.E., Hixson, K.K., Conrad, T.M., Lerman, J.A., Charusanti, P., Polpitiya, A.D., Adkins, J.N., Schramm, G., Purvine, S.O., Lopez-Ferrer, D. and Weitz, K.K., 2010. Omic data from evolved E. coli are consistent with computed optimal growth from genome-scale models. *Molecular Systems Biology*, 6(1):390.

<http://www.ncbi.nlm.nih.gov/pubmed/20664636>

**Usage:** `python astools.py cameo-medium-pfba [option]`

where [option] can be

- model: Model acceptable by Cameo (see <http://cameo.bio/02-import-models.html>).
- change: String to define medium change(s). Each change is defined as <compound ID>:<new value>. For example, EX\_o2\_e,0 will represent anaerobic condition. Multiple changes are delimited using semicolon.
- result\_type: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

**For example:**

```
python astools.py cameo-medium-pfba \
  --model=iAF1260 \
  --change=EX_o2_e,0;EX_glc__D_e,5.0 \
  --result_type=objective
```

## S17. Operation: cameo-mutant-fba

**Synopsis:** Simulate a model after adding mutation(s) using Flux Balance Analysis (FBA), with Cameo.

**Usage:** python astools.py cameo-mutant-fba [option]

where [option] can be

- model: Model acceptable by Cameo (see <http://cameo.bio/02-import-models.html>).
- mutation: String to define mutation(s). Each mutation is defined as <reaction ID>:<upper bound>:<lower bound>. For example, RBFK,0,0 will represent a knock out. Multiple mutations are delimited using semicolon.
- result\_type: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

**For example:**

```
python astools.py cameo-mutant-fba \
  --model=iJO1366 \
  --mutation=NNAM,100,0;RBFK,0,0 \
  --result_type=objective
```

## S18. Operation: cameo-mutant-pfba

**Synopsis:** Simulate a model after adding mutation(s) using Parsimonious Flux Balance Analysis (pFBA), with Cameo.

pFBA reference: Lewis, N.E., Hixson, K.K., Conrad, T.M., Lerman, J.A., Charusanti, P., Polpitiya, A.D., Adkins, J.N., Schramm, G., Purvine, S.O., Lopez-Ferrer, D. and Weitz, K.K., 2010. Omic data from evolved E. coli are consistent with computed optimal growth from genome-scale models. *Molecular Systems Biology*, 6(1):390.

<http://www.ncbi.nlm.nih.gov/pubmed/20664636>

**Usage:** python astools.py cameo-mutant-pfba [option]

where [option] can be

- model: Model acceptable by Cameo (see <http://cameo.bio/02-import-models.html>).
- mutation: String to define mutation(s). Each mutation is defined as <reaction ID>:<upper bound>:<lower bound>. For example, RBFK,0,0 will represent a knock out. Multiple mutations are delimited using semicolon.
- result\_type: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

**For example:**

```
python astools.py cameo-mutant-pfba \  
  --model=iJO1366 \  
  --mutation=NNAM,100,0;RBFK,0,0 \  
  --result_type=objective
```

## S19. Operation: cameo-pfba

**Synopsis:** Simulate a model using Parsimonious Flux Balance Analysis (pFBA), with Cameo.

pFBA reference: Lewis, N.E., Hixson, K.K., Conrad, T.M., Lerman, J.A., Charusanti, P., Polpitiya, A.D., Adkins, J.N., Schramm, G., Purvine, S.O., Lopez-Ferrer, D. and Weitz, K.K., 2010. Omic data from evolved E. coli are consistent with computed optimal growth from genome-scale models. *Molecular Systems Biology*, 6(1):390.  
<http://www.ncbi.nlm.nih.gov/pubmed/20664636>

**Usage:** `python astools.py cameo-pfba [option]`

where [option] can be

- model: Model acceptable by Cameo (see <http://cameo.bio/02-import-models.html>).
- result\_type: Type of result to give. Allowable types are objective (objective value from FBA) or flux (table of fluxes). Default value = objective.

**For example:**

```
python astools.py cameo-pfba \  
  --model=iJO1366 \  
  --result_type=objective
```

## S20. Operation: cameo-rxn-cpds

**Synopsis:** List the reactants and products for each reaction in a model, with Cameo.

**Usage:** `python astools.py cameo-rxn-cpds [option]`

where [option] can be

- model: Model acceptable by Cameo (see <http://cameo.bio/02-import-models.html>).

**For example:**

```
python astools.py cameo-rxn-cpds --model=iJO1366
```

## S21. Operation: cameo-rxn-names

**Synopsis:** List the reaction names in a model, with Cameo.

**Usage:** `python astools.py cameo-rxn-names [option]`

where [option] can be

- `model`: Model acceptable by Cameo (see <http://cameo.bio/02-import-models.html>).

**For example:**

```
python astools.py cameo-rxn-names --model=iJO1366
```

## S22. Example: Glycolysis Model

Filename: `glycolysis_manuscript.modelspec`

[Specification]

type: 1

[Identifiers]

name: glycolysis

author: Maurice Ling

[Objects]

glucose: D-glucose

g6p: a-D-Glucose-6-phosphate

f6p: b-D-Fructose-6-phosphate

f16p: b-D-Fructose-1,6-phosphate

gadp: D-glyceraldehyde 3-phosphate

dhap: Dihydroxyacetone phosphate

bpg13: D-1,3-bisphosphoglycerate

pg3: 3-phosphoglycerate

pg2: 2-phosphoglycerate

pep: phosphoenolpyruvate

pyr: pyruvate

[Initials]

glucose: 1e-3

g6p: 1e-6

f6p: 1e-6

f16p: 1e-6

gadp: 1e-6

dhap: 1e-6

bpg13: 1e-6

pg3: 1e-6

pg2: 1e-6

pep: 1e-6

pyr: 1e-6

```

[Variables]
hk: 1e-6
pgi: 1e-6
pfk: 1e-6
aldo: 1e-6
tpi: 1e-6
gapdh: 1e-6
pkg: 1e-6
pgm: 1e-6
eno: 1e-6
pk: 1e-6

[Reactions]
r1: glucose -> g6p | ${Variables:hk} * glucose
r2: g6p -> f6p | ${Variables:pgi} * g6p
r3: f6p -> f16p | ${Variables:pfk} * f6p
r4: f16p -> gadp + dhap | ${Variables:aldo} * f16p
r5: dhap -> gadp | ${Variables:tpi} * dhap
r6: gadp -> bpg13 | ${Variables:gapdh} * gadp
r7: bpg13 -> pg3 | ${Variables:pkg} * bpg13
r8: pg3 -> pg2 | ${Variables:pgm} * pg3
r9: pg2 -> pep | ${Variables:eno} * pg2
r10: pep -> pyr | ${Variables:pk} * pep

```

## S23. Example: Pentose Phosphate Pathway Model

Filename: ppp\_manuscript.modelspec

```

[Specification]
type: 1

[Identifiers]
name: pentose phosphate
author: Maurice Ling

[Objects]
g6p: a-D-Glucose-6-phosphate
pgl6: 6-phosphogluconolactone
pg6: 6-phosphogluconate
r5p: ribulose-5-phosphate
ri5p: ribose-5-phosphate
x5p: xylulose-5-phosphate
gly3p: glyceraldehyde-4-phosphate
sedo7p: sedoheptulose-7-phosphate
f6p: b-D-Fructose-6-phosphate
ery4p: erythrose-4-phosphate
gadp: D-glyceraldehyde 3-phosphate

[Initials]
g6p: 1e-6
pgl6: 1e-6

```

```

pg6: 1e-6
r5p: 1e-6
ri5p: 1e-6
x5p: 1e-6
gly3p: 1e-6
sedo7p: 1e-6
f6p: 1e-6
ery4p: 1e-6
gadp: 1e-6

[Variables]
g6pd: 1e-6
gl: 1e-6
pgd6: 1e-6
r5pi: 1e-6
r5pe: 1e-6
th: 1e-6
ta: 1e-6

[Reactions]
r1: g6p -> pgl6 | ${Variables:g6pd} * g6p
r2: pgl6 -> pg6 | ${Variables:gl} * pgl6
r3: pg6 -> r5p | ${Variables:pgd6} * pg6
r4: r5p -> ri5p | ${Variables:r5pi} * r5p
r5: r5p -> x5p | ${Variables:r5pe} * r5p
r6: ri5p + x5p -> gly3p + sedo7p | ${Variables:th} * ri5p * x5p
r7: gly3p + sedo7p -> f6p + ery4p | ${Variables:ta} * gly3p *
sedo7p
r8: ery4p + x5p -> gadp + f6p | ${Variables:th} * ery4p * x5p

```

## S24. Example: Merged Glycolysis / Pentose Phosphate Pathway

Glycolysis model and pentose phosphate pathway model were merged using the following command to yield `glycolysis_ppp.modelspec` as the merged model file:

```

python astools.py mergeASM \
    --prefix=exp \
    --
modelfile=models/asm/glycolysis_manuscript.modelspec;models/as
m/ppp_manuscript.modelspec \
    --outputfile=models/asm/glycolysis_ppp.modelspec

```

Filename: `glycolysis_ppp.modelspec`

```

[Specification]
type = 1

```

```

[Identifiers]
name = glycolysis
author = Maurice Ling
name_1 = pentose phosphate

```

```
author_1 = Maurice Ling
```

```
[Objects]
```

```
glucose = D-glucose  
g6p = a-D-Glucose-6-phosphate  
f6p = b-D-Fructose-6-phosphate  
f16p = b-D-Fructose-1,6-phosphate  
gadp = D-glyceraldehyde 3-phosphate  
dhap = Dihydroxyacetone phosphate  
bpg13 = D-1,3-bisphosphoglycerate  
pg3 = 3-phosphoglycerate  
pg2 = 2-phosphoglycerate  
pep = phosphoenolpyruvate  
pyr = pyruvate  
pgl6 = 6-phosphogluconolactone  
pg6 = 6-phosphogluconate  
r5p = ribulose-5-phosphate  
ri5p = ribose-5-phosphate  
x5p = xylulose-5-phosphate  
gly3p = glyceraldehyde-4-phosphate  
sedo7p = sedoheptulose-7-phosphate  
ery4p = erythrose-4-phosphate
```

```
[Initials]
```

```
glucose = 1e-3  
g6p = 1e-6  
f6p = 1e-6  
f16p = 1e-6  
gadp = 1e-6  
dhap = 1e-6  
bpg13 = 1e-6  
pg3 = 1e-6  
pg2 = 1e-6  
pep = 1e-6  
pyr = 1e-6  
pgl6 = 1e-6  
pg6 = 1e-6  
r5p = 1e-6  
ri5p = 1e-6  
x5p = 1e-6  
gly3p = 1e-6  
sedo7p = 1e-6  
ery4p = 1e-6
```

```
[Variables]
```

```
hk = 1e-6  
pgi = 1e-6  
pfk = 1e-6  
aldo = 1e-6  
tpi = 1e-6  
gapdh = 1e-6
```



```

pkg = 1e-6
pgm = 1e-6
eno = 1e-6
pk = 1e-6
g6pd = 1e-6
gl = 1e-6
pgd6 = 1e-6
r5pi = 1e-6
r5pe = 1e-6
th = 1e-6
ta = 1e-6

[Reactions]
exp1 = glucose -> g6p | ${Variables:hk} * glucose
exp2 = g6p -> f6p | ${Variables:pgi} * g6p
exp3 = f6p -> f16p | ${Variables:pfk} * f6p
exp4 = f16p -> gadp + dhap | ${Variables:aldo} * f16p
exp5 = dhap -> gadp | ${Variables:tpi} * dhap
exp6 = gadp -> bpg13 | ${Variables:gapdh} * gadp
exp7 = bpg13 -> pg3 | ${Variables:pkg} * bpg13
exp8 = pg3 -> pg2 | ${Variables:pgm} * pg3
exp9 = pg2 -> pep | ${Variables:eno} * pg2
exp10 = pep -> pyr | ${Variables:pk} * pep
exp11 = g6p -> pgl6 | ${Variables:g6pd} * g6p
exp12 = pgl6 -> pg6 | ${Variables:gl} * pgl6
exp13 = pg6 -> r5p | ${Variables:pgd6} * pg6
exp14 = r5p -> ri5p | ${Variables:r5pi} * r5p
exp15 = r5p -> x5p | ${Variables:r5pe} * r5p
exp16 = ri5p + x5p -> gly3p + sedo7p | ${Variables:th} * ri5p *
x5p
exp17 = gly3p + sedo7p -> f6p + ery4p | ${Variables:ta} * gly3p
* sedo7p
exp18 = ery4p + x5p -> gadp + f6p | ${Variables:th} * ery4p *
x5p

```

## S25. Example: Glycolysis / Pentose Phosphate Pathway Simulation

### Script

The simulation script, `glycolysis_ppp.py`, is generated from the merged model specification, `glycolysis_ppp.modelspec`, using the following command:

```

python astools.py genODE \
    --modelfile=models/asm/glycolysis_ppp.modelspec \
    --mtype=ASM \
    --solver=RK4 \
    --timestep=1 \
    --endtime=21600 \
    --lowerbound=0;0 \
    --upperbound=1e-3;1e-3 \
    --odefile=glycolysis_ppp.py

```

Filename: glycolysis\_ppp.py

```
''' -----  
Python ODE script generated by ASModeller  
(a package in AdvanceSynToolKit)
```

Date Time: 2020-6-7 13:15:15:218630

```
name: glycolysis  
author: Maurice Ling  
name_1: pentose phosphate  
author_1: Maurice Ling  
----- '''
```

```
def glucose(t, y):  
    exp1 = 1e-6 * y[0]  
    return (0) - (exp1)  
  
def g6p(t, y):  
    exp1 = 1e-6 * y[0]  
    exp2 = 1e-6 * y[1]  
    exp11 = 1e-6 * y[1]  
    return (exp1) - (exp2 + exp11)  
  
def f6p(t, y):  
    exp2 = 1e-6 * y[1]  
    exp17 = 1e-6 * y[16] * y[17]  
    exp18 = 1e-6 * y[18] * y[15]  
    exp3 = 1e-6 * y[2]  
    return (exp2 + exp17 + exp18) - (exp3)  
  
def f16p(t, y):  
    exp3 = 1e-6 * y[2]  
    exp4 = 1e-6 * y[3]  
    return (exp3) - (exp4)  
  
def gadp(t, y):  
    exp4 = 1e-6 * y[3]  
    exp5 = 1e-6 * y[5]  
    exp18 = 1e-6 * y[18] * y[15]  
    exp6 = 1e-6 * y[4]  
    return (exp4 + exp5 + exp18) - (exp6)  
  
def dhap(t, y):  
    exp4 = 1e-6 * y[3]  
    exp5 = 1e-6 * y[5]  
    return (exp4) - (exp5)  
  
def bpg13(t, y):  
    exp6 = 1e-6 * y[4]  
    exp7 = 1e-6 * y[6]
```

```

        return (exp6) - (exp7)

def pg3(t, y):
    exp7 = 1e-6 * y[6]
    exp8 = 1e-6 * y[7]
    return (exp7) - (exp8)

def pg2(t, y):
    exp8 = 1e-6 * y[7]
    exp9 = 1e-6 * y[8]
    return (exp8) - (exp9)

def pep(t, y):
    exp9 = 1e-6 * y[8]
    exp10 = 1e-6 * y[9]
    return (exp9) - (exp10)

def pyr(t, y):
    exp10 = 1e-6 * y[9]
    return (exp10) - (0)

def pgl6(t, y):
    exp11 = 1e-6 * y[1]
    exp12 = 1e-6 * y[11]
    return (exp11) - (exp12)

def pg6(t, y):
    exp12 = 1e-6 * y[11]
    exp13 = 1e-6 * y[12]
    return (exp12) - (exp13)

def r5p(t, y):
    exp13 = 1e-6 * y[12]
    exp14 = 1e-6 * y[13]
    exp15 = 1e-6 * y[13]
    return (exp13) - (exp14 + exp15)

def ri5p(t, y):
    exp14 = 1e-6 * y[13]
    exp16 = 1e-6 * y[14] * y[15]
    return (exp14) - (exp16)

def x5p(t, y):
    exp15 = 1e-6 * y[13]
    exp16 = 1e-6 * y[14] * y[15]
    exp18 = 1e-6 * y[18] * y[15]
    return (exp15) - (exp16 + exp18)

def gly3p(t, y):
    exp16 = 1e-6 * y[14] * y[15]
    exp17 = 1e-6 * y[16] * y[17]

```

```

    return (exp16) - (exp17)

def sedo7p(t, y):
    exp16 = 1e-6 * y[14] * y[15]
    exp17 = 1e-6 * y[16] * y[17]
    return (exp16) - (exp17)

def ery4p(t, y):
    exp17 = 1e-6 * y[16] * y[17]
    exp18 = 1e-6 * y[18] * y[15]
    return (exp17) - (exp18)

def boundary_checker(y, boundary, type):
    """
    Private function - called by ODE solvers to perform boundary
    checking
    of variable values and reset them to specific values if the
    original
    values fall out of the boundary values.

    Boundary parameter takes the form of a dictionary with
    variable number
    as key and a list of [<boundary value>, <value to set if
    boundary is
    exceeded>]. For example, the following dictionary for lower
    boundary
    (type = 'lower') {'1': [0.0, 0.0], '5': [2.0, 2.0]} will set
    the lower
    boundary of variable y[0] and [5] to 0.0 and 2.0 respectively.
    This
    also allows for setting to a different value - for example,
    {'1': [0.0,
    1.0]} will set variable y[0] to 2.0 if the original y[0]
    value is
    negative.

    @param y: values for variables
    @type y: list
    @param boundary: set of values for boundary of variables
    @type boundary: dictionary
    @param type: the type of boundary to be checked, either
    'upper' (upper
    boundary) or 'lower' (lower boundary)
    """
    for k in list(boundary.keys()):
        if y[int(k)] < boundary[k][0] and type == 'lower':
            y[int(k)] = boundary[k][1]
        if y[int(k)] > boundary[k][0] and type == 'upper':
            y[int(k)] = boundary[k][1]
    return y

```

```
def RK4(funcs, x0, y0, step, xmax, nonODEfunc=None,
        lower_bound=None, upper_bound=None,
        overflow=1e100, zerodivision=1e100):
    '''
    Generator to integrate a system of ODEs,  $y' = f(x, y)$ , using
    fourth
    order Runge-Kutta method.

    A function (as nonODEfunc parameter) can be included to
    modify one or
    more variables (y0 list). This function will not be an ODE
    (not a
     $dy/dt$ ). This can be used to consolidate the modification of
    one or
    more variables at each ODE solving step. For example,  $y[0]$ 
    =  $y[1] / y[2]$ 
    can be written as

    >>> def modifying_function(y, step):
            y[0] = y[1] / y[2]
            return y

    This function must take 'y' (variable list) and 'step' (time
    step) as
    parameters and must return 'y' (the modified variable list).
    This
    function will execute before boundary checking at each time
    step.

    Upper and lower boundaries of one or more variable can be
    set using
    upper_bound and lower_bound parameters respectively. These
    parameters
    takes the form of a dictionary with variable number as key
    and a list
    of [<boundary value>, <value to set if boundary is
    exceeded>]. For
    example, the following dictionary for lower boundary {'1':
    [0.0, 0.0],
    '5': [2.0, 2.0]} will set the lower boundary of variable
    y[0] and y[5]
    to 0.0 and 2.0 respectively. This also allows for setting
    to a different
    value - for example, {'1': [0.0, 1.0]} will set variable
    y[0] to 2.0 if
    the original y[0] value is negative.

    @param funcs: system of differential equations
    @type funcs: list
    @param x0: initial value of x-axis, which is usually
    starting time
```

```

    @type x0: float
    @param y0: initial values for variables
    @type y0: list
    @param step: step size on the x-axis (also known as step in
calculus)
    @type step: float
    @param xmax: maximum value of x-axis, which is usually
ending time
    @type xmax: float
    @param nonODEfunc: a function to modify the variable list
(y0)
    @type nonODEfunc: function
    @param lower_bound: set of values for lower boundary of
variables
    @type lower_bound: dictionary
    @param upper_bound: set of values for upper boundary of
variables
    @type upper_bound: dictionary
    @param overflow: value (usually a large value) to assign in
event of
    over flow error (usually caused by a large number) during
integration.
    Default = 1e100.
    @type overflow: float
    @param zerodivision: value (usually a large value) to assign
in event
    of zero division error, which results in positive infinity,
during
    integration. Default = 1e100.
    @type zerodivision: float
    ...
yield [x0] + y0
def solver(funcs, x0, y0, step):
    n = len(funcs)
    f1, f2, f3, f4 = [0]*n, [0]*n, [0]*n, [0]*n
    y1 = [0]*n
    for i in range(n):
        try: f1[i] = funcs[i](x0, y0)
        except TypeError: pass
        except ZeroDivisionError: f1[i] = zerodivision
        except OverflowError: f1[i] = overflow
    for j in range(n):
        y1[j] = y0[j] + (0.5*step*f1[j])
    for i in range(n):
        try: f2[i] = funcs[i]((x0+(0.5*step)), y1)
        except TypeError: pass
        except ZeroDivisionError: f2[i] = zerodivision
        except OverflowError: f2[i] = overflow
    for j in range(n):
        y1[j] = y0[j] + (0.5*step*f2[j])
    for i in range(n):

```

```

        try: f3[i] = funcs[i]((x0+(0.5*step)), y1)
        except TypeError: pass
        except ZeroDivisionError: f3[i] = zerodivision
        except OverflowError: f3[i] = overflow
    for j in range(n):
        y1[j] = y0[j] + (step*f3[j])
    for i in range(n):
        try: f4[i] = funcs[i]((x0+step), y1)
        except TypeError: pass
        except ZeroDivisionError: f4[i] = zerodivision
        except OverflowError: f4[i] = overflow
    for i in range(n):
        try: y1[i] = y0[i] + (step * \
            (f1[i] + (2.0*f2[i]) + (2.0*f3[i]) + f4[i])
/ 6.0)

        except TypeError: pass
        except ZeroDivisionError: y1[i] = zerodivision
        except OverflowError: y1[i] = overflow
    return y1
while x0 < xmax:
    y1 = solver(funcs, x0, y0, step)
    if nonODEfunc:
        y1 = nonODEfunc(y1, step)
    if lower_bound:
        y1 = boundary_checker(y1, lower_bound, 'lower')
    if upper_bound:
        y1 = boundary_checker(y1, upper_bound, 'upper')
    y0 = y1
    x0 = x0 + step
    yield [x0] + y0

ODE = list(range(19))
ODE[0] = glucose
ODE[1] = g6p
ODE[2] = f6p
ODE[3] = f16p
ODE[4] = gadp
ODE[5] = dhap
ODE[6] = bpg13
ODE[7] = pg3
ODE[8] = pg2
ODE[9] = pep
ODE[10] = pyr
ODE[11] = pg16
ODE[12] = pg6
ODE[13] = r5p
ODE[14] = ri5p
ODE[15] = x5p
ODE[16] = gly3p
ODE[17] = sedo7p
ODE[18] = ery4p

```

```

y = list(range(19))
y[0] = 1e-3      # glucose : D-glucose
y[1] = 1e-6      # g6p : a-D-Glucose-6-phosphate
y[2] = 1e-6      # f6p : b-D-Fructose-6-phosphate
y[3] = 1e-6      # f16p : b-D-Fructose-1,6-phosphate
y[4] = 1e-6      # gadp : D-glyceraldehyde 3-phosphate
y[5] = 1e-6      # dhap : Dihydroxyacetone phosphate
y[6] = 1e-6      # bpg13 : D-1,3-bisphosphoglycerate
y[7] = 1e-6      # pg3 : 3-phosphoglycerate
y[8] = 1e-6      # pg2 : 2-phosphoglycerate
y[9] = 1e-6      # pep : phosphoenolpyruvate
y[10] = 1e-6     # pyr : pyruvate
y[11] = 1e-6     # pgl6 : 6-phosphogluconolactone
y[12] = 1e-6     # pg6 : 6-phosphogluconate
y[13] = 1e-6     # r5p : ribulose-5-phosphate
y[14] = 1e-6     # ri5p : ribose-5-phosphate
y[15] = 1e-6     # x5p : xylulose-5-phosphate
y[16] = 1e-6     # gly3p : glyceraldehyde-4-phosphate
y[17] = 1e-6     # sedo7p : sedoheptulose-7-phosphate
y[18] = 1e-6     # ery4p : erythrose-4-phosphate

labels = ['time', 'glucose', 'g6p', 'f6p', 'f16p', 'gadp',
'dhap', 'bpg13', 'pg3', 'pg2', 'pep', 'pyr', 'pgl6', 'pg6',
'r5p', 'ri5p', 'x5p', 'gly3p', 'sedo7p', 'ery4p']

lowerbound = {'0': [0.0, 0.0], '1': [0.0, 0.0], '2': [0.0, 0.0],
'3': [0.0, 0.0], '4': [0.0, 0.0], '5': [0.0, 0.0], '6': [0.0,
0.0], '7': [0.0, 0.0], '8': [0.0, 0.0], '9': [0.0, 0.0], '10':
[0.0, 0.0], '11': [0.0, 0.0], '12': [0.0, 0.0], '13': [0.0, 0.0],
'14': [0.0, 0.0], '15': [0.0, 0.0], '16': [0.0, 0.0], '17': [0.0,
0.0], '18': [0.0, 0.0], }

upperbound = {'0': [0.001, 0.001], '1': [0.001, 0.001], '2':
[0.001, 0.001], '3': [0.001, 0.001], '4': [0.001, 0.001], '5':
[0.001, 0.001], '6': [0.001, 0.001], '7': [0.001, 0.001], '8':
[0.001, 0.001], '9': [0.001, 0.001], '10': [0.001, 0.001], '11':
[0.001, 0.001], '12': [0.001, 0.001], '13': [0.001, 0.001], '14':
[0.001, 0.001], '15': [0.001, 0.001], '16': [0.001, 0.001], '17':
[0.001, 0.001], '18': [0.001, 0.001], }

model = RK4(ODE, 0.0, y, 1, 21600, None, lowerbound, upperbound)

```

## S26. Example: Execution of Simulation Script

The above generated simulation script, `glycolysis_ppp.py`, is executed using the following command and the results file is `glycolysis_ppp_simulation.csv`:

```

python astools.py runODE \
    --odefile=glycolysis_ppp.py \
    --sampling=500 \

```



```
--resultfile= glycolysis_ppp_simulation.csv
```

## S27. Example: Running Sensitivity Analysis

Sensitivity analysis is executed to evaluate the effects of varying enzyme concentrations on the distribution of metabolites after 21600 seconds (6 hours) by varying each enzyme concentration one at a time. The parameters to vary for sensitivity analysis will be parameters listed in the Variables section of the model. The command is

```
python astools.py LSA \  
  --modelfile=models/asm/glycolysis_ppp.modelspec \  
  --prefix=sen01 \  
  --mtype=ASM \  
  --multiple=100 \  
  --solver=RK4 \  
  --timestep=1 \  
  --endtime=21600 \  
  --cleanup=True \  
  --outfmt=reduced \  
  --resultfile= glycolysis_ppp_sensitivity_analysis.csv
```

## S28. Example: Simplified Full Model from Yeoh et al. (2020)

Filename: yeoh\_manuscript.modelspec

[Specification]

type: 1

[Identifiers]

name: simplified full model

author: Maurice Ling

[Objects]

glycerol: glycerol

oxygen: oxygen

OD: optical density

FA: ferulic acid

FAcoA: feruyl-co-A

Vanillin: vanillin

[Initials]

glycerol: 0.5

oxygen: 0.05

OD: 0.005

FA: 0.5

FAcoA: 0.0

Vanillin: 0.0

[Variables]

FCS: 0.05

ECH: 0.05

```
[Reactions]
r1: glycerol + oxygen -> OD | 0.16666 * (glycerol / (glycerol
+ 4)) * (oxygen / (oxygen + 0.005)) * (OD / 5)
r2: oxygen -> | 2.0979e-3 - (0.333 * oxygen)
r3: FA -> FAcoA | 32.42806 * OD * ${Variables:FCS} * (FA / (FA
+ 0.2913))
r4: FAcoA -> Vanillin | 156.924 * OD * ${Variables:ECH} *
(FAcoA / (FAcoA + 0.4698))
```

## S29. Example: Simplified Cell Growth Model from Yeoh et al. (2020)

Filename: yeoh\_cell\_growth\_manuscript.modelspec

```
[Specification]
```

```
type: 1
```

```
[Identifiers]
```

```
name: simplified cell growth model
```

```
author: Maurice Ling
```

```
[Objects]
```

```
glycerol: glycerol
```

```
oxygen: oxygen
```

```
OD: optical density
```

```
[Initials]
```

```
glycerol: 0.5
```

```
oxygen: 0.05
```

```
OD: 0.005
```

```
[Variables]
```

```
[Reactions]
```

```
r1: glycerol + oxygen -> OD | 0.16666 * (glycerol / (glycerol
+ 4)) * (oxygen / (oxygen + 0.005)) * (OD / 5)
```

```
r2: oxygen -> | 2.0979e-3 - (0.333 * oxygen)
```

## S30. Example: Simplified Vanillin Production Model from Yeoh et al. (2020)

Filename: yeoh\_vanillin\_production\_manuscript.modelspec

```
[Specification]
```

```
type: 1
```

```
[Identifiers]
```

```
name: simplified vanillin production model
```

```
author: Maurice Ling
```

```

[Objects]
FA: ferulic acid
FAcoA: feruyl-co-A
Vanillin: vanillin

[Initials]
FA: 0.5
FAcoA: 0.0
Vanillin: 0.0

[Variables]
FCS: 0.05
ECH: 0.05

[Reactions]
r1: FA -> FAcoA | 32.42806 * OD * ${Variables:FCS} * (FA / (FA
+ 0.2913))
r2: FAcoA -> Vanillin | 156.924 * OD * ${Variables:ECH} *
(FAcoA / (FAcoA + 0.4698))

```

### S31. Example: Simplified Merged Model from Yeoh et al. (2020)

Filename: yeoh\_merged\_manuscript.modelspec

```

[Specification]
type = 1

[Identifiers]
name = simplified vanillin production model
author = Maurice Ling
name_1 = simplified cell growth model
author_1 = Maurice Ling

[Objects]
FA = ferulic acid
FAcoA = feruyl-co-A
Vanillin = vanillin
glycerol = glycerol
oxygen = oxygen
OD = optical density

[Initials]
FA = 0.5
FAcoA = 0.0
Vanillin = 0.0
glycerol = 0.5
oxygen = 0.05
OD = 0.005

```

```

[Variables]
FCS = 0.05
ECH = 0.05

[Reactions]
exp1 = FA -> FAcoA | 32.42806 * OD * ${Variables:FCS} * (FA /
(FA + 0.2913))
exp2 = FAcoA -> Vanillin | 156.924 * OD * ${Variables:ECH} *
(FAcoA / (FAcoA + 0.4698))
exp3 = glycerol + oxygen -> OD | 0.16666 * (glycerol /
(glycerol + 4)) * (oxygen / (oxygen + 0.005)) * (OD / 5)
exp4 = oxygen -> | 2.0979e-3 - (0.333 * oxygen)

```