Univariate Taylor Polynomial Arithmetic Applied to Matrix Factorizations in the Forward and Reverse Mode

EuroAD 2010
Paderborn, 03.06.2010

Sebastian F. Walter[3], Lutz Lehmann[4]

Montag, 30. April 2010

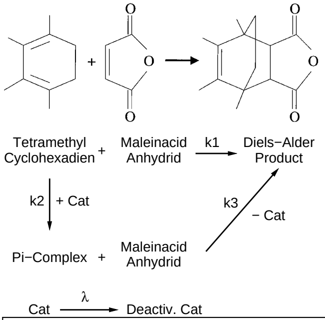
[3]sebastian.walter@gmail.com

[4]llehmann@mathematik.hu-berlin.de

PART I:

Motivation for Forward/Reverse Univariate Taylor Polynomial Arithmetic : **Optimum Experimental Design**

Optimum Experimental Design in Chemical Engineering



- non-catalyzed and catalyzed reaction path
- deactivation of the catalyst
- batch process
- measurements: product mass concentration
- control of educt molar numbers, catalyst concentration, temperature profile
- five unknown model parameters

$$\dot{n}_1 = -k \cdot \frac{n_1 \cdot n_2}{m_{tot}}, \qquad n_1(0) = n_{a1}$$

$$\dot{n}_2 = -k \cdot \frac{n_1 \cdot n_2}{m_{tot}}, \qquad n_2(0) = n_{a2}$$

$$\dot{n}_3 = k \cdot \frac{n_1 \cdot n_2}{m_{tot}}, \qquad n_3(0) = 0$$

$$k = k_1 \cdot \exp\left(-\frac{E_1}{R} \cdot \left(\frac{1}{T} - \frac{1}{T_{ref}}\right)\right) + k_{kat} \cdot c_{kat} \cdot \exp(-\lambda \cdot t) \cdot \exp\left(-\frac{E_{kat}}{R} \cdot \left(\frac{1}{T} - \frac{1}{T_{ref}}\right)\right)$$

$$n_4 = n_{a4} \qquad T = \vartheta + 273$$

$$m_{tot} = n_1 \cdot M_1 + n_2 \cdot M_2 + n_3 \cdot M_3 + n_4 \cdot M_4$$

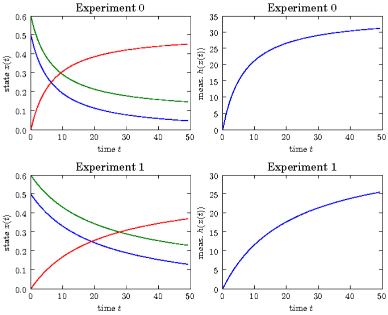
Optimum Experimental Design in Chemical Engineering (Cont.)

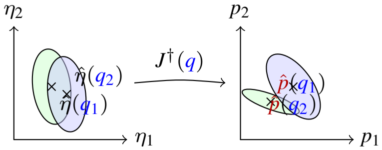
- **Dynamics**: Defined by ODE
- **Goal**: Estimate parameters $p = (k_1, k_{\text{kat}}, E_{\text{kat}}, \lambda, E_1)$
- **Problem**: Errors in the measurements η result in errors in parameters p.
- **nonlinear regression** with additive iid normal errors

$$\eta_m = h_m(t_m, x(t_m), p, q) + \varepsilon_m \ , \quad m = 1, \dots, N_M$$

$$\varepsilon_m \sim \mathcal{N}(0, \sigma_m^2)$$

- η_m are measurements, h **measurement model function** (connects model to the real world)
- Controls $q = (n_{a1}, n_{a2}, n_{a4}, c_{kat}, \theta)$ influence the error propagation.
- Therefore: Find controls q such that the "uncertainty" in p is as "small" as possible.





Simplified Derivation of an Uncertainty Measure

- **Unconstrained Nonlinear Parameter Estimation:**
$$\hat{p} \quad = \quad \mathrm{argmin}_p \|F(p)\|_2^2 \,,$$
where $F(p) = \Sigma^{-1}(\eta - h)$
measurements η, measurement function $h \in \mathbb{R}^{N_M}$, $\Sigma \in \mathbb{R}^{N_M \times N_M}$,

- **Solution Operator**:
$J^\dagger : F \mapsto p$ of linearized parameter estimation
$$J^\dagger \quad = \quad \left(J(\hat{p})^T J(\hat{p})\right)^{-1} J(\hat{p})^T$$
$$J(p) \quad = \quad \frac{\mathrm{d}F}{\mathrm{d}p}(p)$$

- **Linear Error Propagation**:
(computation of the covariance matrix C):
$$C \quad := \quad \mathbb{E}[(\hat{p} - \mathbb{E}[p])(\hat{p} - \mathbb{E}[p])^T] = J^\dagger \underbrace{\mathbb{E}[(\hat{F} - \mathbb{E}[F])(\hat{F} - \mathbb{E}[F]))^T]}_{=I}(J^\dagger)^T$$
$$= \quad (J^T J)^{-1}$$
(independent of $\hat{\eta}$)

Simplified Derivation of a Uncertainty Measure (cont.)

- Statistical Interpretation of the Covariance Matrix C:
  Defines **Confidence Region CR**:

$$\mathrm{CR} := \left\{ p : (p - \hat{p})^T C^{-1} (p - \hat{p}) \leq N_p \hat{\sigma}^2 F(N_p, N_M - N_p, 1 - \alpha) \right\}$$

where α is statistical significance level, F the F-distribution, $\hat{\sigma}$ unbiased estimate of the std

- **Typical Choices of Obj. Function:**

$$\begin{aligned}
\Phi_A(q) &= \frac{1}{N_P}\mathrm{tr}(C) = \frac{1}{N_P}\mathrm{tr}(J^T J)^{-1} && \text{A-criterion}\\
\Phi_D(q) &= \det(K^T C K)^{\frac{1}{N_P}} && \text{D-criterion}\\
\Phi_E(q) &= \max\{\lambda : \lambda \text{ eigenvalue of } C\} && \text{E-criterion}\\
\Phi_M(q) &= \max\{\sqrt{C_{ii}}, i = 1, \dots, N_P\} && \text{M-criterion}
\end{aligned}$$

- K is a projection s.t. $K^T C K$ is regular

Overall Objective Function

- **Part I: Computation of J_1 and J_2**

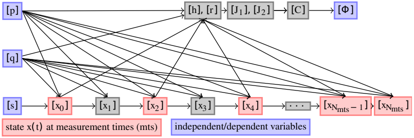
$$
\begin{aligned}
J_1[n_{\mathrm{mts}}, :] &= \frac{\sqrt{w_{\mathrm{mts}}}}{\sigma_{n_{\mathrm{mts}}}(x(t_{n_{\mathrm{mts}}}; s, u(t_{n_{\mathrm{mts}}}; q), q)} \frac{\mathrm{d}}{\mathrm{d}(p, s)} \left(h(t_{n_{\mathrm{mts}}}, x(t_{n_{\mathrm{mts}}}; s, u(t_{n_{\mathrm{mts}}}; q), p))\right) \\
J_2 &= \frac{\mathrm{d}}{\mathrm{d}(p, s)} r(q, p, s)
\end{aligned}
$$

- **Part II: Numerical Linear Algebra**

$$
\begin{aligned}
C(J_1, J_2) &= (\mathrm{I}, 0) \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathrm{I} \\ 0 \end{pmatrix} \\
&= \left(Q_2^T (Q_2 J_1^T J_1 Q_2^T)^{-1} Q_2 \right) \\
\Phi &= \lambda_1(C) \quad \text{, max. eigenvalue}
\end{aligned}
$$

where $J_2^T = (Q_1^T, Q_2^T)(L, 0)^T$

- **Computational Graph**



N_{mts} Number measurment times, σ std of a measurement, q controls, p nature given parameter, s pseudo-Parameter (e.g. initial values), u control functions

Experimental Design Optimization: Required Derivatives

- Gradient-type optimizers require the gradient $\nabla_q \Phi(q)$
- thus: second order derivatives (mixed partial derivatives in parameters p and control vector q)
- parameter robust OED:

$$\Phi_{\text{robust}}(q) := \phi(C(p,q)) + \gamma \left\| \frac{\mathrm{d}}{\mathrm{d}p} \phi(C(p,q)) \right\|_{2,\Sigma}$$

  i.e. requires third order derivatives (twice in parameters p and once in control vector q).
- Other objective functions may require even four'th and higher derivatives
- Matrices have often very high condition numbers (e.g. J)
- Number of controls N_q is much larger than number of parameters N_p $\Rightarrow$ **reverse mode** of AD
- Want **efficient**, **easy to use**, **flexible**, **numerically robust** methods for **forward/reverse mode AD**

PART II: Theory, Algorithms and Software

Reminder: Taylor Arithmetic

- Forward mode AD can be done by **Univariate Taylor Polynomial** (**UTP**) arithmetic:

$$
\begin{aligned}
f : \mathbb{R}^N &\rightarrow \mathbb{R}^M \\
\frac{\mathrm{d}f}{\mathrm{d}x}(x_0)x_1 &= \frac{\mathrm{d}}{\mathrm{d}t} f(x_0 + x_1 t)\Big|_{t=0} , \quad x_1 \in R^N
\end{aligned}
$$

- f is a composite function of *elementary functions* $\phi_l \in \{+, -, *, /, \sin, \dots \}$, i.e. $f = \phi_L \circ \phi_{L-1} \circ \dots \phi_1$.
- it suffices to provide Taylor arithmetic implementations for all elementary functions $\{+, -, *, /, \sin, \dots \}$.
- the UTP algorithms are also used in the **reverse mode of AD**

Algorithms for Univariate Taylor Polynomials over Scalars (UTPS)

- binary operations

$z = \phi(x,y)$	$d = 0, \ldots, D$	OPS	MOVES
$x + cy$	$z_d = x_d + cy_d$	$2D$	$3D$
$x \times y$	$z_d = \sum_{k=0}^{d} x_k y_{d-k}$	D^2	$3D$
x/y	$z_d = \frac{1}{y_0}\left[x_d - \sum_{k=0}^{d-1} z_k y_{d-k}\right]$	D^2	$3D$

- unary operations

$y = \phi(x)$	$d = 0, \ldots, D$	OPS	MOVES
$\ln(x)$	$\tilde{y}_d = \frac{1}{x_0}\left[\tilde{x}_d - \sum_{k=1}^{d-1} x_{d-k}\tilde{y}_k\right]$	D^2	$2D$
$\exp(x)$	$\tilde{y}_d = \sum_{k=1}^{d} y_{d-k}\tilde{x}_k$	D^2	$2D$
$\sqrt{x}$	$y_d = \frac{1}{2y_0}\left[x_d - \sum_{k=1}^{d-1} y_k y_{d-k}\right]$	$\frac{1}{2}D^2$	$3D$
x^r	$\tilde{y}_d = \frac{1}{x_0}\left[r\sum_{k=1}^{d} y_{d-k}\tilde{x}_k - \sum_{k=1}^{d-1} x_{d-k}\tilde{y}_k\right]$	$2D^2$	$2D$
$\sin(v)$ $\cos(v)$	$\tilde{s}_d = \sum_{j=1}^{d} \tilde{v}_j c_{d-j}$ $\tilde{c}_d = \sum_{j=1}^{d} -\tilde{v}_j s_{d-j}$	$2D^2$	$3D$
$\tan(v)$	$\tilde{\phi}_d = \sum_{j=1}^{d} w_{d-j}\tilde{v}_j$ $\tilde{w}_d = 2\sum_{j=1}^{d} \phi_{d-j}\tilde{\phi}_j$		
$\arcsin(v)$	$\tilde{\phi}_d = w_0^{-1}\left(\tilde{v}_d - \sum_{j=1}^{d-1} w_{d-j}\tilde{\phi}_j\right)$ $\tilde{w}_d = -\sum_{j=1}^{d} v_{d-j}\tilde{\phi}_j$		
$\arctan(v)$	$\tilde{\phi}_d = w_0^{-1}\left(\tilde{v}_d - \sum_{j=1}^{d-1} w_{d-j}\tilde{\phi}_j\right)$ $\tilde{w}_d = 2\sum_{j=1}^{d} v_{d-j}\tilde{v}_j$		

Apply UTP to Numerical Linear Algebra (NLA) Algorithms

- **Possibility 1**: Apply standard AD techniques to the NLA algorithms
  - will non-differentiable operations cause problems? (e.g. pivoting or treatment of degenerate cases)
  - how treat factorizations that are **not unique** in nominal solution (e.g eigenvalue decomposition with repeated eigenvalues). Possibly higher-order information makes it unique. That means that e.g. for $[y]_D = f([x]_D)$ it happens that $y_0 = y_0(x_0, x_1, x_2, \dots)$ and **not** $y_0 = y_0(x_0)$ as usually assumed.
  - **memory consumption**: NLA algorithms often have $\mathcal{O}(N^3)$ complexity, therefore also $\mathcal{O}(N^3)$ memory requirement? Always possible to reduce to $\mathcal{O}(N^2)$?
  - source trafo software featuring UTP?
  - operator overloading software for UTP exists (ADOL-C, CppAD) but is relatively slow and needs retaping for program branches (pivoting...)
  - code reuse of existing algorithms?
  - performance: how hard to parallelize? Optimized implementations a la ATLAS? NLA is going to stay. But what about new coding paradigms?
- **Possibility 2**: Matrix Calculus Approach, topic of this talk

Newton's Method

- Many functions are implicitly defined by algebraic equations:
  - multiplicative inverse: $y = x^{-1}$ by $0 = xy - 1$
  - in general for independent x and dependent y:

$$0 \quad = \quad F(x, y)$$

- **Newton's Method**[29]: Let $F([x], [y]_D) \stackrel{D}{=} 0$ and $F'([x], [y]_D) \mod t^D$ invertible. Then

$$\begin{aligned} 0 &\stackrel{D+E}{=} F([x], [y]_{D+E}) \\ 0 &\stackrel{D+E}{=} F([x], [y]_D) + F'([x], [y]_D)[\Delta y]_E t^D \\ [\Delta y]_E &\stackrel{E}{=} -\left(F'([x], [y]_E)\right)^{-1} [\Delta F]_E \end{aligned}$$

- $[X]_D \equiv [X_0, \ldots, x_{D-1}] \equiv \sum_{d=0}^{D-1} x_d t^d, \quad [\Delta F]_E t^D \stackrel{D+E}{=} F([x], [y]_D)$
- if $E = D$ then number of correct coefficients is doubled

[29] also called Newton-Hensel lifting or Hensel lifting

Univariate Taylor Polynomial Arithmetic on Matrices (UTPM)

- Application of Newton's Method to defining equations
- **Defining equations** of the QR decomposition:

$$\begin{aligned} 0 &\stackrel{D}{=} [Q]_D[R]_D - [A]_D \\ 0 &\stackrel{D}{=} [Q]_D^T[Q]_D - \mathrm{I} \\ 0 &\stackrel{D}{=} P_L \circ [R]_D \ , \end{aligned}$$

  where $(P_L)_{ij} = \delta_{i>j}$ and element-wise multiplication $\circ$.
- **Defining equations** of the symmetric eigenvalue decomposition

$$\begin{aligned} 0 &\stackrel{D}{=} [Q]_D^T[A]_D[Q]_D - [\Lambda]_D \\ 0 &\stackrel{D}{=} [Q]_D^T[Q]_D - \mathrm{I} \\ 0 &\stackrel{D}{=} (P_L + P_R) \circ [\Lambda]_D \ . \end{aligned}$$

- **Defining equations** of the Cholesky Decomposition

$$\begin{aligned} 0 &\stackrel{D}{=} [L]_D[L]_D^T - [a]_D \\ 0 &\stackrel{D}{=} P_D \circ [L]_D - \mathrm{I} \\ 0 &\stackrel{D}{=} P_R \circ [L]_D \ . \end{aligned}$$

- etc...

Algorithm: Forward UTPM of the Rectangular QR Decomposition

input : $[A]_D = [A_0, \ldots, A_{D-1}]$, where $A_d \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D-1$, $M \geq N$.
output: $[Q]_D = [Q_0, \ldots, Q_{D-1}]$ matrix with orthonormal column vectors, where $Q_d \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D-1$
output: $[R]_D = [R_0, \ldots, R_{D-1}]$ upper triangular, where $R_d \in \mathbb{R}^{N \times N}$, $d = 0, \ldots, D-1$

$Q_0, R_0 = \mathrm{qr}\,(A_0)$

for $d = 1$ **to** $D-1$ **do**

$\Delta F = A_d - \sum_{k=1}^{d-1} Q_{d-k} R_k$
$S = -\frac{1}{2} \sum_{k=1}^{d-1} Q_{d-k}^T Q_k$
$P_L \circ X = P_L \circ (Q_0^T \Delta F R_0^{-1} - S)$
$X = P_L \circ X - (P_L \circ X)^T$
$R_d = Q_0^T \Delta F - (S + X) R_0$
$Q_d = (\Delta F - Q_0 R_d) R_0^{-1}$

end

Algorithm: Reverse UTPM of the Rectangular QR Decomposition

input : $[A]_D = [A_0, \dots, A_{D-1}]$, where $A_d \in \mathbb{R}^{M\times N}$, $d = 0, \dots, D-1$, $M \geq N$.
input : $[Q]_D = [Q_0, \dots, Q_{D-1}]$ matrix with orthonormal column vectors, where $Q_d \in \mathbb{R}^{M\times N}$, $d = 0, \dots, D-1$
input : $[R]_D = [R_0, \dots, R_{D-1}]$ upper triangular, where $R_d \in \mathbb{R}^{N\times N}$, $d = 0, \dots, D-1$
input/output: $[\bar{A}]_D = [\bar{A}_0, \dots, \bar{A}_{D-1}]$, where $\bar{A}_d \in \mathbb{R}^{M\times N}$, $d = 0, \dots, D-1$, $M \geq N$.
input : $[\bar{Q}]_D = [\bar{Q}_0, \dots, \bar{Q}_{D-1}]$, where $\bar{Q}_d \in \mathbb{R}^{M\times N}$, $d = 0, \dots, D-1$
input : $[\bar{R}]_D = [\bar{R}_0, \dots, \bar{R}_{D-1}]$, where $\bar{R}_d \in \mathbb{R}^{N\times N}$, $d = 0, \dots, D-1$

$$
\begin{aligned}
[\bar{A}]_D &= [\bar{A}]_D + ([\bar{Q}]_D - [Q]_D[Q]_D^T[\bar{Q}]_D)[R]_D^{-T}) \\
&\quad + [Q]_D \left([\bar{R}]_D + P_L \circ \left([R]_D[\bar{R}]_D^T - [\bar{R}]_D[R]_D^T + [Q]_D^T[\bar{Q}]_D - [\bar{Q}]_D^T[Q]_D \right) [R]_D^{-T} \right)
\end{aligned}
$$

ALGOPY Live Example: *QR* decomposition

```
import numpy; from algopy import UTPM

# QR decomposition, UTPM forward
D,P,M,N = 3,1,5,2
A = UTPM(numpy.random.rand(D,P,M,N))
Q,R = UTPM.qr(A)
B = UTPM.dot(Q,R)

# check that the results are correct
print 'Q.T Q - 1\n',UTPM.dot(Q.T,Q) - numpy.eye(N)
print 'QR - A\n',B - A
print 'triu(R) - R\n', UTPM.triu(R) - R

# QR decomposition, UTPM reverse
Bbar = UTPM(numpy.random.rand(D,P,M,N))
Qbar,Rbar = UTPM.pb_dot(Bbar, Q, R, B)
Abar = UTPM.pb_qr(Qbar, Rbar, A, Q, R)

print 'Abar - Bbar\n',Abar - Bbar
```

ALGOPY Live Example: Moore-Penrose Pseudo inverse

```
import numpy; from algopy import CGraph, Function, UTPM, dot, qr, eigh, in
D,P,M,N = 2,1,5,2
# generate badly conditioned matrix A
A = UTPM(numpy.zeros((D,P,M,N)))
x = UTPM(numpy.zeros((D,P,M,1))); y = UTPM(numpy.zeros((D,P,M,1)))
x.data[0,0,:,0] = [1,1,1,1,1]; x.data[1,0,:,0] = [1,1,1,1,1]
y.data[0,0,:,0] = [1,2,1,2,1]; y.data[1,0,:,0] = [1,2,1,2,1]
alpha = 10**-5; A = dot(x,x.T) + alpha*dot(y,y.T); A = A[:,:2]
# Method 1: Naive approach
Apinv = dot(inv(dot(A.T,A)),A.T)
print 'naive approach: A Apinv A - A = 0 \n', dot(dot(A, Apinv),A) - A
print 'naive approach: Apinv A Apinv - Apinv = 0 \n', dot(dot(Apinv, A),Ap
print 'naive approach: (Apinv A)^T - Apinv A = 0 \n', dot(Apinv, A).T - d
print 'naive approach: (A Apinv)^T - A Apinv = 0 \n', dot(A, Apinv).T - d
# Method 2: Using the differentiated QR decomposition
Q,R = qr(A)
tmp1 = solve(R.T, A.T)
tmp2 = solve(R, tmp1)
Apinv = tmp2
print 'QR approach: A Apinv A - A = 0 \n', dot(dot(A, Apinv),A) - A
print 'QR approach: Apinv A Apinv - Apinv = 0 \n', dot(dot(Apinv, A),Apinv
print 'QR approach: (Apinv A)^T - Apinv A = 0 \n', dot(Apinv, A).T - dot(
print 'QR approach: (A Apinv)^T - A Apinv = 0 \n', dot(A, Apinv).T - dot(
```

Algorithm: Forward UTPM of Symmetric Eigenvalue Decomposition

input : $[A]_D = [A_0, \dots, A_{D-1}]$, where $A_d \in \mathbb{R}^{N \times N}$ symmetric positive definite, $d = 0, \dots, D-1$
output: $[\tilde{\Lambda}]_D = [\tilde{\Lambda}_0, \dots, \tilde{\Lambda}_{D-1}]$, where $\Lambda_0 \in \mathbb{R}^{N \times N}$ diagonal and $\Lambda_d \in \mathbb{R}^{N \times N}$ block diagonal $d = 1, \dots, D-1$.
output: $b \in \mathbb{N}^{N_b+1}$, array of integers defining the blocks. The integer N_B is the number of blocks. Each block has the size of the multiplicity of an eigenvalue λ_{n_b} of Λ_0 s.t. for sl $= b[n_b] : b[n_b+1]$ one has $(Q_0[:, \mathrm{sl}\,])^T A_0 Q_0[:, \mathrm{sl}\,] = \lambda_{n_b} \mathrm{I}$.

$\Lambda_0, Q_0 = \mathrm{eigh}\,(A_0)$

$E_{ij} = (\Lambda_0)_{jj} - (\Lambda_0)_{ii}$
$H = P_B \circ (1/E)$

for $d = 1$ **to** $D-1$ **do**

$S = -\frac{1}{2} \sum_{k=1}^{d-1} Q_{d-k}^T Q_k$
$K = \Delta F + \tilde{Q}_0^T A_d \tilde{Q}_0 + S\Lambda_0 + \Lambda_0 S$
$\tilde{Q}_d = Q_0(S + H \circ K)$
$\tilde{\Lambda}_d = \bar{P}_B \circ K$

end

- for the special case of distinct eigenvalues, this algorithm suffices
- for repeated eigenvalues this algorithm is one step in a little more involved algorithm

Test Example for the Symmetric Eigenvalue Decomposition[44]

- Orthonormal Matrix:

$$
\begin{aligned}
Q(t) &= \frac{1}{\sqrt{3}} \begin{pmatrix} \cos(x(t)) & 1 & \sin(x(t)) & -1 \\ -\sin(x(t)) & -1 & \cos(x(t)) & -1 \\ 1 & -\sin(x(t)) & 1 & \cos(x(t)) \\ -1 & \cos(x(t)) & 1 & \sin(x(t)) \end{pmatrix} \\
\Lambda(t) &= \operatorname{diag}(x^2 - x + \frac{1}{2}, 4x^2 - 3x, \delta(-\frac{1}{2}x^3 + 2x^2 - \frac{3}{2}x + 1) + (x^3 + x^2 - 1), 3x - 1) \ ,
\end{aligned}
$$

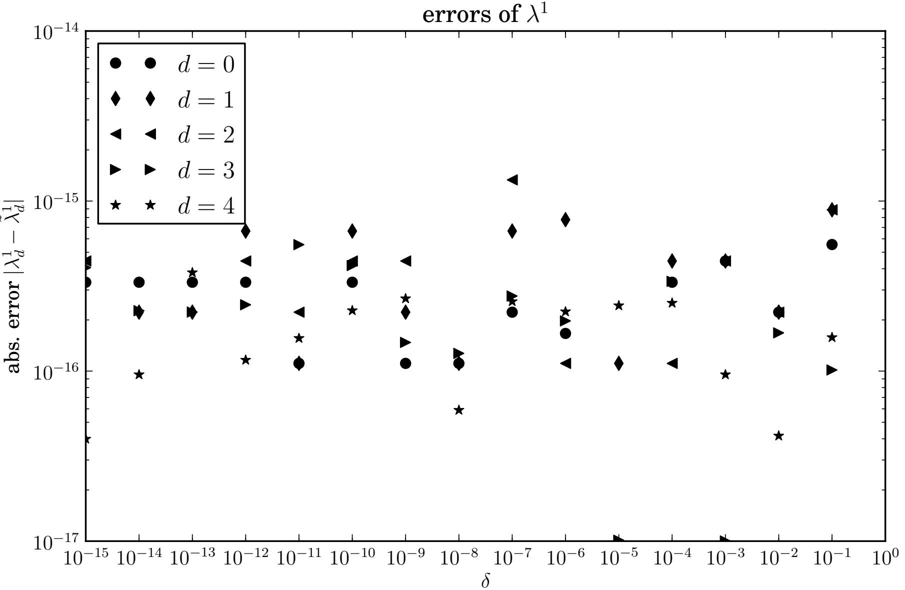
  where $x \equiv x(t) := 1 + t$.
- constant $\delta = 0$ means **repeated eigenvalues**, $\delta > 0$ distinct but close
- In Taylor arithmetic one obtains

$$
\begin{aligned}
\Lambda_0 &= \operatorname{diag}(1/2, 1, 1 + \delta, 2) \\
\Lambda_1 &= \operatorname{diag}(1, 5, 5 + \delta, 3) \\
\Lambda_2 &= \operatorname{diag}(2, 8, 8 + \delta, 0) \\
\Lambda_3 &= \operatorname{diag}(0, 0, 6 - 3\delta, 0) \\
\Lambda_d &= \operatorname{diag}(0, 0, 0, 0), \quad \forall d \geq 4 \ .
\end{aligned}
$$

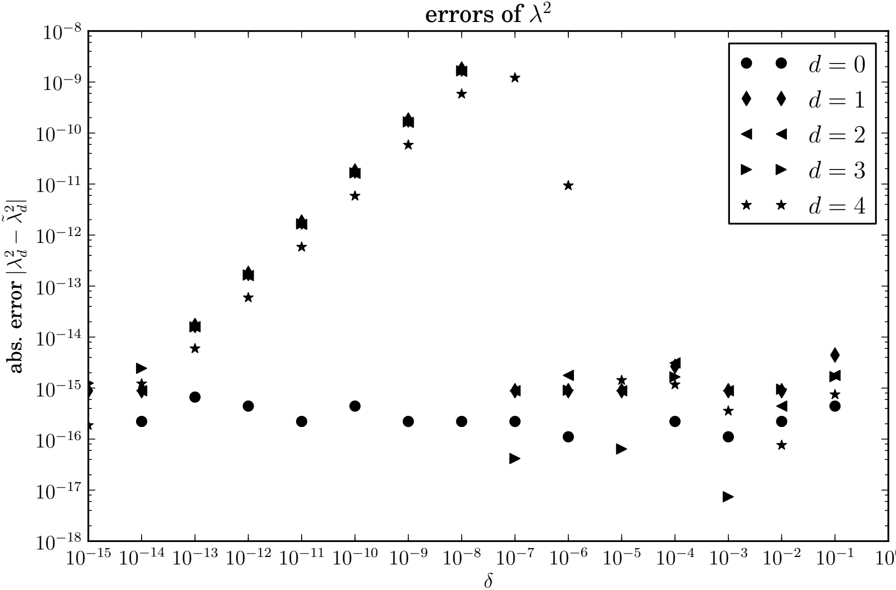
- Define $A(t) = Q(t)\Lambda(t)Q(t)$ and try to reconstruct $\Lambda(t)$ and $Q(t)$.

[44] Example adapted from Andrew and Tan, Computation of Derivatives of Repeated Eigenvalues and the Corresponding Eigenvectors of Symmetric Matrix Pencils, SIAM Journal on Matrix Analysis and Applications

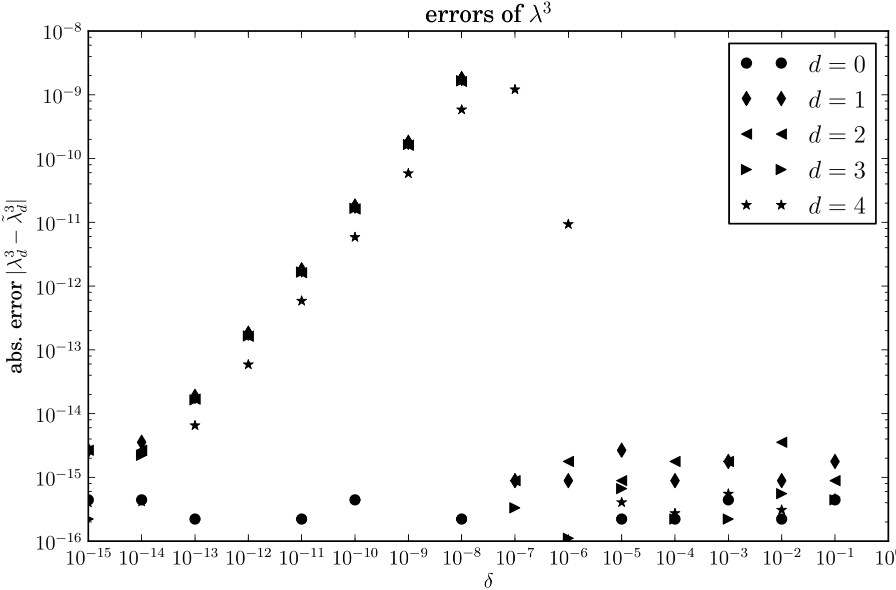
Test Example for the Symmetric Eigenvalue Decomposition (cont.)



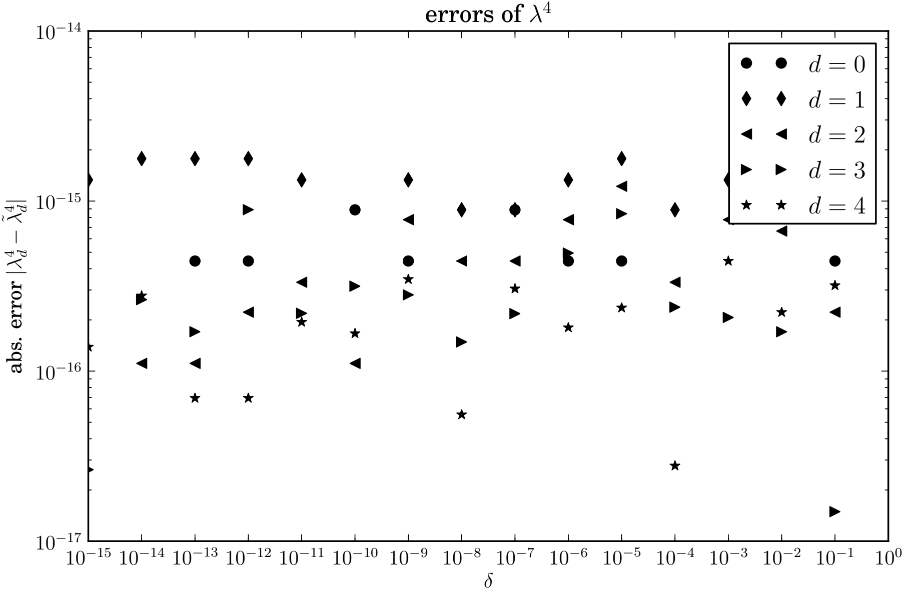
Test Example for the Symmetric Eigenvalue Decomposition (cont.)



Test Example for the Symmetric Eigenvalue Decomposition (cont.)



Test Example for the Symmetric Eigenvalue Decomposition (cont.)



The E-Criterion of the Opt. Exp. Design Problem

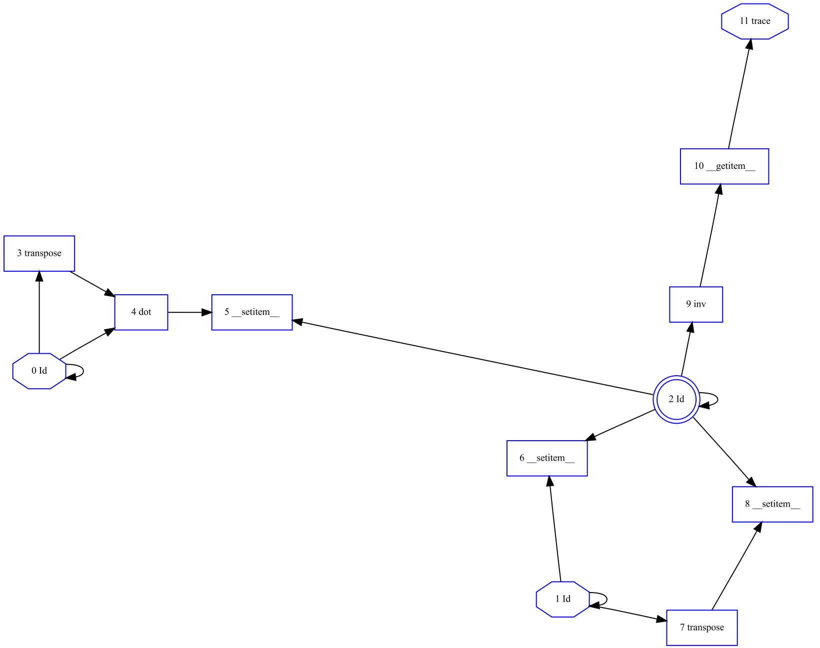
- Compute $\nabla_q \mathrm{eigh}(C(q))$, where

$$C \quad = \quad (\mathrm{I}, 0) \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathrm{I} \\ 0 \end{pmatrix} .$$

```
import numpy
from algopy import CGraph, Function, UTPM, dot, inv, zeros, eigh

def C(J1,J2):
    """generic implementation of the covariance computation"""
    Np = J1.shape[1];   Nr = J2.shape[0]
    tmp = zeros((Np+Nr, Np+Nr), dtype=J1)
    tmp[:Np,:Np] = dot(J1.T,J1)
    tmp[Np:,:Np] = J2
    tmp[:Np,Np:] = J2.T
    return inv(tmp)[:Np,:Np]

D,P,Nm,Np,Nr = 2,1,50,4,3
cg = CGraph()
J1 = Function(UTPM(numpy.random.rand(D,P,Nm,Np)))
J2 = Function(UTPM(numpy.random.rand(D,P,Nr,Np)))
Phi = Function.eigh(C(J1,J2))[0][0]
cg.independentFunctionList = [J1,J2]; cg.dependentFunctionList = [Phi]
cg.plot('pics/cgraph.svg')
```


11 trace
10 __getitem__
3 transpose
9 inv
4 dot
5 __setitem__
0 Id
2 Id
6 __setitem__
8 __setitem__
1 Id
7 transpose

Some Software for Forward/Reverse UTP

Name	Description	Status	LOC
algopy	forward/reverse UTPM in Python www.github.com/b45ch1/algopy	alpha	10388
pysolvind	Python Bindings to SolvIND/DAESOL-II	alpha	9743
pyadolc	Python Bindings to ADOL-C (C++) www.github.com/b45ch1/pyadolc	stable	6895
pycppad	Python Bindings to CppAD (C++) www.github.com/b45ch1/pycppad	stable	1334
taylorpoly	forward/reverse UTPS/UTPM (C) includes Python bindings www.github.com/b45ch1/taylorpoly	alpha	9276

LOC include unit tests but exclude comments (about 25% of the line count are comments)

- Summary:
    - Have a fairly complete set of useful tools in Python now
    - TAYLORPOLY hosts ANSI-C algorithms that can be used from basically all programming languages
- Outlook:
    - Reverse mode of *QR* decomposition of quadratic by singular matrices
    - Reverse mode of the symmetric eigenvalue decomposition for the case of repeated eigenvalues
    - derive UTPM algorithm for the Singular Value Decomposition and generalized eigenvalue decomposition
    - port all existing algorithms from ALGOPY to TAYLORPOLY