Easy Specification and Solution of Ordinary Differential Equations on Graphs

Reinhard Hemmerling

University of Göttingen

28 February 2012

Outline

Introduction

- FSPM – Functional Structural Plant Model
- Structure described by L-systems
  $\rightarrow$ parallel string rewriting
- Function described by **differential equations**
  $\rightarrow$ mostly ODEs, but also DDEs, PDEs
- Examples: photosynthesis, partitioning, respiration

Motivation

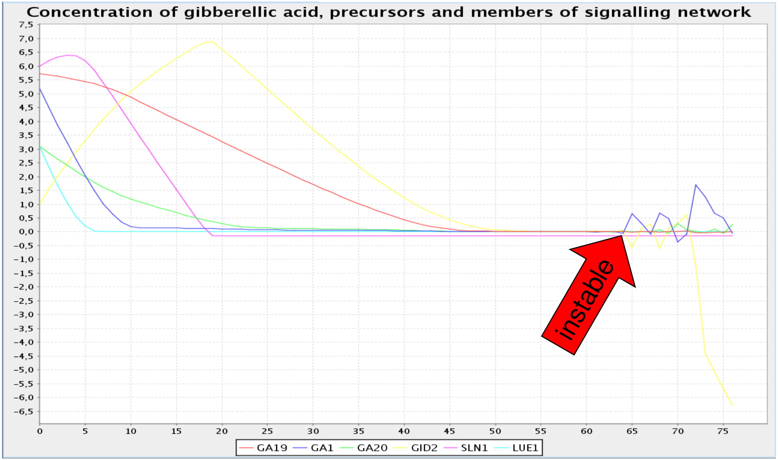
- We want to specify **ordinary differential equations** (ODEs)

$$\frac{dy}{dt} = f(t, y)$$

  on a graph structure, and **solve** them **numerically**
- But ODEs are **continuous**, while rule application is **discrete**
- Often found to be implemented by **forward Euler integration** as part of the rewriting step

Euler integration is bad

(Example from [BSHK$^+$08])



A-stability

Definition (Dahlquist[Dah63])

A method is called A-stable, if all solutions tend to zero, as $n \to \infty$, when the method is applied with fixed positive h to any differential equation of the form,

$$dy/dt = \lambda y,$$

where λ is a complex constant with negative real part.

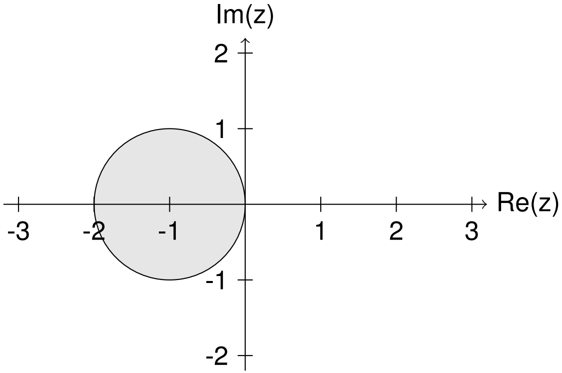
Stability of forward Euler

$$\boxed{y_{n+1} = y_n + h \cdot f(t_n, y_n)}$$

$$y_{n+1} = y_n + h\lambda y_n$$
$$y_n = (1 + h\lambda)^n y_0$$

$\Rightarrow$ stable if $|1 + h\lambda| < 1$

Region of *absolute stability* for forward Euler ($z = h\lambda$)



Stability of backward Euler

$$\boxed{y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1})}$$

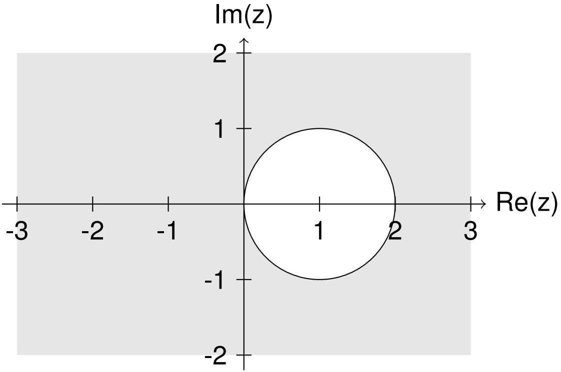
$$y_{n+1} = y_n + h\lambda y_{n+1}$$

$$y_n = \left(\frac{1}{1-h\lambda}\right)^n y_0$$

$\Rightarrow$ stable if $|1-h\lambda| > 1$

$\Rightarrow$ **unconditionally stable**, but implicit equation must be solved

Region of *absolute stability* for backward Euler ($z = h\lambda$)



Stiffness

Example of Damped Spring [Cas03]

$$u'' + (\lambda + 1)u' + \lambda u = 0$$

Compare with

$$u' = -u$$

Converted to set of 1st order ODEs

$$u' = z, z' = -(\lambda + 1)z - \lambda u, \lambda > 0$$

Exact solution

$$u(t) = Ae^{-t}$$

Exact solution

$$u(t) = Ae^{-t} + Be^{-\lambda t}$$

Required step size

$$0 < h < 2$$

Required step size

$0 < h < 2$ and $0 < h < \frac{2}{\lambda}$

What if integration was limited to just $[0, 1/\lambda]$?

Stiffness

Example of Damped Spring [Cas03]

$$u'' + (\lambda + 1)u' + \lambda u = 0$$

Converted to set of 1st order ODEs

$$u' = z, z' = -(\lambda + 1)z - \lambda u, \lambda > 0$$

Exact solution

$$u(t) = Ae^{-t} + Be^{-\lambda t}$$

Required step size

$$0 < h < 2 \text{ and } 0 < h < \frac{2}{\lambda}$$

Compare with

$$u' = -u$$

Exact solution

$$u(t) = Ae^{-t}$$

Required step size

$$0 < h < 2$$

What if integration was limited to just $[0, 1/\lambda]$?

Stiffness

Example of Damped Spring [Cas03]

$$u'' + (\lambda + 1)u' + \lambda u = 0$$

Converted to set of 1st order ODEs

$$u' = z, z' = -(\lambda + 1)z - \lambda u, \lambda > 0$$

Exact solution

$$u(t) = Ae^{-t} + Be^{-\lambda t}$$

Required step size

$0 < h < 2$ and $0 < h < \frac{2}{\lambda}$

Compare with

$$u' = -u$$

Exact solution

$$u(t) = Ae^{-t}$$

Required step size

$$0 < h < 2$$

What if integration was limited to just $[0, 1/\lambda]$?

Stiffness

Example of Damped Spring [Cas03]

$$u'' + (\lambda + 1)u' + \lambda u = 0$$

Converted to set of 1st order ODEs

$$u' = z, z' = -(\lambda + 1)z - \lambda u, \lambda > 0$$

Exact solution

$$u(t) = Ae^{-t} + Be^{-\lambda t}$$

Required step size

$0 < h < 2$ and $0 < h < \frac{2}{\lambda}$

Compare with

$$u' = -u$$

Exact solution

$$u(t) = Ae^{-t}$$

Required step size

$$0 < h < 2$$

What if integration was limited to just $[0, 1/\lambda]$?

Stiffness

Example of Damped Spring [Cas03]

$$u'' + (\lambda + 1)u' + \lambda u = 0$$

Converted to set of 1st order ODEs

$$u' = z, z' = -(\lambda + 1)z - \lambda u, \lambda > 0$$

Exact solution

$$u(t) = Ae^{-t} + Be^{-\lambda t}$$

Required step size

$0 < h < 2$ and $0 < h < \frac{2}{\lambda}$

Compare with

$$u' = -u$$

Exact solution

$$u(t) = Ae^{-t}$$

Required step size

$$0 < h < 2$$

What if integration was limited to just $[0, 1/\lambda]$?

Stiffness

Example of Damped Spring [Cas03]

$$u'' + (\lambda + 1)u' + \lambda u = 0$$

Converted to set of 1st order ODEs

$$u' = z, z' = -(\lambda + 1)z - \lambda u, \lambda > 0$$

Exact solution

$$u(t) = Ae^{-t} + Be^{-\lambda t}$$

Required step size

$0 < h < 2$ and $0 < h < \frac{2}{\lambda}$

Compare with

$$u' = -u$$

Exact solution

$$u(t) = Ae^{-t}$$

Required step size

$$0 < h < 2$$

What if integration was limited to just $[0, 1/\lambda]$?

Stiffness

Example of Damped Spring [Cas03]

$$u'' + (\lambda + 1)u' + \lambda u = 0$$

Converted to set of 1st order ODEs

$$u' = z, z' = -(\lambda + 1)z - \lambda u, \lambda > 0$$

Exact solution

$$u(t) = Ae^{-t} + Be^{-\lambda t}$$

Required step size

$0 < h < 2$ and $0 < h < \frac{2}{\lambda}$

Compare with

$$u' = -u$$

Exact solution

$$u(t) = Ae^{-t}$$

Required step size

$$0 < h < 2$$

What if integration was limited to just $[0, 1/\lambda]$?

Stiffness

Example of Damped Spring [Cas03]

$$u'' + (\lambda + 1)u' + \lambda u = 0$$

Converted to set of 1st order ODEs

$$u' = z, z' = -(\lambda + 1)z - \lambda u, \lambda > 0$$

Exact solution

$$u(t) = Ae^{-t} + Be^{-\lambda t}$$

Required step size

$$0 < h < 2 \text{ and } 0 < h < \frac{2}{\lambda}$$

Compare with

$$u' = -u$$

Exact solution

$$u(t) = Ae^{-t}$$

Required step size

$$0 < h < 2$$

What if integration was limited to just $[0, 1/\lambda]$?

To Euler or not to Euler...

Conclusion

Must use other method than Euler, alternatives:

- Runge-Kutta methods
- Multistep methods
- Extrapolation methods

Other interesting features

- Error estimation and variable step size
- Interpolation
- Handling discontinuities in f
- Switching functions

To Euler or not to Euler...

Conclusion

Must use other method than Euler, alternatives:

- Runge-Kutta methods
- Multistep methods
- Extrapolation methods

Other interesting features

- Error estimation and variable step size
- Interpolation
- Handling discontinuities in f
- Switching functions

But . . .

- Implementation can be tricky
- Better resort to some existing library
- Still to convince the biologist
  $\rightarrow$ must be easy to use
  $\rightarrow$ make it 'look' like before
- Handle dynamic/growing structures

Outline

Example: Transport in a tree

Differential equations in the model

Diffusion between nodes A and B

$$-\frac{d[A_c]}{dt} = \frac{d[B_c]}{dt} = D \cdot ([A_c] - [B_c])$$

Production, Consumption, Growth

$$\frac{d[A_c]}{dt} = P_A - C_A \cdot [A_c]$$

$$\frac{d[A_l]}{dt} = \gamma \cdot C_A \cdot [A_c]$$

Example: Transport in a tree

Differential equations in the model

Diffusion between nodes A and B

$$-\frac{d[A_c]}{dt} = \frac{d[B_c]}{dt} = D \cdot ([A_c] - [B_c])$$

Production, Consumption, Growth

$$\frac{d[A_c]}{dt} = P_A - C_A \cdot [A_c]$$
$$\frac{d[A_l]}{dt} = \gamma \cdot C_A \cdot [A_c]$$

Example: Transport in a tree

XL code using Euler integration

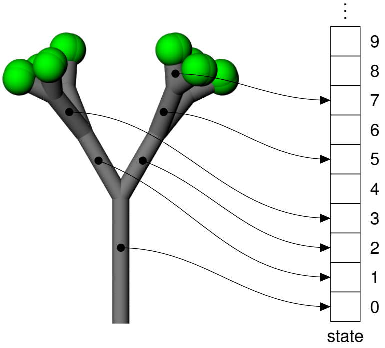
```
// apply production to nodes
a:A ::> a[carbon] :+= h * PROD;

// apply consumption and convert to growth
b:B ::> {
  double rate = CONS * b[carbon];
  b[carbon] :-= h * rate;
  b[length] :+= h * γ * rate;
}

// perform diffusion between nodes
ca:C (-->)+ : (cb:C) ::> {
  double rate = D * (ca[carbon] - cb[carbon]);
  ca[carbon] :-= h * rate;
  cb[carbon] :+= h * rate;
}
```

Using advanced methods

- Must express as initial value problem (IVP):

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0,$$

- Need to determine mapping between node attributes and elements of state vector:



Using advanced methods

Steps that must be performed

- In main function:
  1) Calculate length of state vector
  2) Allocate state vector
  3) Create mapping between attributes of nodes and entries in state vector
  4) Copy state from graph to state vector
  5) Perform integration
  6) Copy state from state vector to graph
- In rate function:
  1) Initialize rate vector with zero
  2) Copy state from state vector to graph
  3) Calculate rates using XL rules and accumulate them in the rate vector

Using advanced methods

XL code of main function

```
// calculate length of state vector
final int N = 2 * (int) count((* C *));

// allocate state vector
final double[] y0 = new double[N];
final double[] y  = new double[N];

// create mapping between attributes of nodes
// and entries in state vector
int index = 0;
[ c:C ::> { c[index] = index; index += 2; } ]

// copy state from graph to y0
[ c:C ::> y0[c[index]+0] = c[carbon]; ]
[ c:C ::> y0[c[index]+1] = c[length]; ]

// integrate over time
integrate(time, y0, time + DT, y);
time += DT;

// copy state from y to graph
[ c:C ::> c[carbon] = y[c[index]+0]; ]
[ c:C ::> c[length] = y[c[index]+1]; ]
```

Using advanced methods

XL code of rate function

```
void getRate(double[] rate, double time, double[] state) [
  // zero output array
  { java.util.Arrays.fill(rate, 0); }

  // copy state y to graph
  c:C ::> c[carbon] = y[c[index]+0];
  c:C ::> c[length] = y[c[index]+1];

  // apply production to A nodes
  c:A ::> rate[c[index]] += PROD;
  // apply consumption and convert to growth
  c:B ::> {
    double r = CONS * c[carbon];
    rate[c[index]+0] -= r;
    rate[c[index]+1] += γ * r;
  }
  // perform diffusion between nodes
  ca:C (-->)+ : (cb:C) ::> {
    double r = D * (ca[carbon] - cb[carbon]);
    rate[ca[index]] -= r;
    rate[cb[index]] += r;
  }
]
```

Conclusion

- This basically works, but **hard to use**
- Especially if attributes are distributed over a class hierarchy
- **prone to errors**
- Must **automatize creation of mapping** between graph and state vector
- Idea: introduction of **rate assignment operator** :'= to mark attributes for numerical integration

Rate assignment operator

XL code of rate function

```
// rate function, state is provided implicitly
void getRate() [
  // apply production to nodes
  a:A ::> a[carbon] :'= PROD;

  // apply consumption and convert to growth
  b:B ::> {
    double rate = CONS * b[carbon];
    b[carbon] :'= -rate;
    b[len   ] :'= γ * rate;
  }

  // perform diffusion between nodes
  ca:C (-->)+ : (cb:C) ::> {
    double rate = D * (ca[carbon] - cb[carbon]);
    ca[carbon] :'= -rate;
    cb[carbon] :'= +rate;
  }
]
```

Monitor Functions

- Needed to **communicate with the integrator** during integration
- For instance, to plot state data in regular intervals
- . . . or to **stop integration** when a condition becomes fulfilled
- . . . **to perform structural changes** of the graph
- Monitor function $g : \mathbb{R}^n \rightarrow \mathbb{R}$ maps state to (a single) value
- Root finding used to **determine exact event time**

Monitor Functions

XL code for branching

```
// install monitor on every leaf
a:A ::> monitor(
  // monitor function g
  void=>double a[carbon] - T,
  // event handler
  new Runnable() {
    public void run() [
      // split into branches with leaf on top
      a ==>
        [RU( 30) RH(75) B(0) A(a[carbon]/2)]
        [RU(-30) RH(75) B(0) A(a[carbon]/2)]
      ;
    ]
  }
);
```

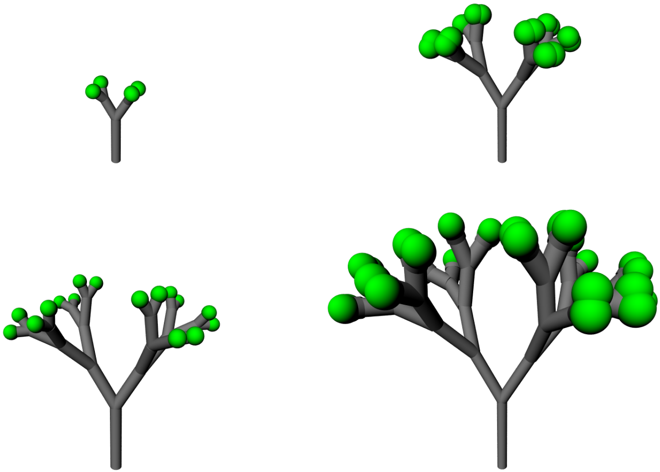
Tolerance specification

- Annotate property to request absolute/relative tolerance
- Might be needed in some cases
- Using this information is up to the integrator
- Example:

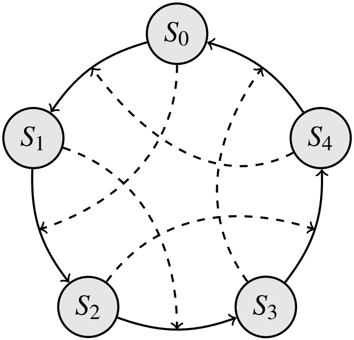
```
@Tolerance(absolute=1e-6, relative=1e-4)
double t;

module A(@Tolerance(absolute=1e-4) double n);
```

Outline

Example: Transport in a tree



Circular transport with inhibition



$$\frac{d[S_{i\oplus 2}]}{dt} = -\frac{d[S_{i\oplus 1}]}{dt} = \begin{cases} \mu \cdot [S_{i\oplus 1}] & \text{if } [S_i] \leq T \\ 0 & \text{otherwise.} \end{cases}$$

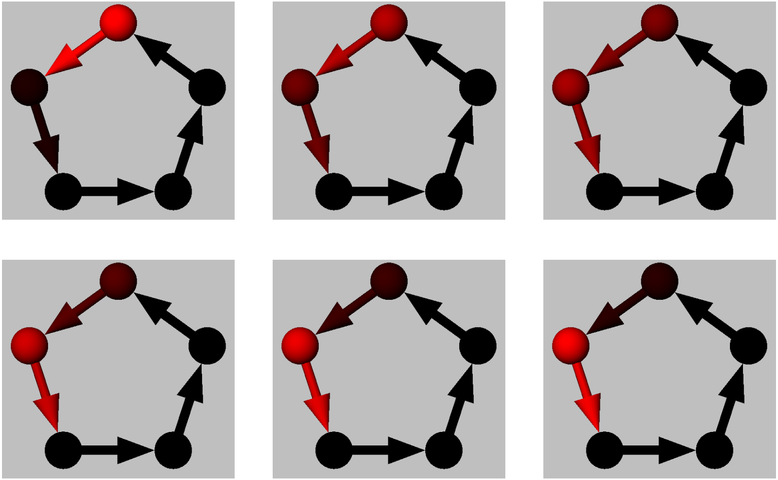
Circular transport with inhibition

XL code to model this

```
// apply ODE to each triple (x,y,z)
x:S -EDGE_0-> y:S -EDGE_0-> z:S ::> {
  double rate = x[c] > T ? 0 : μ * y[c];
  y[c] :'= -rate;
  z[c] :'= +rate;
}
```

Circular transport with inhibition

Resulting transport



dL-systems in XL

dL-system of *Anabaena catenula*

initial string: $F_h(x_{max}, c_{max})\ F_v(x_{max}, c_{max})\ F_h(x_{max}, c_{max})$

$F(x_l, c_l) < F_v(x, c) > F(x_r, c_r):$

  if $x < x_{max}\ \&\ c > c_{min}$

    solve $\frac{dx}{dt} = rx, \frac{dc}{dt} = D \cdot (c_l + c_r - 2c) - \mu c$

  if $x = x_{max}\ \&\ c > c_{min}$

    produce $F_v(kx_{max}, c)\ F_v((1-k)x_{max}, c)$

  if $c = c_{min}$

    produce $F_h(x, c)$

$F_h(x, c):$

  solve $\frac{dx}{dt} = r_x(x_{max} - x), \frac{dc}{dt} = r_c(c_{max} - c)$

dL-systems in XL

XL code for *Anabaena catenula* — rate function

```
// vegetative cells
F(xl, cl) fv:FV(x, c) F(xr, cr) ::> {
  fv[x] :'= R * x;
  fv[c] :'= D * (cl - 2*c + cr) - MU * c;
}

// heterocystic cells
fh:FH(x, c) ::> {
  fh[x] :'= R_X * (X_MAX - x);
  fh[c] :'= R_C * (C_MAX - c);
}
```

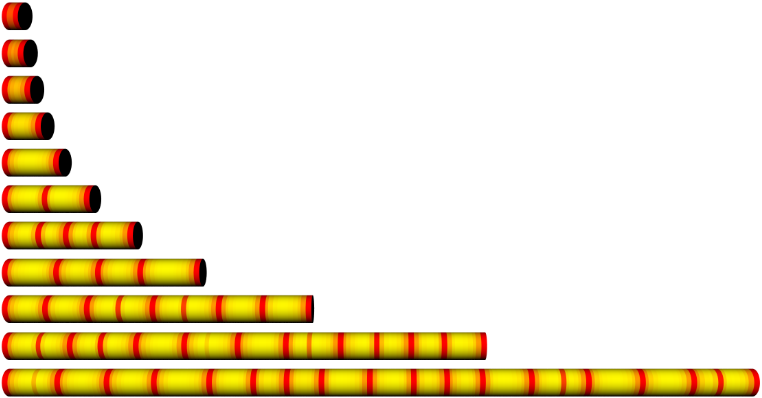
dL-systems in XL

XL code for *Anabaena catenula* — monitor functions

```
// set monitor for reaching maximum length
fv:FV ::> monitor(void=>double fv[x] - X_MAX,
  new Runnable() {
    public void run() [
      fv ==> FV(K*X_MAX, fv[c]) FV((1-K)*X_MAX, fv[c]);
    ]
  });

// set monitor for reaching minimum concentration
fv:FV ::> monitor(void=>double fv[c] - C_MIN,
  new Runnable() {
    public void run() [
      fv ==> FH(fv[x], fv[c]);
    ]
  });
```

dL-systems in XL

Simulation results



Outline

Summary

- **Combination** between **discrete** (graph rewriting rules) and **continuous** (ODE) processes
- User does not have to reimplement numerical integrators
- Numerical integration method can be **exchanged easily**
- Enhanced **accuracy** and **stability**
- **Separation** between **integration** of ODEs and **structural changes** in the graph
- Used in [HSK10, HEK10, EvdK10, SBSHK10]

Thank you for your attention.

Bibliography I

[BSHK+08] Gerhard Buck-Sorlin, Reinhard Hemmerling, Ole Kniemeyer, Benno Burema, and Winfried Kurth.
A rule-based model of barley morphogenesis, with special respect to shading and gibberellic acid signal transduction.
Annals of Botany, 101:1109–1123, 2008.

[Cas03] J. R. Cash.
Efficient numerical methods for the solution of stiff initial-value problems and differential algebraic equations.
Proceedings of the Royal Society: Mathematical, Physical and Engineering Sciences, 459(2032):797–815, 2003.

[Dah63] Germund G. Dahlquist.
A special stability problem for linear multistep methods.
BIT Numerical Mathematics, 3(1):27–43, 1963.

[EvdK10] J. B. Evers and A. R. van der Krol.
Hormonal regulation of branching modulated by light quality.
In T. DeJong and D. Da Silva, editors, *Proceedings of the 6th International Workshop on Functional-Structural Plant Models*, pages 149–151, Davis, 2010. University of California.

Bibliography II

[HEK10] R. Hemmerling, J. B. Evers, and W. Kurth.
Easy specification of differential equations describing biological processes, exemplified for plant hormone dynamics.
In T. DeJong and D. Da Silva, editors, *Proceedings of the 6th International Workshop on Functional-Structural Plant Models*, pages 158–160, Davis, 2010. University of California.

[HSK10] Reinhard Hemmerling, Katarína Smoleňová, and Winfried Kurth.
A programming language tailored to the specification and solution of differential equations describing processes on networks.
In Adrian-Horia Dediu, Henning Fernau, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications*, volume 6031 of *Lecture Notes in Computer Science*, pages 297–308. Springer Berlin / Heidelberg, 2010.

[SBSHK10] K. Smoleňová, G. Buck-Sorlin, R. Hemmerling, and W. Kurth.
Extension of a functional-structural model of barley for modelling of carbon and nitrogen partitioning.
In T. DeJong and D. Da Silva, editors, *Proceedings of the 6th International Workshop on Functional-Structural Plant Models*, pages 27–29, Davis, 2010. University of California.