KU LEUVEN

OCP/MPC Workshop 2024
Nonlinear Programming

MISTI MIT Global Experiences

Wilm Decré
wilm.decre@kuleuven.be

Overview

- **nonlinear programming**
- interior-point methods
- computing derivatives using algorithmic differentiation

Nonlinear programming

$$\begin{aligned}&\underset{x\in\mathbb{R}^n}{\text{minimize}}\, f(x)\\ &\text{subject to}\\ &\begin{cases} g(x) = 0\\ h(x) \le 0\end{cases}\end{aligned}$$

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ \underset{\lambda\in\mathbb{R}^m,\, \nu\in\mathbb{R}^p,\, \nu\ge 0}{\text{maximize}}\ \underbrace{f(x) + \lambda^\intercal g(x) + \nu^\intercal h(x)}_{\mathcal{L}(x,\lambda,\nu)}$$

$\mathcal{L}(x,\lambda,\nu) \longleftarrow$ Lagrangian

if x^* is a locally optimal solution, then there exist λ^*, ν^* such that

$$\begin{cases}\nabla_x \mathcal{L}(x^*,\lambda^*,\nu^*) = 0\\ g(x^*) = 0\\ h(x^*) \le 0\\ \nu^* \ge 0\\ {\nu^*}^\intercal h(x^*) = 0,\end{cases}$$

(under some mild assumptions)
first-order necessary conditions for optimality, or KKT conditions

- no inequality constraints: system of nonlinear equations
- inequality constraints
  - active-set methods
    - iteratively find out which constraints are active and which ones are not (e.g., SQP-AS)
  - interior-point methods $\longleftarrow$ focus of today

Overview

- nonlinear programming
- **interior-point methods**
- computing derivatives using algorithmic differentiation

Interior-point method

"barrier" reformulation

primal-dual (PD) equations

$$\underset{x}{\text{minimize}}\, f(x) \quad \text{subject to} \quad \begin{cases} g(x) = 0 \\ h(x) \le 0 \end{cases} \longrightarrow \underset{x,s}{\text{minimize}}\, f(x) - \mu \sum_i \log s_i \quad \text{subject to} \quad \begin{cases} g(x) = 0 \\ h(x) + s = 0 \end{cases}$$

for $\mu \to 0$

$$\longrightarrow \begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ h(x^*) + s^* = 0 \\ [s^*]\nu^* - \mu e = 0 \end{cases}$$

with e a vector of ones

- we again obtain a system of nonlinear equations
- μ is adjusted over iterations, in order to (hopefully) converge to a solution of the original problem
- in this course we will use interior-point solver Ipopt

Interior-point method

$$\underset{x,y}{\text{minimize}}\ 2(x^2+y^2-1)-x$$
subject to
$$\begin{cases} x^2+y^2=1 \\ x \geq y \end{cases}$$

```
import casadi as cs

opti = cs.Opti()
x = opti.variable(2)

opti.minimize(2*(x[0]**2+x[1]**2-1)-x[0])
opti.subject_to(x[0]**2+x[1]**2-1==0)
opti.subject_to(x[0]>= 0.0)

p_opts = {'expand': True}
s_opts = {"print_level": 5}
solver = opti.solver('ipopt', p_opts, s_opts)
opti.set_initial(x,[0.8, 1.0])

sol = opti.solve()
print(sol.value(x))
```

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  4.8000000e-01 6.40e-01 6.10e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  2.1464177e-01 6.44e-01 4.51e-01  -1.7 1.20e+00    -  1.00e+00 2.50e-01f  3
   2 -1.3964056e-01 4.57e-01 5.53e-01  -1.7 1.03e+00    -  1.00e+00 1.00e+00F  1
   3 -5.3286086e-01 3.04e-01 3.80e-02  -1.7 5.45e-01    -  1.00e+00 1.00e+00f  1
   4 -9.7045257e-01 1.97e-02 1.39e-02  -1.7 1.31e-01    -  1.00e+00 1.00e+00h  1
   5 -9.9981472e-01 1.24e-04 2.24e-04  -3.8 9.71e-03    -  1.00e+00 1.00e+00h  1
   6 -9.9999997e-01 1.95e-08 3.59e-08  -5.7 1.25e-04    -  1.00e+00 1.00e+00h  1
   7 -1.0000000e+00 1.33e-15 9.83e-16  -8.6 3.59e-08    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 7

[...]

EXIT: Optimal Solution Found.
      solver  :   t_proc      (avg)   t_wall       (avg)    n_eval
       nlp_f  | 874.00us ( 87.40us)  53.29us (   5.33us)        10
       nlp_g  |   1.12ms (112.30us)  63.05us (   6.30us)        10
  nlp_grad_f  | 831.00us ( 92.33us)  51.09us (   5.68us)         9
  nlp_hess_l  | 530.00us ( 75.71us)  32.77us (   4.68us)         7
   nlp_jac_g  | 688.00us ( 76.44us)  43.00us (   4.78us)         9
       total  | 124.76ms (124.76ms)   7.79ms (   7.79ms)         1
```

- in the next slides we will highlight the main concepts needed to interpret this output
- disclaimer: we will not cover all advanced features and accelerating heuristics, but focus on main concepts

Ipopt (sketch)

$$\begin{array}{l}\underset{x}{\text{minimize}}\, f(x)\\ \text{subject to}\\ \begin{cases} g(x) = 0 \\ x \geq 0\end{cases}\end{array} \longrightarrow \begin{array}{l}\underset{x,s}{\text{minimize}}\, f(x) - \mu \sum_i \log s_i \\ \text{subject to}\\ \begin{cases} g(x) = 0 \\ x - s = 0\end{cases}\end{array} \quad \text{for } \mu \to 0 \quad \longrightarrow \begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 & \longleftarrow \text{dual feasibility} \\ g(x^*) = 0 & \longleftarrow \text{primal feasibility} \\ \lceil x^* \rfloor \nu^* - \mu e = 0 \end{cases}$$

- consider problem formulation above
- generalizing to unconstrained x_i for some i, lower + upper bounds, and general inequalities is easy
- further reading, cf. [2]

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\, f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$

$$\longrightarrow \begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \lceil x^* \rfloor \nu^* - \mu e = 0 \end{cases}$$

- current iterate: $x^{(i)}, \lambda^{(i)}, \nu^{(i)}$
- linearize PD equations to compute PD system

$$\begin{bmatrix} \nabla_{xx}\mathcal{L}^{(i)} & \nabla g^{(i)} & -I \\ \nabla g^{(i)\mathrm{T}} & 0 & 0 \\ \lceil \nu^{(i)} \rfloor & 0 & \lceil x^{(i)} \rfloor \end{bmatrix} \begin{pmatrix} p_x^{(i)} \\ p_\lambda^{(i)} \\ p_\nu^{(i)} \end{pmatrix} = - \begin{pmatrix} \nabla f^{(i)} + \nabla g^{(i)} \lambda^{(i)} - \nu^{(i)} \\ g^{(i)} \\ \lceil x^{(i)} \rfloor \nu^{(i)} - \mu e \end{pmatrix}$$

- for many problems, this *function evaluation* is an expensive step!
  - $\nabla_{xx}\mathcal{L}^{(i)}, \nabla g^{(i)}, \nabla f^{(i)}, g^{(i)}$

Overall algorithm sketch

```
 0: while no convergence of full problem
 1:   while no convergence of barrier subproblem
 2:     evaluate PD system at current iterate
 3:     compute search direction and initial trial point
 4:     if PD not acceptable, perform inertia correction
          and regularization
 5:     while trial point not accepted
 6:       perform SOC(s)
 7:         if trial point not accepted
 8:           backtrack line search
 9:         if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter $\mu$
```

```
[...]

EXIT: Optimal Solution Found.
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  | 874.00us ( 87.40us)  53.29us (  5.33us)        10
       nlp_g  |   1.12ms (112.30us)  63.05us (  6.30us)        10
  nlp_grad_f  | 831.00us ( 92.33us)  51.09us (  5.68us)         9
  nlp_hess_l  | 530.00us ( 75.71us)  32.77us (  4.68us)         7
   nlp_jac_g  | 688.00us ( 76.44us)  43.00us (  4.78us)         9
       total  | 124.76ms (124.76ms)   7.79ms (  7.79ms)         1
```

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\, f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$

$$\longrightarrow \begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \lceil x^* \rfloor \nu^* - \mu e = 0 \end{cases}$$

- solve linear system

$$\begin{bmatrix} \nabla_{xx}\mathcal{L}^{(i)} & \nabla g^{(i)} & -I \\ \nabla g^{(i)\mathrm{T}} & 0 & 0 \\ \lceil \nu^{(i)} \rfloor & 0 & \lceil x^{(i)} \rfloor \end{bmatrix} \begin{pmatrix} p_x^{(i)} \\ p_\lambda^{(i)} \\ p_\nu^{(i)} \end{pmatrix} = - \begin{pmatrix} \nabla f^{(i)} + \nabla g^{(i)} \lambda^{(i)} - \nu^{(i)} \\ g^{(i)} \\ \lceil x^{(i)} \rfloor \nu^{(i)} - \mu e \end{pmatrix}$$

$$\begin{cases} \begin{bmatrix} \nabla_{xx}\mathcal{L}^{(i)} + \Sigma^{(i)} & \nabla g^{(i)} \\ \nabla g^{(i)\mathrm{T}} & 0 \end{bmatrix} \begin{pmatrix} p_x^{(i)} \\ p_\lambda^{(i)} \end{pmatrix} = - \begin{pmatrix} \nabla f^{(i)} - \mu \lceil x^{(i)} \rfloor^{-1} e + \nabla g^{(i)} \lambda^{(i)} \\ g^{(i)} \end{pmatrix} \\ p_\nu^{(i)} = \mu \lceil x^{(i)} \rfloor^{-1} e - \nu^{(i)} - \Sigma^{(i)} p_x^{(i)} \end{cases}$$

with $\Sigma^{(i)} = \lceil x^{(i)} \rfloor^{-1} \lceil \nu^{(i)} \rfloor$

Overall algorithm sketch

```
0:  while no convergence of full problem
1:    while no convergence of barrier subproblem
2:      evaluate PD system at current iterate
3:      compute search direction and initial trial point
4:      if PD not acceptable, perform inertia correction
          and regularization
5:      while trial point not accepted
6:        perform SOC(s)
7:          if trial point not accepted
8:            backtrack line search
9:          if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter $\mu$
```

- set initial values for $\alpha, \alpha_\nu \in (0,1]$
- initial trial point:
  - $x^{(i+1)} \leftarrow x^{(i)} + \alpha p_x^{(i)}$
  - $\lambda^{(i+1)} \leftarrow \lambda^{(i)} + \alpha p_\lambda^{(i)}$
  - $\nu^{(i+1)} \leftarrow \nu^{(i)} + \alpha_\nu p_\nu^{(i)}$

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\, f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$

$$\longrightarrow \begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \lceil x^* \rfloor \nu^* - \mu e = 0 \end{cases}$$

- in order to have a unique solution and descent direction:
  - reduced Hessian must be positive definite
  - PD system must be nonsingular

- inertia correction and regularization

$$\begin{bmatrix} \nabla_{xx}\mathcal{L}^{(i)} + \Sigma^{(i)} + \delta_w I & \nabla g^{(i)} \\ \nabla g^{(i)\mathrm{T}} & -\delta_c I \end{bmatrix} \begin{pmatrix} p_x^{(i)} \\ p_\lambda^{(i)} \end{pmatrix} = - \begin{pmatrix} \nabla f^{(i)} - \mu \lceil x^{(i)} \rfloor^{-1} e + \nabla g^{(i)} \lambda^{(i)} \\ g^{(i)} \end{pmatrix}$$

for some $\delta_w, \delta_c \geq 0$

- compute search direction and initial trial point using this modified system

Overall algorithm sketch

```
 0: while no convergence of full problem
 1:   while no convergence of barrier subproblem
 2:     evaluate PD system at current iterate
 3:     compute search direction and initial trial point
 4:     if PD not acceptable, perform inertia correction
          and regularization
 5:     while trial point not accepted
 6:       perform SOC(s)
 7:         if trial point not accepted
 8:           backtrack line search
 9:         if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter $\mu$
```

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\, f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$

$$\longrightarrow \begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \lceil x^* \rfloor \nu^* - \mu e = 0 \end{cases} \longrightarrow$$

- Ipopt uses a filter technique to check if a trial point is acceptable
- basic concept
  - filter contains points that do not *dominate* each other, meaning that not <u>both</u> the objective function and the constraint violation of one point are lower than any other one
  - a new iterate is acceptable to the filter if is not dominated by any point in the filter

Overall algorithm sketch

```
 0: while no convergence of full problem
 1:   while no convergence of barrier subproblem
 2:     evaluate PD system at current iterate
 3:     compute search direction and initial trial point
 4:     if PD not acceptable, perform inertia correction
            and regularization
 5:     while trial point not accepted
 6:       perform SOC(s)
 7:         if trial point not accepted
 8:           backtrack line search
 9:         if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter $\mu$
```

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$

$$\longrightarrow \begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \lceil x^* \rfloor \nu^* - \mu e = 0 \end{cases}$$

- second-order correction if trial step: $\tilde{p}_x^{(i)}$ has been rejected and constraint violation increases for this step
- goal: try to reduce infeasibility
- compute correction for the search direction that satisfies $\nabla g^{(i)}\, p_{x,\text{soc}}^{(i)} + g(x^{(i)} + \tilde{p}_x^{(i)}) = 0$
- corrected search direction then becomes

$$\hat{p}_x^{(i)} = \tilde{p}_x^{(i)} + p_{x,\text{soc}}^{(i)}$$

Overall algorithm sketch

```
0:  while no convergence of full problem
1:    while no convergence of barrier subproblem
2:      evaluate PD system at current iterate
3:      compute search direction and initial trial point
4:      if PD not acceptable, perform inertia correction
           and regularization
5:      while trial point not accepted
6:        perform SOC(s)
7:          if trial point not accepted
8:            backtrack line search
9:          if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter $\mu$
```

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$

$$\longrightarrow \begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \lceil x^* \rfloor \nu^* - \mu e = 0 \end{cases}$$

- backtrack line search
  - $\alpha \leftarrow \frac{1}{2}\alpha$
  - $x^{(i+1)} \leftarrow x^{(i)} + \alpha p_x^{(i)}$
  - $\lambda^{(i+1)} \leftarrow \lambda^{(i)} + \alpha p_\lambda^{(i)}$
  - $\nu^{(i+1)} \leftarrow \nu^{(i)} + \alpha_\nu p_\nu^{(i)}$

Overall algorithm sketch

```
 0: while no convergence of full problem
 1:   while no convergence of barrier subproblem
 2:     evaluate PD system at current iterate
 3:     compute search direction and initial trial point
 4:     if PD not acceptable, perform inertia correction
            and regularization
 5:     while trial point not accepted
 6:       perform SOC(s)
 7:         if trial point not accepted
 8:           backtrack line search
 9:         if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter $\mu$
```

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\, f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$

$$\longrightarrow \begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \lceil x^* \rfloor \nu^* - \mu e = 0 \end{cases}$$

- feasibility restoration phase
  - restore feasibility <u>or</u> detect local infeasibility
- regular and alternative method
- main idea of regular method:

$$\underset{x}{\text{minimize}} \|g(x)\|_1 + \frac{\zeta}{2}\|D_R(x - x_R)\|_2^2$$

- main idea of alternative method: check that new trial point satisfies primal–dual equations sufficiently better than current iterate

Overall algorithm sketch

```
 0: while no convergence of full problem
 1:   while no convergence of barrier subproblem
 2:     evaluate PD system at current iterate
 3:     compute search direction and initial trial point
 4:     if PD not acceptable, perform inertia correction
            and regularization
 5:     while trial point not accepted
 6:       perform SOC(s)
 7:         if trial point not accepted
 8:           backtrack line search
 9:           if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter $\mu$
```

$$\underset{x,y}{\text{minimize}}\ 2(x^2+y^2-1)-x$$
subject to
$$\begin{cases} x^2+y^2=1 \\ x \geq 0 \end{cases}$$

Ipopt (sketch)

now we can interpret the main parts of the Ipopt output

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  4.8000000e-01 6.40e-01 7.58e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1 -9.3019474e-02 5.47e-01 4.70e-01  -1.7 6.30e-01    -  6.78e-01 1.00e+00f  1
   2 -5.6304824e-01 1.96e-01 4.98e-01  -1.7 8.21e-01    -  1.00e+00 1.00e+00F  1
   3 -9.3469596e-01 4.28e-02 4.08e-02  -1.7 5.08e-01    -  1.00e+00 1.00e+00F  1
   4 -9.9696688e-01 2.02e-03 6.93e-05  -2.5 4.06e-02    -  1.00e+00 1.00e+00h  1
   5 -9.9999846e-01 1.02e-06 2.03e-06  -3.8 1.01e-03    -  1.00e+00 1.00e+00h  1
   6 -1.0000000e+00 2.82e-13 7.84e-11  -5.7 5.12e-07    -  1.00e+00 1.00e+00h  1
   7 -1.0000000e+00 0.00e+00 2.51e-14  -8.6 2.12e-11    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 7
```

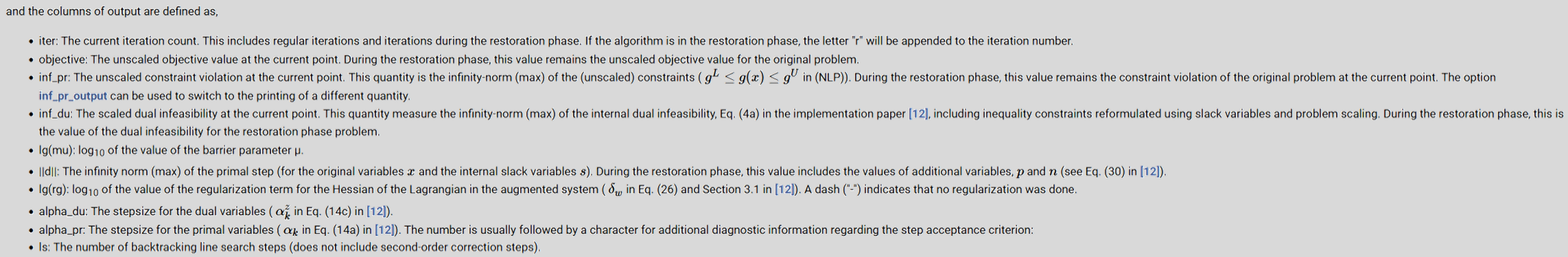
```python
import casadi as cs

opti = cs.Opti()
x = opti.variable(2)

opti.minimize(2*(x[0]**2+x[1]**2-1)-x[0])
opti.subject_to(x[0]**2+x[1]**2-1==0)
opti.subject_to(x[0]>= 0.0)

p_opts = {'expand': True}
s_opts = {"print_level": 5}
solver = opti.solver('ipopt', p_opts, s_opts)
opti.set_initial(x,[0.8, 1.0])

sol = opti.solve()
print(sol.value(x))
```


and the columns of output are defined as,

- iter: The current iteration count. This includes regular iterations and iterations during the restoration phase. If the algorithm is in the restoration phase, the letter "r" will be appended to the iteration number.
- objective: The unscaled objective value at the current point. During the restoration phase, this value remains the unscaled objective value for the original problem.
- inf_pr: The unscaled constraint violation at the current point. This quantity is the infinity-norm (max) of the (unscaled) constraints ($g^L \leq g(x) \leq g^U$ in (NLP)). During the restoration phase, this value remains the constraint violation of the original problem at the current point. The option inf_pr_output can be used to switch to the printing of a different quantity.
- inf_du: The scaled dual infeasibility at the current point. This quantity measure the infinity-norm (max) of the internal dual infeasibility, Eq. (4a) in the implementation paper [12], including inequality constraints reformulated using slack variables and problem scaling. During the restoration phase, this is the value of the dual infeasibility for the restoration phase problem.
- lg(mu): $\log_{10}$ of the value of the barrier parameter μ.
- ||d||: The infinity norm (max) of the primal step (for the original variables x and the internal slack variables s). During the restoration phase, this value includes the values of additional variables, p and n (see Eq. (30) in [12]).
- lg(rg): $\log_{10}$ of the value of the regularization term for the Hessian of the Lagrangian in the augmented system (δ_w in Eq. (26) and Section 3.1 in [12]). A dash ("-") indicates that no regularization was done.
- alpha_du: The stepsize for the dual variables (α_k^z in Eq. (14c) in [12]).
- alpha_pr: The stepsize for the primal variables (α_k in Eq. (14a) in [12]). The number is usually followed by a character for additional diagnostic information regarding the step acceptance criterion:
- ls: The number of backtracking line search steps (does not include second-order correction steps).

- more information available at https://coin-or.github.io/Ipopt/OUTPUT.html

Overview

- nonlinear programming
- interior-point methods
- **computing derivatives using algorithmic differentiation**

Computing derivatives

recall: evaluating the PD system at the current iterate

$$\begin{bmatrix} \nabla_{xx}\mathcal{L}^{(i)} & \nabla g^{(i)} & -I \\ {\nabla g^{(i)}}^{\mathrm{T}} & 0 & 0 \\ \lceil \nu^{(i)} \rfloor & 0 & \lceil x^{(i)} \rfloor \end{bmatrix} \begin{pmatrix} p_x^{(i)} \\ p_\lambda^{(i)} \\ p_\nu^{(i)} \end{pmatrix} = -\begin{pmatrix} \nabla f^{(i)} + \nabla g^{(i)}\lambda^{(i)} - \nu^{(i)} \\ g^{(i)} \\ \lceil x^{(i)} \rfloor \nu^{(i)} - \mu e \end{pmatrix}$$

for many problems, this function evaluation is an expensive step!

- $\nabla_{xx}\mathcal{L}^{(i)}, \nabla g^{(i)}, \nabla f^{(i)}, g^{(i)}$

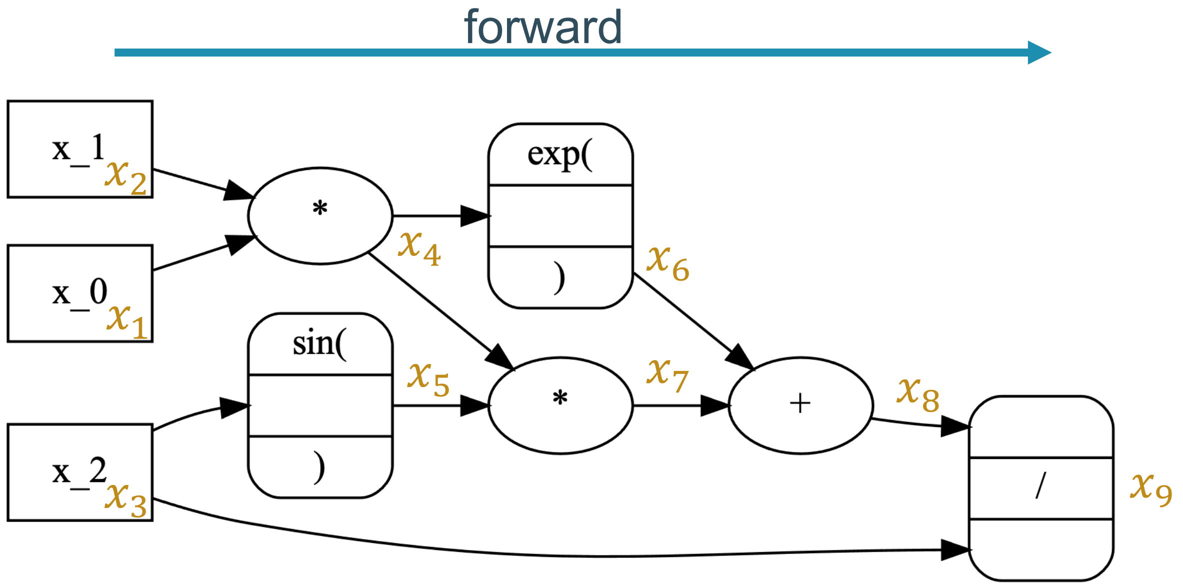
- **automatic differentiation – our main workhorse**
    - break down code into a composition of elementary arithmetic operations and apply the chain rule to produce the derivatives
- **numerical differentiation**
    - finite differences (FD)
    - often expensive and limited accuracy (truncation error + rounding error)

Automatic differentiation

- also known as: algorithmic differentiation, auto-diff, AD
- key idea: apply chain rule of calculus:
  - $z = f(y) = f(g(x))$
  - $\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x}$
- example (based on [1]): $f(x) = \frac{x_1 x_2 \sin(x_3) + e^{x_1 x_2}}{x_3}$
- we evaluate f, broken down in arithmetic terms

- $x_1 = x_1$
- $x_2 = x_2$
- $x_3 = x_3$
- $x_4 = x_1 x_2$
- $x_5 = \sin(x_3)$
- $x_6 = e^{x_4}$
- $x_7 = x_4 x_5$
- $x_8 = x_6 + x_7$
- $x_9 = x_8 / x_3$

Automatic differentiation – evaluating function



- $x_1 = x_1$
- $x_2 = x_2$
- $x_3 = x_3$
- $x_4 = x_1 x_2$
- $x_5 = \sin(x_3)$
- $x_6 = e^{x_4}$
- $x_7 = x_4 x_5$
- $x_8 = x_6 + x_7$
- $x_9 = x_8 / x_3$

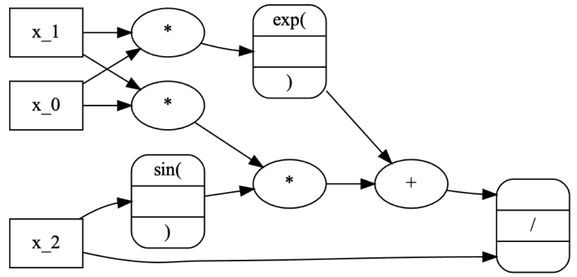
```
from casadi import *
from casadi.tools import * # check that you have
pydot installed

x = SX.sym('x', 3)
y = (x[0]*x[1]*sin(x[2])+exp(x[0]*x[1]))/x[2]

y = cse(y)

graph = dotdraw(y,direction="LR")
dotsave(y,format="svg", direction="LR")
```

`y = cse(y)` - common subexpression elimination
If not done, we get:



to compute the Jacobian of $f(x)$, we apply the chain rule, either inside-out (forward) or outside-in (reverse)

Automatic differentiation – forward mode AD

- define directional derivative along *seed vector* p
  - $D_p x_i = (\nabla x_i)^\intercal p = \sum_{j=1}^{3} \frac{\partial x_i}{\partial x_j} p_j$
- we then have $D_p x_9 = \nabla f(x)^\intercal p$
- we traverse forward, and, once we have x_i, we compute $D_p x_i$ using the chain rule
- example: $p = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$
  - $D_p x_1 = 1$, $D_p x_2 = 0$, $D_p x_3 = 0$

- $x_1 = x_1$
- $x_2 = x_2$
- $x_3 = x_3$
- $x_4 = x_1 x_2$
- $x_5 = \sin(x_3)$
- $x_6 = e^{x_4}$
- $x_7 = x_4 x_5$
- $x_8 = x_6 + x_7$
- $x_9 = x_8 / x_3$

Automatic differentiation – forward mode AD

- example: $p = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$
  - $D_p x_1 = 1, D_p x_2 = 0, D_p x_3 = 0$
  - $D_p x_4 = \frac{\partial x_4}{\partial x_1} D_p x_1 + \frac{\partial x_4}{\partial x_2} D_p x_2 = x_2$
  - $D_p x_5 = \frac{\partial x_5}{\partial x_3} D_p x_3 = 0$
  - $D_p x_6 = \frac{\partial x_6}{\partial x_4} D_p x_4 = e^{x_4}\, x_2$
  - $D_p x_7 = \frac{\partial x_7}{\partial x_4} D_p x_4 + \frac{\partial x_7}{\partial x_5} D_p x_5 = x_5 x_2$
  - $D_p x_8 = \frac{\partial x_8}{\partial x_6} D_p x_6 + \frac{\partial x_8}{\partial x_7} D_p x_7 = e^{x_4}\, x_2 + x_5 x_2$
  - $D_p x_9 = \frac{\partial x_9}{\partial x_3} D_p x_3 + \frac{\partial x_9}{\partial x_8} D_p x_8 = \frac{e^{x_4}\, x_2 + x_5 x_2}{x_3} (= \frac{e^{x_1 x_2}\, x_2 + \sin(x_3) x_2}{x_3})$

- $x_1 = x_1$
- $x_2 = x_2$
- $x_3 = x_3$
- $x_4 = x_1 x_2$
- $x_5 = \sin(x_3)$
- $x_6 = e^{x_4}$
- $x_7 = x_4 x_5$
- $x_8 = x_6 + x_7$
- $x_9 = x_8 / x_3$

Automatic differentiation – forward mode AD

- for $f(x)\ \mathbb{R}^n \rightarrow \mathbb{R}^m$
- to compute the full (dense) Jacobian we must traverse n times, i.e., use a seed vector for every input
- if there is sparsity, we can exploit this by choosing the seed vectors in a smart way!
  (CasADi does this for you)

- $x_1 = x_1$
- $x_2 = x_2$
- $x_3 = x_3$
- $x_4 = x_1 x_2$
- $x_5 = \sin(x_3)$
- $x_6 = e^{x_4}$
- $x_7 = x_4 x_5$
- $x_8 = x_6 + x_7$
- $x_9 = x_8 / x_3$

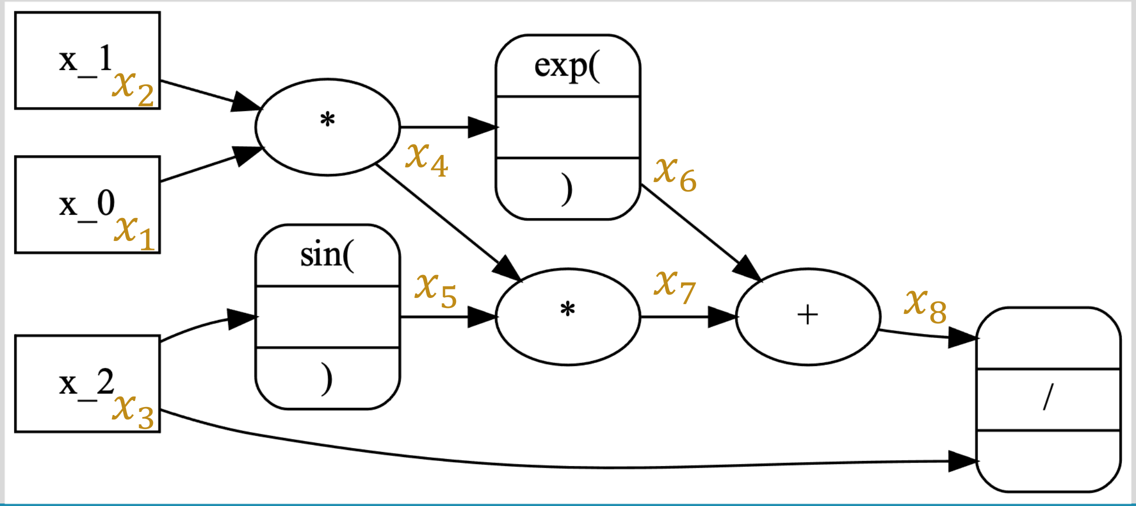
Automatic differentiation – reverse mode AD

- we work *reversely* compared to the function evaluation
- define adjoint variables $\bar{x}_i = \frac{\partial f}{\partial x_i} = \sum_{j \text{ a child of } i} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i}$
- start with all $\bar{x}_i = 0$
- set $\bar{x}_9 = 1$ (x_9 contains the final function value of f)
- as soon as it becomes known: $\bar{x}_i \mathrel{+}= \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i}$
- we traverse reverse, i.e., from children to parents
- once all children of node i have been visited, it is declared finalized
- to compute the full (dense) Jacobian we must traverse m times, i.e., use a seed vector for every output (in this case there is only one output, $\bar{x}_9 = 1$)
- reverse mode typically computationally more interesting if $n \gg m$
- entire computational graph must be stored, can require lots of memory! To reduce storage requirements, at the cost of extra arithmetic (partial reevaluations / forward-backward sweeps), *checkpointing* or *rematerialization* techniques are used

Automatic differentiation – reverse mode AD

- $x_1 = x_1$
- $x_2 = x_2$
- $x_3 = x_3$
- $x_4 = x_1 x_2$
- $x_5 = \sin(x_3)$
- $x_6 = e^{x_4}$
- $x_7 = x_4 x_5$
- $x_8 = x_6 + x_7$
- $x_9 = x_8 / x_3$

- $\bar{x}_9 = 1$ (finalized), $\bar{x}_i = 0$ for $i = 1..8$
- $$\begin{cases} \bar{x}_8 \mathrel{+}= \dfrac{\partial f}{\partial x_9}\dfrac{\partial x_9}{\partial x_8} = \bar{x}_9 \dfrac{\partial x_9}{\partial x_8} = \dfrac{1}{x_3} \text{ (finalized)} \\ \bar{x}_3 \mathrel{+}= \dfrac{\partial f}{\partial x_9}\dfrac{\partial x_9}{\partial x_3} = -\dfrac{x_8}{(x_3)^2} \end{cases}$$
- $$\begin{cases} \bar{x}_7 \mathrel{+}= \dfrac{\partial f}{\partial x_8}\dfrac{\partial x_8}{\partial x_7} = \dfrac{1}{x_3} \text{ (finalized)} \\ \bar{x}_6 \mathrel{+}= \dfrac{\partial f}{\partial x_8}\dfrac{\partial x_8}{\partial x_6} = \dfrac{1}{x_3} \text{ (finalized)} \end{cases}$$
- … try it yourself!



Automatic differentiation

- interesting video https://www.youtube.com/watch?v=twTIGuVhKbQ
- CasADi chooses mode (you can override) and has strategies to exploit sparsity
- CasADi code:

```
x = SX.sym('x', 3)
y = (x[0]*x[1]*sin(x[2])+exp(x[0]*x[1]))/x[2]
y = cse(y)
options = {"enable_forward":
True,"enable_reverse": True}
f = Function('f', [x], [y], options)
J = f.jacobian()
print(f(x))
print(f.n_instructions())
print(J.n_instructions())
```

computing Jacobian-times-vector or Jacobian-transposed-times-vector:

```
jtimes(y,x,[1,0,0],False)
jtimes(y,x,[1],True)
```

References and further reading

- [1] Nocedal, J. and Wright, S.J. (2006). *Numerical optimization.* New York: Springer. Available at https://link.springer.com/book/10.1007/978-0-387-40065-5
- [2] Wächter, A., Biegler, L. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.* 106, 25–57 (2006). https://doi.org/10.1007/s10107-004-0559-y
- [3] CasADi Docs - https://web.casadi.org/docs